VIEW

Exploratory Molecular Visualization System

TR93-30

January, 1993

Lawrence D. Bergman

Department of Computer Science
The University of North Carolina
Chapel Hill, NC 27599-3175

This work was supported by the National Center for Research Resources, National Institutes of Health.

VIEW

Table of Contents

Installing the VIEW System via ftp

Larry Bergman

2/16/93

The VIEW system is available on anonymous ftp and consists of a set of software and a set of data. The VIEW software requires about 20 megabytes of disk space to install. The data (including Pdb files, VIEW database files, geometry files, drawing tools, and snapshot files) also requires about 20 megabytes. Once installation is completed, and intermediate files are removed, the system software will occupy about 10 megabytes, and the VIEW database files will also occupy about 10 megabytes.

VIEW is installed in three steps. First the files are obtained through anonymous ftp. Next, the system software is installed, and finally, the data is installed.

OBTAINING THE INSTALLATION FILES

1) Go to the directory where you intend to install the VIEW system software.

2) Connect to UNC's anonymous ftp. Type:

   ftp ftp.cs.unc.edu

   Ftp will prompt you for a name. Type in: **anonymous**
   Ftp will next prompt you for a password. Type in your e-mail address
   Ftp will respond with the prompt: ftp>

3) Set the transfer mode to binary by typing: **binary**

4) Go to the directory where the VIEW files are located by typing:

   cd pub/VIEW

3) Retrieve the VIEW installation files. There are three files to be retrieved. Each time you get an ftp prompt, you can type the next retrieval command. Type:

   get install_view
   get view_executables.Z
   get view_data.Z

   The first retrieval will be very quick; the last two may require several minutes each.

4) Exit ftp by typing: **quit**

INSTALLING THE SYSTEM SOFTWARE

1) Mark the installation script executable by typing:

chmod 777 install_view

2) Run the installation script. Type:

install_view

The installation will require about 1 minute. No error or warning messages should be generated during installation. Call me if you note any difficulties.

INSTALLING THE SYSTEM DATA

Run the data installation script. Type:

install_view_data -directory ***view_data_directory***

where *view_data_directory* is the full pathname of the directory where the VIEW data is to be located. The data does not need to be in the same location as the system software, although *view_system_directory/*data is a common choice.

Be sure that *view_data_directory* exists prior to executing this script. The data installation script will report on its progress as it installs the data. The installation will require about 2 minutes.

Once the data has been successfully installed, you can delete the files *install_view, install_view_data* (created by the installation procedure, *view_data* and *view_executables* (note that the installation procedure removes the .Z suffix from the last two files).

CREATING A NEW DATA AREA

You can create a new data area as follows. This procedure will be particularly useful for other users that want to use VIEW from their own data areas. This setup procedure creates a VIEW subdirectory structure in a specified location, and creates links to the system data (tools, geometry, pdb files, snapshots).

1) Create your VIEW data directory (*your_data_directory*).

2) Go to the VIEW system directory. Type:

cd ***view_system_directory***

3) Run the data directory creation script. Type:

create_view_data_directory -directory ***your_data_directory***

PREPARING TO RUN VIEW

Once the system and data have been installed, you must execute your start-up shell file. Type:

source ***your_data_directory/*****setupview**

or

source ***view_data_directory/*****setupview**

if you intend running in the system data area.

This will set up the VIEW command names. You will need to source the **setupview** script in each window you intend to run VIEW from. You may wish to do this automatically from your .cshrc.

You are now ready to run VIEW using the commands described in the document Running the VIEW System.

PROBLEMS WITH INSTALLATION

If you run into difficulties installing VIEW, please call me at (919)962-1964 or 1976, or send me e-mail at bergman@cs.unc.edu.

VIEW

Exploratory Molecular Visualization System

Overview

Table of Contents

VIEW Exploratory Molecular Visualization System Overview

Larry Bergman
1/27/93

1. INTRODUCTION

The VIEW system is designed for exploratory visualization of molecules, particularly macromolecules such as proteins. The system, which runs on IRIS VGX, GTX, Indigo and Crimson series machines, includes:

- a 3-D geometry viewer
- a library of visualization tools
- facilities for customizing the tools or designing your own.

Exploration of a molecule begins with a simple representation, for example, a vector representation of the backbone of a portion of a protein (Figure 1).

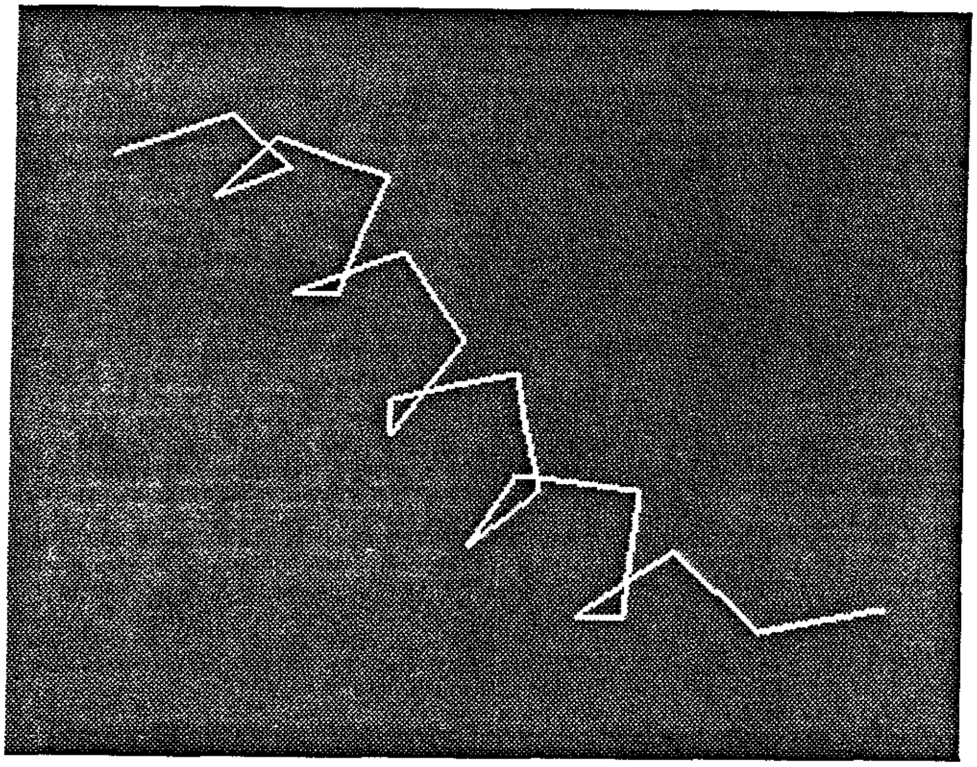


Figure 1

A vector representation of an alpha helix used to start visualization

Using a series of *drawing tools* you can sketch in a variety of visual representations of portions of the molecule. You might invoke a tool that draws sidechains bonds as cylinders, a tool to change the radius or color of the sidechains, a tool that draws a cylinder to fit a helix axis, and finally, a tool that allows you to rotate an individual sidechain or a portion of the main chain. A visualization that might be produced using these tools is shown in figure 2.

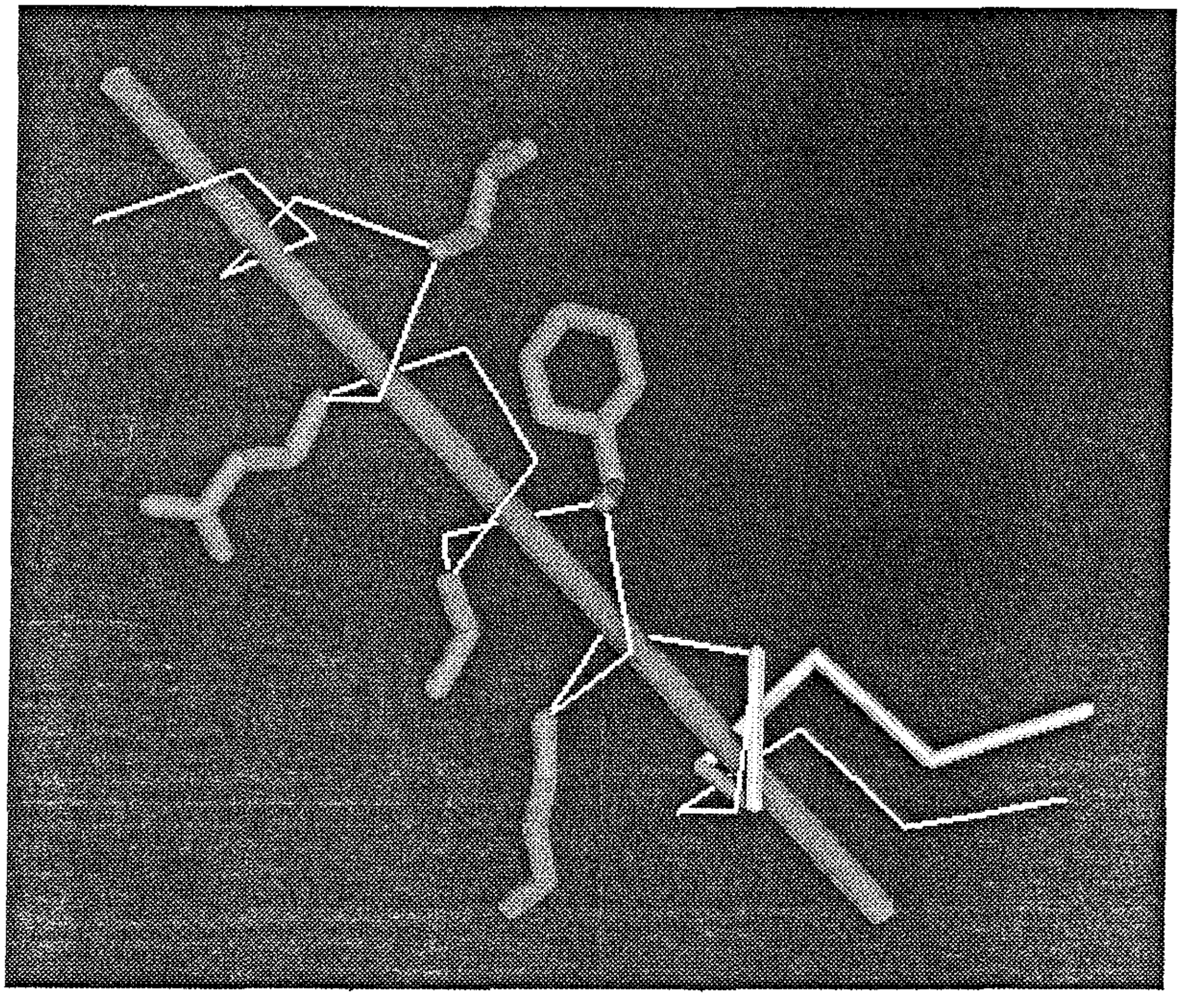


Figure 2

A VIEW visualization

The library contains over fifty ready-build tools for creating and altering elements of a visualization. The VIEW system also has a facility for modifying existing tools or developing new ones. The tools are written in a simplified C-like programming language with special constructs for molecular database access and modification; creation, modification, and management of 3-D geometry; and interaction with the 3-D image. The system includes debugging facilities for aiding the modification of existing drawing tools and development of new tools.

This overview describes the background required to use the VIEW system, continues with a tutorial example of use of the system, and concludes with an introduction to some VIEW-specific terminology that is used throughout the user's manual.

2. INTENDED AUDIENCE

The VIEW system is designed for use by biochemists or computer programmers who have some familiarity with chemical/molecular concepts and terminology. The user should be familiar with SGIs, in particular, use of the SGIs' windowing system (mwm).

There are two levels at which VIEW may be employed. Using the supplied library of visualization (drawing) tools requires some familiarity with computer-based molecular graphics systems, but no programming background is assumed (this is the background assumed for the document, *VIEW User Interface Description*). If you wish to modify the tools or develop new ones, familiarity with a traditional programming language such as Basic, FORTRAN, or (particularly) C will be helpful. In addition, some fundamental computer graphics concepts (such as transformations and shading), and familiarity with 3-D constructive geometry and vector algebra will be required for you to develop tools effectively; this background is assumed for readers of the document, *VIEW Drawing Tool Definition Language – Language Description*. Use of the tool development environment requires some experience with interactive debuggers such as dbx; such experience is assumed for readers of the document, *VIEW Drawing Tool Definition Language – Development Environment*

3. USING THE VIEW SYSTEM - AN EXAMPLE

This example will trace the steps that you would go though in drawing a representation of the backbone of a protein molecule and several side chains. It is presented as a tutorial; you will find this most useful if you actually follow these steps using the VIEW system. The steps in this tutorial are demonstated in the video tape, *An Introduction to the VIEW Exploratory Molecular Visualization System.*

Initialize the VIEW system by typing "iview". You will see an empty display panel with a dark gray background on the left in which 3-D molecular geometry will be displayed, a control panel on the right labeled "Main panel", and a small panel in the upper right labeled "Tool executing".

Position the mouse cursor over the button labeled **Read** in the **Group Operations** area of the main panel and click with any of the mouse buttons (the right mouse button is used for picking in the display panel; you may want to get in the habit of using the right mouse button for selection operations). You will see a *file listing* appear, listing all geometry files that are available in alphabetical order. Scroll through this list by positioning the mouse cursor over the inner gray rectangle in the scroll area, to the left of the list of file names. While holding down any mouse button, move the mouse up or down. Position the cursor over the entry "single_helix" in the file list and click with the mouse. The entry will be highlighted in brown. Click on the **Read** button in the file list. The file list will disappear, and a message saying "loading geometry file: single_helix" will appear at the top of the screen. This message will disappear shortly (while the message is displayed, the user interface of the system is deactivated), and a line drawing showing connections between alpha carbons for a single helix of a protein will be drawn in the display panel. The atoms are represented by positions where line segments come together.

Now you are ready to use drawing tools to sketch some geometry. Click on the **All tools** button in the **Drawing Tools** area of the main panel. A file list of all available library tools will appear. These tools and their use is described in the document, *VIEW Drawing Tool Library Description*. Click on the tool ***cyl_line_ca*** with the mouse. Now click on the button labeled **Execute**. This will start the tool. The tool will be running when you see its name appear in the small panel labeled "Tool Executing". Additionally, the message "Select start atom" will appear at the top of the screen. Each selection of geometry required by a tool is accompanied by a message of this form (although the text of each message will vary depending on the tool). At this point, the tool is waiting for you to pick an atom on-screen.

The tool that you have selected allows you to select a starting and ending atom position, and will then draw a series of cylinders between successive alpha carbons connecting these positions. Position the mouse cursor over an atom position near the left-hand end of the helix and click with the RIGHT mouse button. You will know that you have picked successfully if a red sphere appears at the atom position. Repeat this, selecting an atom near the right-hand end of the helix. After the second sphere has appeared, the connecting cylinders will be drawn. Holding the middle mouse button down, with the cursor inside the display panel, you may use the virtual trackball (described in Section 2.1.2 of the document*VIEW User Interface Description*) to rotate the geometry.

Now, draw in a few side chains. Execute the***sidechain_bonds***. tool. This tool will draw the side chain that is associated with each main-chain atom that you select. Note that the name in the **Tool executing** panel is now "sidechain_bonds". Executing this tool caused an exit of the previously executing tool (*cyl_line_ca*); only one tool may execute at a time (with the exception of tool-defined events, described in section 4.4). Select a few atom positions (in the same manner as you did previously, using the right mouse button), and watch the tool draw in a side-chain at each atom you select. At any point you may remove the last drawn sidechain by clicking the button labeled **Undo** in the main panel. Try drawing three or four sidechains and then removing them all using **Undo**. Note that after undo, you may continue drawing sidechains. Draw a few more sidechains for later use in this example.

The next step is to change the radius of the sidechains and the color of the backbone. To facilitate selecting the backbone, turn off the display of the original stick drawing (which is now mostly inside the cylinders). To do this, click on **Display** in the **Group Operations** subpanel. A *button panel* will appear that will list three *geometry* *groups*, "single_helix" (the geometry that we started with), "cyl_line_ca" (produced by the drawing tool of that name), and "sidechain_bonds". Try clicking on these buttons with the mouse. Note that groups that are displayed have a yellow light in their button. Now toggle off the group "single_helix", but leave the other two groups on. Close the **Group display** panel by clicking on **Close**.

Select and execute the tool ***change_radius_group***. Click on one of the sidechains. A *query* will appear at the top of the screen showing you the radius of the cylinder you selected (the value will be 0.2 Angstroms). Change the value by positioning the mouse cursor within the text box, hitting the backspace key (to delete the 2) and then typing a 3, changing the value to 0.3. Press "return" or click on **OK** when the value suits you. The query will disappear. Now click the mouse anywhere on any of the sidechains (indicating to the drawing tool which group is to be modified). You will see the sidechains all change size.

Now, select and execute the tool ***recolor_group***. Select an object with an initial color that you wish to modify (select a yellow cylinder on the backbone). The color (255,155,55) (red = 255, blue = 155, green = 55) will be displayed in a query. You may modify each field as before by positioning the mouse cursor within the text area (be sure that each color component is between 0 and 255). Change the color, and then click on **OK**. Now pick with the mouse on any element of the group that is be recolored (any backbone cylinder). The entire backbone group will change to the color that you have specified. You may undo the color change using the **Undo** button.

In the final portion of this tutorial you will rotate a single sidechain. Begin by deleting the sidechains drawn previously. Click on **Remove** in the **Group Operations** subpanel. A **Group remove** panel will appear with a button for each group. Click on "sidechain_bonds". The button will highlight in yellow indicating that this group has been selected for removal. Now click on **OK** at the bottom of the panel. The group will be removed from the panel and from the display. Close the Group remove panel by clicking on **Close**.

Draw a single sidechain using the ***sidechain_bonds*** tool. Next, execute the ***rotate_axis*** tool. This tool allows you to select a rotation axis and then specify one or more groups that are to rotate around that axis. Begin by selecting the alpha-beta carbon bond of the sidechain (the bond that connects the sidechain to the mainchain). You will see half of the bond turn red indicating a succcessful selection (all bonds are drawn as half-bonds to allow association of atom records from the database with each half). Now select any portion of the sidechain, specifying it as the group to rotate. Again the selected geometry will turn red. If you select from the wrong group at any time, you may press **Undo**, and restart specification for the tool.

Now, you may rotate the sidechain using keyboard *events* (described in section 4.4). Pressing the "r" or "e" keyboard keys will rotate the sidechain. See what happens if you hold one of these keys down. Pressing the "s" key will produce a query that allows you to change the amount of rotation for each key press.

This simple example should give you a taste of the VIEW interface and how drawing tools are used. To see further examples of the drawing tools, their use, and other VIEW interface features, view the video tapes, *VIEW, a System for Exploring Molecular Structure,* and *An Introduction to the VIEW Exploratory Molecular Visualization System.* For a description of the tools in the tool library, see *VIEW Drawing Tool Library Description.*

4. VIEW BASIC CONCEPTS

In these section, we present some of the fundamental terminology and concepts that are used in other sections of the User's Manual.

4.1 Geometry

Molecules are represented in VIEW using a few simple 3-D geometric forms. The forms of *display geometry* available are:

- cylinders
- lines
- Spheres
- 3-D text
- triangles

Geometry may produced using drawing tools (described below), or may be read from previous-generated *geometry files.*

4.2 Geometry Groups

Items of display geometry may be collected together into *geometry groups.* You may manipulate a group in a number of ways. You may turn off its display (causing it to become invisible), delete it from the system, operate on it as an entity using drawing tools (for example, coloring all items in the group), read or write it from/to a geometry file.

Drawing tools often create groups that have the name of the tool. Thus, the ***triangle_atoms*** drawing tool will create a group with the name "triangle_atoms" which will contain all geometry generated by the tool. Subsequent use of the same tool will add additional geometric elements to this group. A tool may also define other groups, with names defined by the tool. Thus, the ***db_all_bonds*** tool creates a set of groups including ones named "main_chain" and "side_chain".

4.3 Units

The distance units used in the system are defined in the databases that create the initial geometry used for drawing. Unless you write your own tools and create your own databases, you will be working with atom positions with Angstroms as units.

4.3 Drawing tools

Drawing tools are the heart of the VIEW system. They provide the capability of creating, modifying and deleting geometry from the display, or specifying ways to interact with the geometry. In addition to the library of drawing tools provided with the system, you may add new tools or modify existing ones at anytime by using the tool specification language (See *VIEW Drawing Tool Definition Language – Language Description* for a description of the language syntax and semantics, and *VIEW Drawing Tool Definition Language – Development Environment* for a description of how to build, modify, or test tools).

4.4 Interactive Events

Drawing tools may define what are known as *events*. These are actions to be performed when a keyboard key is depressed or a dial is turned. A tool's events become *active* when the tool is executed. The events active at any time are shown in the *event panel* (opened by clicking on **Show Events** in the **Drawing Tools** subpanel of the main panel). Each event will replace any event previously defined on that same key or dial, and will remain active until it is replaced by another event on the same key or dial, or explicitly removed. Events are not affected by exiting a tool, or by the implicit exit that goes with executing a new tool.

The tool language provides a statement that removes event definitions. Using this statement, drawing tools may remove any or all previously defined events. The tool library contains a tool named "remove_events" that removes all currently defined events.

Events operate immediately; they are not queued. Only one event may be active at a time. Events may be initiated even when another event is executing. This is shown diagrammatically below. Time is displayed horizontally with a different event on each line. A dotted line means that the event is active.

```
a ------------                              --------------
               b -----              -----
                        c ----------
```

The event on the "a" key is started and runs until the "b" key is pressed. The "b" event executes until the "c" key is pressed. When the "c" event is completed, the "b" event resumes. Likewise, when the "b" event terminates, the "a" event executes to completion.

An exception to the execution model presented above is that when an event is executing, it may not be interrupted by the same event. Thus if an event on the "a" key is executing, a second depression of the "a" key (before the first event has completed) will have no effect. The new "a" event is not permitted to interrupt the first event, nor is it stored for later processing. The same holds true for dials; movement of a dial will be ignored while an event on that dial is being processed.

Events may operate sequentially. That is, the "a" key may be pressed repeatedly and as long as the previous event on the key has completed when the key is pressed, the event will execute over and over. The same effect may be obtained by holding a key down. After the key has been depressed for about a second, it will act as if it were being pressed repeatedly. Likewise, turning a dial continually will result in the event for that dial being executed each time the previous event has completed.

Converting Brookhaven Protein Databank (PDB) Files to VIEW Database Format

Larry Bergman
1/22/93

The *pdbtoview* command is used to convert Brookhave Protein databank (PDB) files to VIEW database format. *pdbtoview* will be defined when you have completed the system installation described in the document *Installing the VIEW System*.

The syntax of this command is:

pdbtoview -directory *directory_name* *molecule_name*

-directory *directory_name* – defines the directory under which the subdirectories that contain the PDB files (in a subdirectory called Pdb) and the VIEW database files (in a subdirectory called Database) are located.

molecule_name – The name of the PDB file (in the directory *directory_name*/Pdb) that is to be converted. The VIEW database file produced (in the directory *directory_name*/Database) will have the same name.

EXAMPLE:

```
pdbtoview -directory /usr/people/view/my_data 1crn
```

will convert the molecule 1crn (crambin) from PDB to VIEW database format.

NOTE: This command will process all ATOM and HETATM records within the PDB file up to the first TER record. Any records after the first TER record will not be processed. If you wish to process multiple segments (between TER records), extract each into a separate file in the *Pdb* subdirectory, and process each separately using the *pdbtoview* command to create separate VIEW database files.

Running the VIEW System

Larry Bergman

1/24/93

VIEW is run with the *runview* command or alternately with the *iview* command. These commands will be defined when you have completed the system installation described in the document *Installing the VIEW System.*

Prior to executing *runview* or *iview,* you should complete converting all data from PDB format that you wish to use using the pdbtoview program (See the document *Converting Brookhaven Protein Databank (PDB) Files to VIEW Database Format).*

The syntax of the *runview* command is:

runview -directory *directory_name* [-antialias] [-background *red green blue*]
[-dials] [-snapshot *snapshot_name*] [*geometry_file 1 geometry_file 2 . . . geometry_file n*]

-directory *directory_name* – defines the directory under which the subdirectories that contain geometry files, tools, database files, and snapshot files are located.

optional parameters

-antialias – specifies that antialiasing for lines is to be turned on. The default is no antialiasing.

-background red green blue – defines the background color for the display window. red, green, and blue are integers between 0 and 255. If not supplied, the default background is (40,40,40).

-dials – if supplied, this flag specifies that the dial box is to be used in place of the virtual trackball. The default is that dials are not to be used, and the mouse will control movement of the image (using the virtual trackball).

-snapshot *snapshot_name* – specifies a pre-existing snapshot to use when initializing the system. A snapshot is a record of the display state that captures which panels are open, the position of all panels, and the contents of the user tools panel. Snapshots are stored in *directory_name*/Snapshot.

geometry_file 1 ... geometry_file n – geometry files that are to be loaded. Geometry files are stored in *directory_name*/Geometry. Geometry files may be listed after all other command-line options.

EXAMPLE:

```
runview  -directory /usr/people/view/my_data
    -background 100 100 100  single_helix
```

will start the VIEW system with a gray display panel and the geometry file "single_helix" loaded.

The *iview* command is a simpler form of *runview* that requires no -directory option. The directory that you specified for the *install_view_data* command when you installed the VIEW system will be used to locate all data files. All optional arguments for *iview* are identical to*runview*.

EXAMPLE:

```
iview  -background 100 100 100  single_helix
```

VIEW

User Interface Description

Table of Contents

VIEW User Interface Description

Larry Bergman
1/24/93

1. INTRODUCTION

The VIEW system interface consists of a display window which may contain 3-D geometry, mouse and/or dial manipulation of geometry in the display window, and a set of on-screen interface objects including button panels, editors, dialogue boxes, and file lists.

This document starts with a description of the types of interface objects that are used in controlling the VIEW system. The screen configuration and attributes of the graphics display are described. A description of various panels follows, including panels that control groups and panels that control drawing tools. Screen snapshotting is defined and described. A few additional miscellaneous features are discussed.

2. USING THE INTERFACE

2.1 Physical devices

The physical devices are the keyboard, the mouse, and (optionally) the dial box.

2.1.1 Keyboard

The keyboard is used for typing text in editors and text areas, entering/changing text in queries, and individual keys may also be used to trigger interactive events.

Some of the keys have special functions.

"backspace" key	— deletes the character that precedes the text cursor.
"delete" key	— deletes the character that follows the text cursor.

The "return" and "tab" keys also have special functions that depend on the type of object and will be discussed with each.

2.1.2 Mouse

The right mouse button is used for picking or *selecting*. Most of the drawing tools require selecting one or more geometric objects.

To select:

1) Position the cursor arrow tip over the object you wish to select
2) Click the right mouse button.

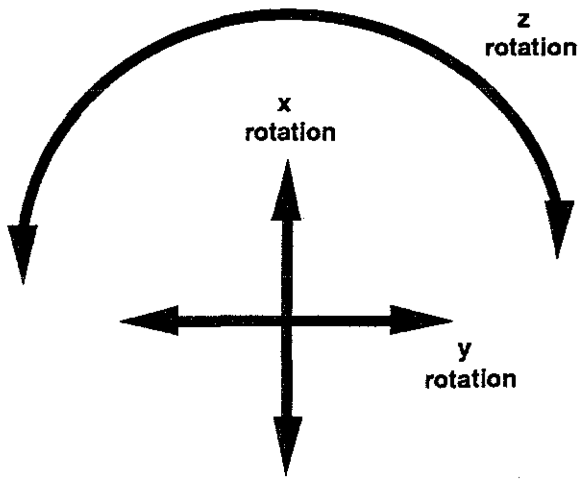


Figure 1

Rotation of 3-D geometry using the virtual trackball (mouse motions shown using arrows)

3-D geometry displayed in the display window may be manipulated using the mouse. The screen cursor simulates a trackball (unless "-dials" is specified when VIEW is invoked; see section 2.1.3 below and figure 1). The following table shows the results of mouse cursor movement while different mouse buttons are depressed.

Left mouse button	— The geometry will follow the mouse's motion (translate)
Middle mouse button	— The geometry will rotate about its center

Up and down motions near screen center	— Rotates geometry about the X (horizontal) axis
Side-to-side motions near screen center	— Rotates geometry about the Y (vertical) axis
Motions near edge of the screen	— Rotates geometry about the Z (out-of-screen) axis

Both buttons depressed — The geometry will be scaled

Motions up or right — The geometry will shrink

Motions down or left — The geometry will expand

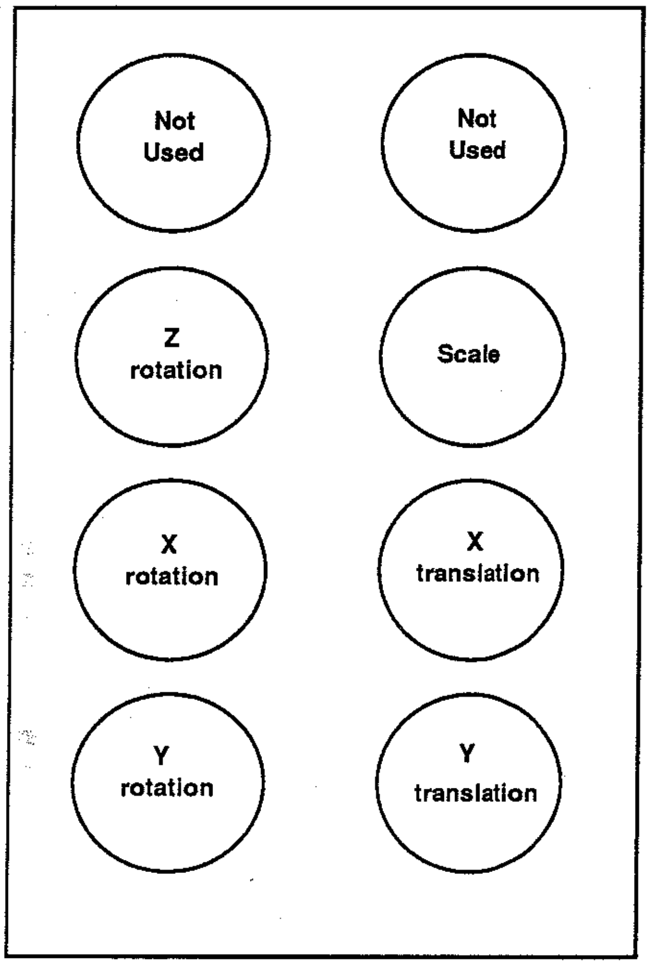


Figure 2

Dial control of display geometry

2.1.3 Dials

The SGI-supplied dial box has two functions in the VIEW system. It may be used to manipulate the 3-D geometry in the display window (if the "-dials" option is specified; see *Running the VIEW System*). Additionally, interactive events may be assigned to dials by specific drawing tools.

The assignment of dials to 3-D manipulations is presented in figure 2. The unconventional ordering of the dials (z,x,y) makes it simplest to do rotations about y, the most common viewing operation.

If you develop a new tool that assigns an event to one of the system-defined dials (0-5), your event will override the system definition.

2.2 On-screen devices and objects

There are several types of interface objects that are displayed on-screen:

- buttons (figure 3)
- button panels (figure 3)
- file lists (figure 4)
- queries (figure 5)
- confirmations (figure 6)
- information messages (figure 7).

A final type of interface object, an editor, is used only for development of new drawing tools, and is described in the document, *VIEW Interactive Tool Definition Language - Development Environment.*

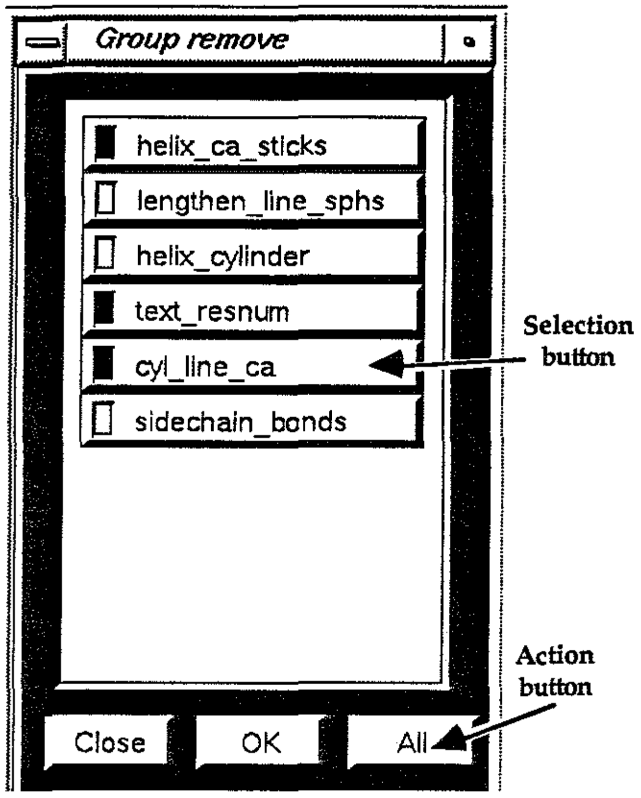


Figure 3

Button panel showing both selection and action buttons

2.2.1 Buttons

Buttons (figure 3) are actuated by positioning the mouse cursor over the button and then clicking with any of the mouse buttons. Buttons come in two styles: action buttons and selection buttons.

A *selection button* records a change in system state. In some cases, the selection serves as information for some action to be specified later with an action button (e.g., selecting a group in the **Group write** button panel). In other cases, a selection button causes an immediate change (e.g. executing a drawing tool in the **User tools** panel, or toggling a group using the **Group display** panel). In either case, the selection button is either highlighted or dehighlighted, visually indicating the state. When *selected*, a rectangle to the left of the button's text is yellow; when not selected, the rectangle is gray. Action buttons lack this highlight area.

Action buttons cause something to happen immediately (such as opening or closing a panel, executing a drawing tool, or writing a file). Their effect may be considered to end as soon as the button is pressed.

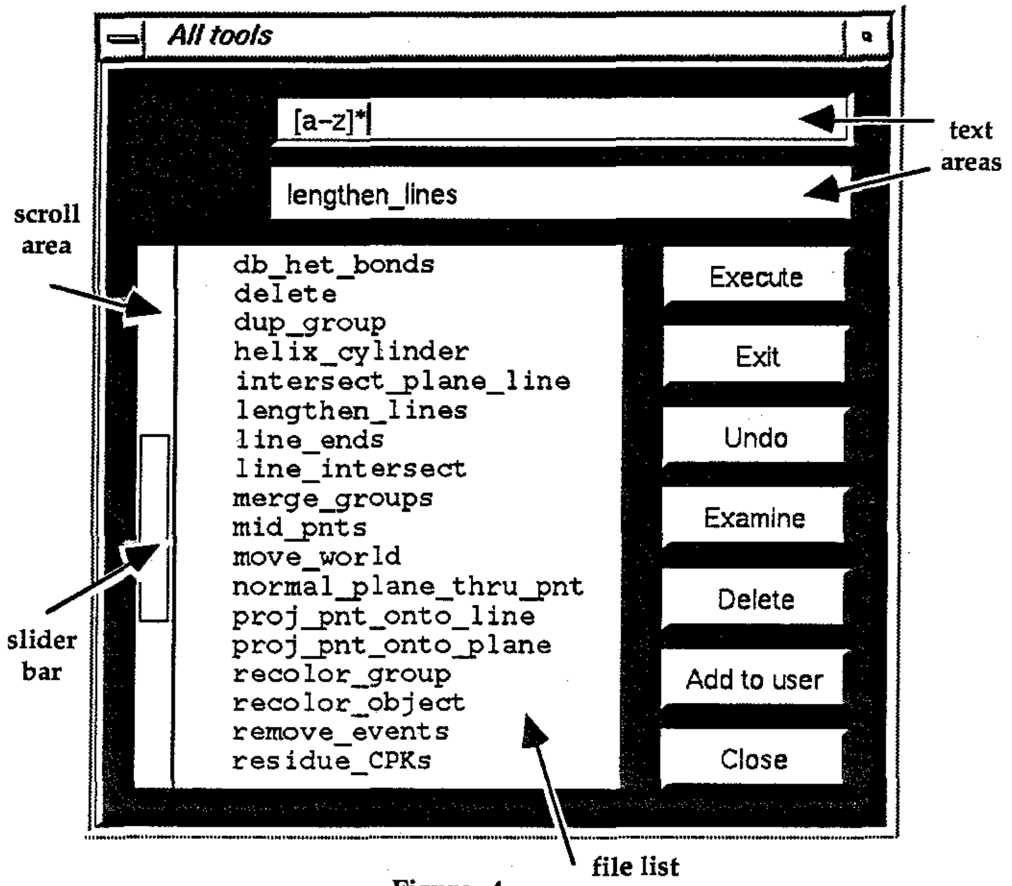
2.2.2 Button Panels

Button panels (figure 3) are organized groups of buttons. A button panel provides a single function or a group of closely related functions. For example, there is a button panel that is used to remove groups from the display window. This panel has one button for each group, used to specify which groups to remove, a button that allows you to select all groups for removal, and a button for actually performing the removal.

All button panels (except the **Main** panel which is always open) contain a button labeled "Close" that is used to delete the panel from the screen; the **Close** button will not be described for each individual panel. Panels may always be reopened by clicking on the appropriate button in the **Main** panel.

2.2.3 Text areas

File lists and queries contain boxes, known as *text areas*, in which you may type or modify text (figure 4).



```
db_het_bonds
delete
dup_group
helix_cylinder
intersect_plane_line
lengthen_lines
line_ends
line_intersect
merge_groups
mid_pnts
move_world
normal_plane_thru_pnt
proj_pnt_onto_line
proj_pnt_onto_plane
recolor_group
recolor_object
remove_events
residue_CPKs
```

Figure 4

All tools file list showing text areas

Objects that contain text areas will pop-up with a text cursor (vertical blue bar) in the topmost area. The text cursor may be repositioned in any of the text areas by positioning the mouse cursor, then clicking any mouse button. The text cursor will appear at the mouse cursor position. If the mouse cursor is to the right of the text, the text cursor will appear at the end of the text. With the text cursor positioned, new text can be inserted by typing.

Several editing operations may be performed by using a combination of mouse and keyboard as indicated in the following table:

Desired Result	User Action	System Response
Selecting text	Position the mouse cursor at the place in the text that the selection is to start or end. Hold down the left mouse button.	Blue vertical text cursor will appear at mouse cursor position
	While continuing to depress the left mouse button, drag the mouse over the text.	Text will be highlighted in black.
Deleting selected text	Press the backspace or delete key.	Highlighted text will disappear.
Replacing selected text	Type in replacement text.	New text will replace highlighted text.

2.2.4 File lists

File lists are panels used for selecting drawing tools, geometry files, or screen snapshot files (figure 4). The file list contains a box that lists all files that are available. If the list is longer than the box, a scroll area will be displayed to the left of the file list. You may use the slide bar within the scroll area to move up and down the file list by:

1) positioning the mouse cursor over the slide bar,
2) depressing and holding any of the mouse buttons and,
3) moving the cursor up or down within the scroll area.

Larger movements are possible by clicking any mouse button with the cursor positioned in the scroll area either above or below the slide bar.

Files are selected by positioning the mouse cursor over them and then clicking with any mouse button. The selected file will highlight brown, and its name will appear in a box labeled "name". Use the name in this area to determine which file will be chosen for operations in the file list (such as **Execute** in the **All tools** file list).

You may restrict the set of files listed by specifying a *pattern*. A pattern is a string that is used to search all the available file names. Those names that match the pattern will be listed. In determining if a file name matches, the following rules apply:

- Any single character in the pattern other than * or ? will match an identical character in the file name.

- An asterisk, *, in the pattern will match any sequence of characters (zero or more) in the file_name. This is commonly referred to as a *wild-card character*.

- A question mark, ?, in the pattern will match any single character in the file name.

- A set of characters contained in square brackets, [], will match any one of the characters at the specified position in the file name. Specifying a starting number or letter, followed by a dash, followed by an ending number or letter, is a short-hand notation for the set of all numbers or letters between the start and end inclusive. In figure 4, for example, the pattern [a-z]* will match anything that contains a starting alphabetic character.

Let us look at a few examples of pattern matching. Suppose we have the set of files:

```
1crn_all_sticks
1crn_ca_cyls
1crn_ca_sticks
1crn_cb_sticks
1crn_cg_sticks
1crn_call_sticks
2mhr_all_sticks
2mhr_ca_sticks
```

1) Use the pattern, " *" to see all available file names.

2) The pattern, " 1crn_c*sticks" , would produce a smaller file list containing:

```
1crn_ca_sticks
1crn_call_sticks
1crn_cb_sticks
1crn_cg_sticks
```

For the first file, the asterisk matched the string, " a" . For the second, the asterisk matched the string, " all " .

3) The pattern, " *_c?_*" , would produce a file list containing:

```
1crn_ca_cyls
1crn_ca_sticks
1crn_cb_sticks
1crn_cg_sticks
2mhr_ca_sticks
```

For the first two files, the question mark matched the string, " a" . For the third, the question mark matched the string, " b" .

4) The pattern, *_c[ag]_* would produce a file list containing:

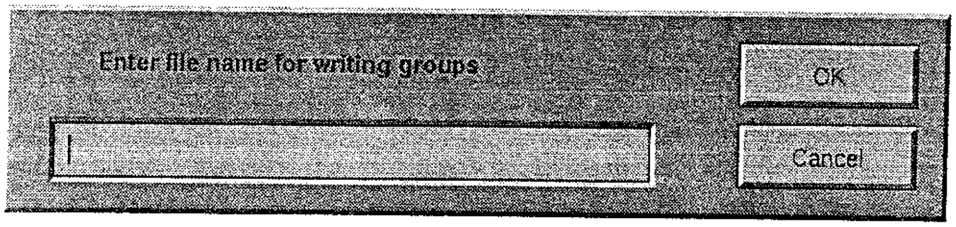
1crn_ca_cyls
1crn_ca_sticks
1crn_cg_sticks
2mhr_ca_sticks

The expression in square brackets, [ag], will match either an "a" or a "g". Thus, we get all the files from example three except for the single file that had a "b" in this position ("1crn_cb_sticks"). The pattern "*_c[a-g]_*" would produce the same file listing as example three, by specifying any character between "a" and "g" (inclusive) in place of example three's question mark.

Patterns are specified by editing the text in the text area labeled " pattern" . Text is edited as described in section 2.2.3 above. If the " return" or "tab" key is pressed after the pattern has been changed, the list of files that matches the new pattern will be retrieved. The pattern box will turn white while the system is processing the match.

2.2.5 Queries

Queries are used for entering or modifying strings. Each query contains one or more text entry areas, each of which can be edited as described in section 2.2.3 above (figure 5).



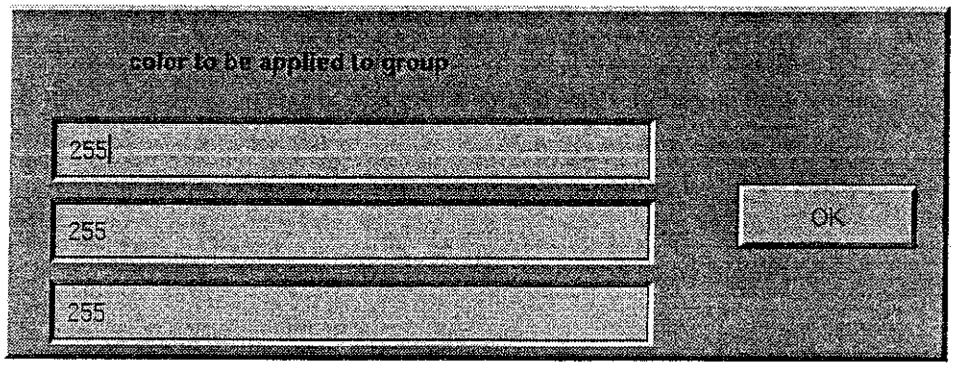


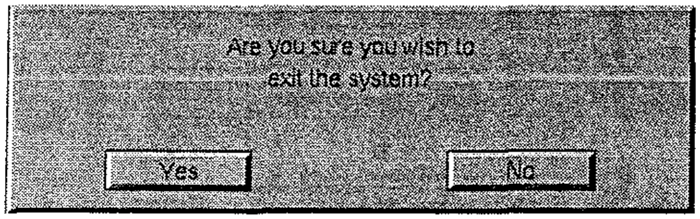
Figure 5

Single text area and multiple text area queries

The "return" and "tab" keys have identical functions. Their operation depends on whether the query has one text area or more than one, and also on whether or not the text has been edited (text added, deleted, or replaced) as described in the following table. These seemly arbitrary rules are imposed by the FORMS widget library used in building the VIEW system.

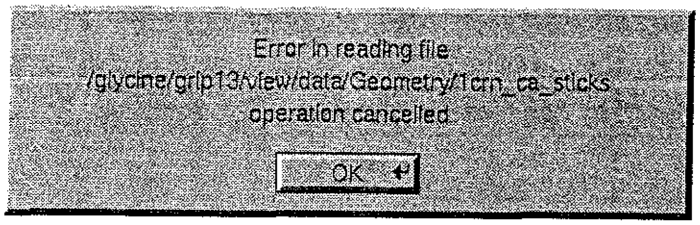
	Text unedited	**Text edited**
Single text area query	no effect	query will disappear and new text will be accepted
Multiple text area query	cursor will move to next text area. If in the last area, cursor will move to first.	cursor will move to next text area. If in the last area, cursor will move to first.

When all the strings in the query are as you want them, click on the **OK** button (or, if the query has a single text area and the text has been edited, you may press "return" or "tab"). Some queries will have a **Cancel** button. Clicking on this will cancel the operation that generated the query.



Are you sure you wish to
exit the system?

Figure 6a



Error in reading file
/glycine/grip13/view/data/Geometry/1crn_ca_sticks
operation cancelled

Figure 6b

Yes/No and OK confirmations

2.2.6 Confirmations

Confirmations are user interface objects that require you to reply to a simple question. There are two types. Yes/No confirmations (figure 6a) require that you respond with Yes or No to a question by clicking on one of two buttons.

Yes Button	Confirms the action in progress and proceeds with it.
No Button	Cancels the current operation.

OK confirmations (figure 6b) report an error and require that you click on **OK** in order to proceed. Both types of confirmations disappear when you click on one of the confirmation buttons.

Select object with initial color

Figure 7

Information message

2.2.7 Information messages

Certain operations will cause an information message to be displayed at the top of the screen (figure 7). There are two types of these messages. A *selection message* has a blue background and indicates that a drawing tool is waiting for you to select an element of geometry in the display window. This message will disappear automatically when you perform the selection (and often will be replaced by another selection message, or the same message will be repeated).

The second type is a *wait message*. This type has a red background and indicates that the system is busy with some operation. An example is a message saying "Loading geometry file: 1crn_all_sticks" . The interface will not respond to the mouse or dials as long as a wait message is displayed. These messages disappear automatically when the operation is completed.

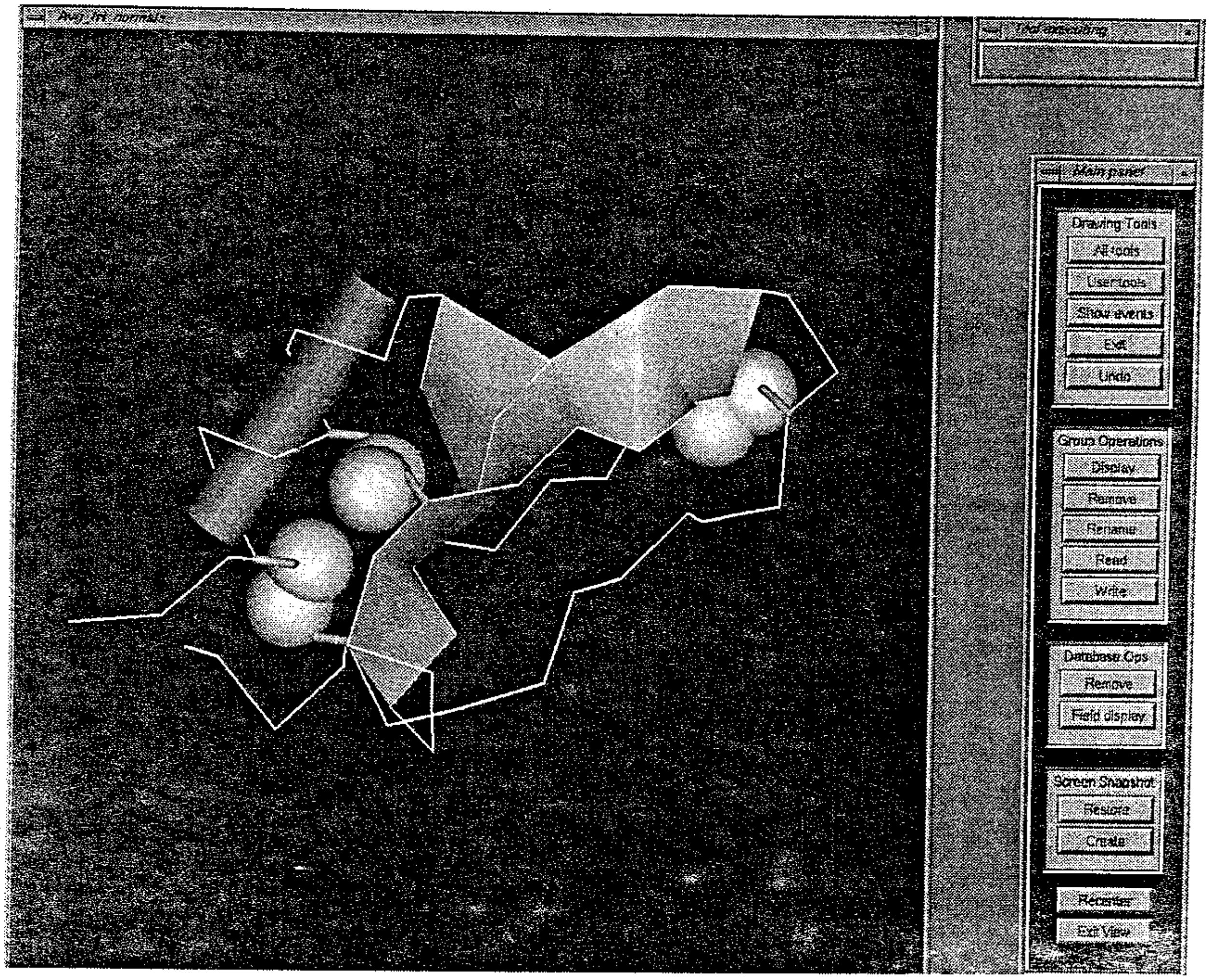


Figure 8

The VIEW system

3. THE INITIAL CONFIGURATION

When the system is initialized, a display panel (in which all graphics will be displayed) and a **Main** panel (used for invoking all system functions) will appear at predefined locations on the screen (figure 8). The **Main panel** (figure 9) consists of four subpanels each of which contains several buttons. Each subpanel contains a set of related functions; the order of the buttons in the subpanel is based on expected frequency of use – most used at the top. The subpanels, the buttons contained in each, and a brief description of the function of each button is shown in the following table:

Subpanel	Button	Function
Drawing Tools	All tools	Generates the **All tools** panel which allows you to execute, examine, delete any drawing tool from the library; add tools to User tools panel.
	User tools	Generates the **User tools** panel which allows you to execute any of a selected set of drawing tools from the library.
	Show Events	Generates the **Show events** panel, which displays all currently active events.
	Exit	Cancels the currently executing drawing tool.
	Undo	Undoes the effects of the last drawing operation (exact effect of **Undo** varies depending on which tool operated last)
Group Operations	Toggle	Generates the **Group display** panel, which allows you to turn the display of geometry groups on or off.
	Remove	Generates the **Group remove** panel, which allows you to remove any or all geometry groups.
	Rename	Generates the **Group rename** panel, which allows you to specify new names for geometry groups.
	Write	Generates the **Group write** panel, which allows you to write any or all geometry groups to a disk file

Subpanel	Button	Function
Database Ops	Remove	Generates the **Database remove** panel which allows you to remove any or all databases.
	Field display	Generates the **Field display** panel which displays a summary of the contents of all currently loaded databases.
Screen Snapshot	Restore	Generates the **Restore snapshot** panel which allows you to select a previously generated snapshot file to be restored
	Create	Creates a new snapshot file containing the current interface configuration

There are also two buttons at the bottom of the main panel with the following functions.

Button	Function
Recenter	Recenters all geometry in the display window and scales it to fit within the display.
Exit View	Exits the system

The layout of the main panel is shown in figure 9.

4. THE DISPLAY WINDOW

3-D molecular geometry is presented in a graphics window referred to as the *display panel*

4.1 Geometry display

The VIEW system displays geometry using an orthonormal projection. This projection, unlike a perspective projection, produces equal-sized screen images for equal-sized objects; there is no size change based on distance from the viewer. Translating objects along the z-axis (out-of-screen axis), will have no effect on their screen size. For this reason, the scaling operation does not operate by translating objects in z (as in many systems that employ perspective).

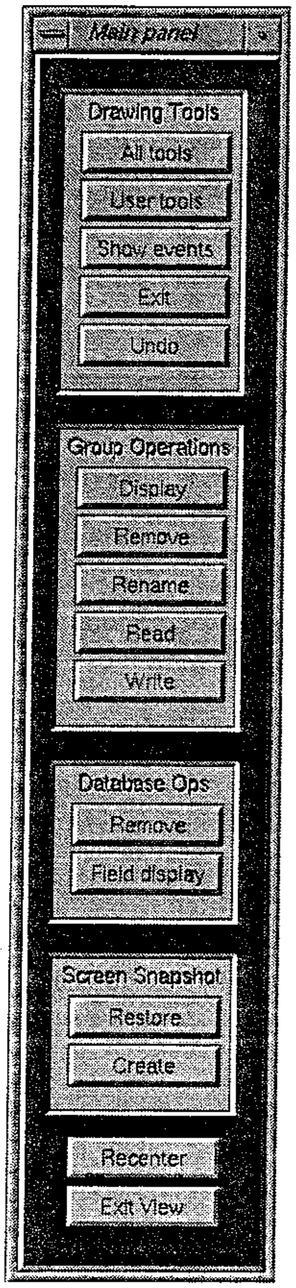


Figure 9
Main panel

Frequently polygons are used to tesselate a curved surface. When that is the case, ***flat shading*** , in which each pixel of a polygon is assigned the same shade, produces clunky, unattractive images. The VIEW renderer supports a more sophisticated technique known as *Gouroud shading*, which uses surface normals defined at each polygon vertex to produce smoothly shaded polygons. The normals are those of the curved surface that is being represented. The simpler drawing tools (such as ***triangle***), do not define per-vertex normals, resulting in flat-shaded polygons. Some of the more sophisticated tools such as ***ribbon*** do compute vertex normals, giving smooth surfaces. There is a tool named ***avg_triangle_normals*** that will compute average normals for all triangles in a geometry group, smoothing the shading for any polygons with coincident vertices. Section 4.3.6.5 in the document *VIEW Tool Definition Language - Language Description* describes how to define triangle normals when writing a new tool.

The 3-D geometry is lit using two white light sources. The light positions are fixed and may not be manipulated, nor may the lights be turned off.

The VIEW system supports anti-aliasing of lines. Anti-aliasing produces a higher quality image by minimizing the effect of jaggy lines at the expense of interaction speed. Starting the VIEW system with the command flag "-antialias" will turn on antialiasing (see *Running the VIEW System*). The default is no antialiasing.

4.2 Manipulating the Geometry

3-D geometry displayed in the display window may be manipulated with respect to the viewpoint using the mouse as described in section 2.1.2.

The right mouse button is used for picking as described in section 2.1.2.

If the "-dials" option is specified when the program is started, the mouse will not perform viewing manipulations of the image. Instead, only the dialbox will be used to rotate, translate, and scale the image. The assignment of dials to these functions is described is section 2.1.3.

5. GROUP OPERATIONS

The subpanel in the main panel labeled "Group Operations" contains buttons used to invoke panels that provide for toggling groups on and off, removing groups, reading groups from files or to writing them to files.

5.1 Displaying Groups

The **Group display** button panel (produced by clicking on **Display** in the **Group Operations** subpanel of the **Main** panel) is used for turning groups on or off in the display window (figure 10). The **Group display** panel shows all groups that are currently turned on (indicated by a yellow button), as well as those that are turned off (indicated by a gray button). Clicking on groups will toggle them from On to Off and vice versa.

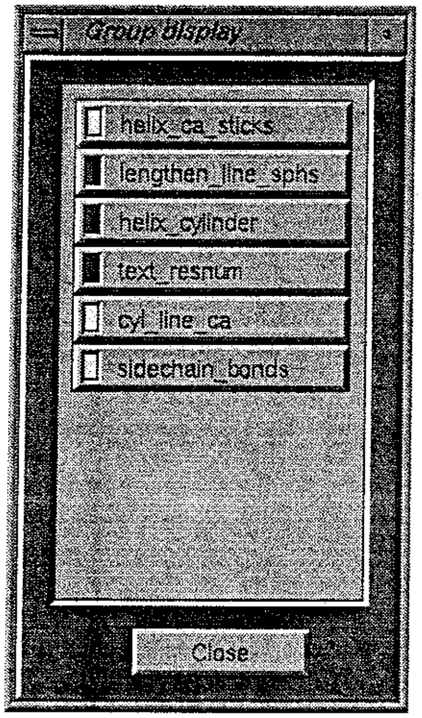


Figure 10

Group display panel

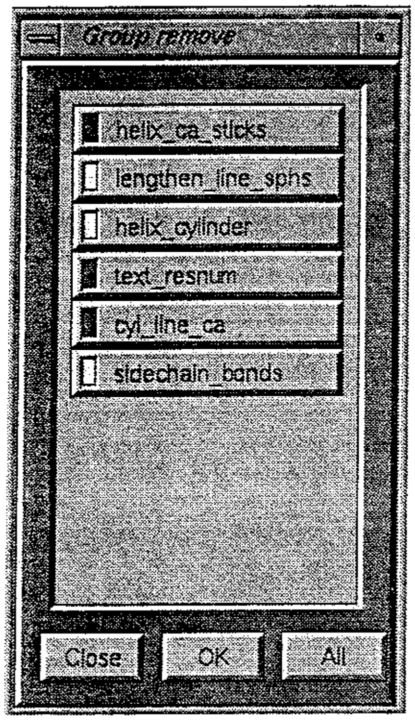


Figure 11

Group remove panel

5.2 Removing Groups

The **Group remove** button panel (produced by clicking on **Remove** in the **Group Operations** subpanel of the **Main** panel) is used for permanently removing groups of geometry (figure 11). Note that this operation is not undoable, and should be used with care. You may remove a subset of the currently defined groups, or all currently defined groups.

To remove a subset of the currently defined groups:

Desired Result	User Action	System Response
Selecting a group for removal or	Click on the name of an unhighlighted group.	Button turns from gray to yellow.
Select all groups for removal	Click on **All** button at the bottom of the panel.	All group name buttons turn from gray to yellow.
Deselecting a previously chosen group	Click on a the name of a highlighted group.	Button turns from yellow to gray.
Confirming the selection of groups for removal	Click on **OK** button, at the bottom of the panel, when all selections are completed.	Selected groups are removed.

5.3 Renaming Groups

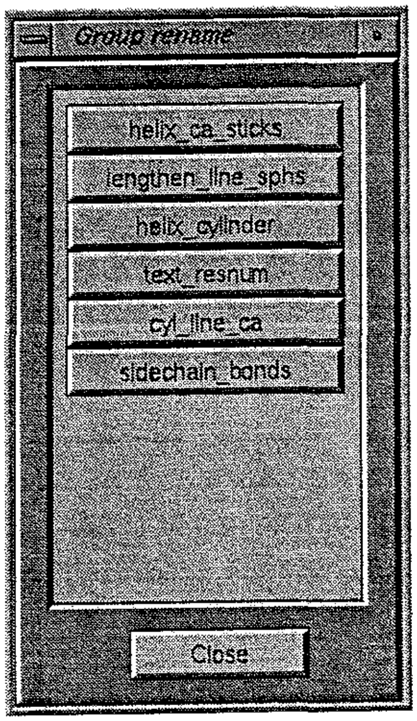


Figure 12

Group rename panel

The **Group rename** button panel (produced by clicking on **Rename** in the **Group Operations** subpanel of the **Main** panel) is used for renaming geometry groups (figure 12). Renaming is particularly useful for creating multiple groups using a single tool. For example, suppose that you wish to create two separate groups of triangles using the ***triangle*** drawing tool. To do this you would use the tool to create the triangles in the first group. You would then rename the "triangle" group (to, say, "triangle2") and then continue to draw triangles; the new triangles would go into a group with the original name (in this case "triangle").

Note that the renamed group cannot have additional geometry added to it (at least not by the tool used to create it, for that group knows only tool-specific names). In order to add to the renamed group, its name would have to be changed back to its original name (in the example, the "triangle2" group would have to be renamed "triangle").

Groups are named one-at-a-time using the **Rename groups** panel.

Desired Result	User Action	Result
Renaming a group	Click on the name of the group you wish to change.	Prompts you for the new name of the group.
Confirming the new name for a group	Specify a new name and hit "return" or "tab" or click on the **OK** button.	Renames the group.
Canceling the new name for a group	Click on **cancel** button.	Leaves the name unchanged.

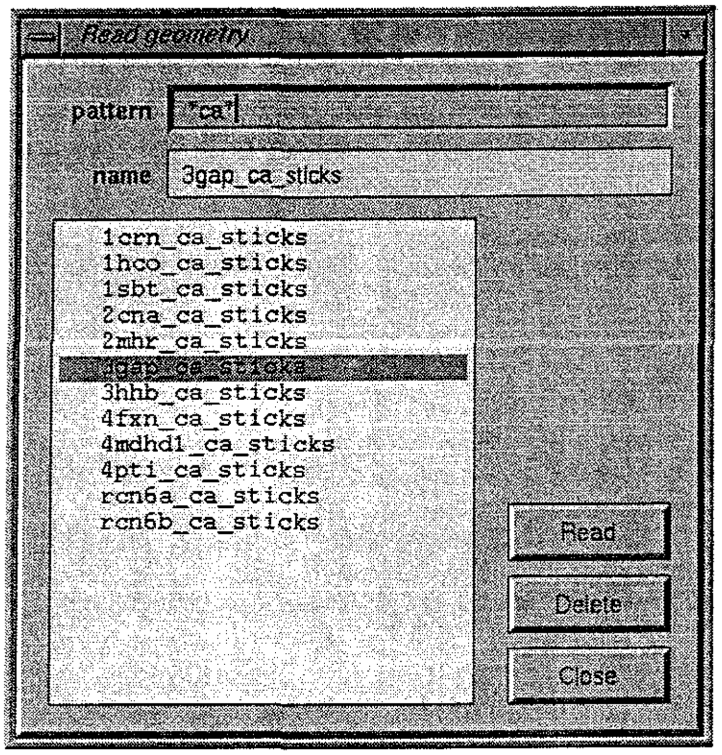
Note: The system will not allow you to create multiple groups with the same name. If you try to rename a group to an already existing name, the system will cancel the operation and issue a warning message.

5.4 Writing Groups

The **Group write** button panel (produced by clicking on **Write** in the **Group Operations** subpanel of the **Main** panel) is used for generating a disk file containing one or more groups of geometry (layout is identical to the **Group remove** panel, figure 11). These are referred to as *geometry files.*

To write to a geometry file:

Desired Result	User Action	System Response
Selecting a group for writing or	Click on the name of an unhighlighted group.	Button turns from gray to yellow.
Selecting all groups for writing	Click on All button at the bottom of the panel.	All group name buttons turn from gray to yellow
Deselecting a previously chosen group	Click on a the name of a highlighted group.	Button turns from yellow to gray.
Confirming the selection of groups for writing	Click on OK button, when all selections are completed	Prompts you for the name of the file to be written.
Naming the geometry file	Specify a name and hit "return" or "tab" or click on the OK button in the name query.	Writes the geometry file.
Canceling the write operation	Click on the Cancel button in the name query	No action will be taken.



1crn_ca_sticks
1hco_ca_sticks
1sbt_ca_sticks
2cna_ca_sticks
2mhr_ca_sticks
3gap_ca_sticks
3hhb_ca_sticks
4fxn_ca_sticks
4mdhd1_ca_sticks
4pti_ca_sticks
rcn6a_ca_sticks
rcn6b_ca_sticks

Figure 13

Read geometry panel

5.5 Reading Groups

The **Read** button in the **Group Operations** subpanel of the **Main** panel produces a file list containing previously generated geometry files that may be read from disk (figure 13).

Desired Result	User Action	System Response
Selecting a file to read	Click on the file name with the mouse.	File name will highlight in brown and will be displayed in the **name** area.
Confirming the selection of file to be read	Click on **Read** button.	File will be read from disk.

5.7 Deleting geometry files

Geometry files may deleted from disk file using the **delete** button in the **Group read** panel. Click on the **Read** button in the **Group Operations** subpanel of the **Main** panel to produce this panel.

Desired Result	User Action	System Response
Selecting a file to delete	Click on the file name with the mouse.	File name will highlight in brown and will be displayed in the **name** area.
Confirming the selection of file to be deleted	Click on **Delete** button.	File will be deleted from disk and the file list will be updated.

6. TOOL OPERATIONS

Tools may be accessed from two different panels. The **All tools** panel will allow you to access any drawing tool. You may execute, examine, or delete a tool from this panel. The **User tools** panel contains a subset of the available tools. From this panel, tools may only be executed. You may select those tools that are to be listed in the **User Tools** panel as described in section 6.4.1 below.

6.1 Executing a tool

Executing a tool starts the operation of that tool. The **Tool executing** panel will display the name of the currently executing tool. Most tools (but not all), once started, require you to pick one or more objects in the display window. Executing a new tool will cause the tool that was previously executing to cease.

Note that deactivation of events does not operate in the same fashion. When a tool is executed, all events that are defined by that tool become active. When another tool is executed, previously defined events are NOT deactivated unless the new tool explicitly deactivates them, or replaces them by defining new events on the same keys or dials.

6.2 Exiting a tool

Exit will terminate the currently executing drawing tool. It does not, however, deactivate events defined by that drawing tool.

6.3 Undo

Undo will undo the last drawing operation. It is for undoing the effects of drawing tools; although it may remove the effects of operations performed from the interface (e.g. renaming or reading groups), these interface operations are not individually undoable. For example, suppose that two triangles are drawn using the ***triangles*** tool, the "triangles" group is renamed to "tris", and then a cylinder is drawn using the ***helix cylinder*** tool. Clicking on **Undo** (in the **Drawing Tools** subpanel of the

Main panel, in the **All tools** panel, or in the **User tools** panel), will cause the cylinder to be removed, leaving both triangles in the group called "tris". Clicking **Undo** again, will undo the rename, but it will also back up over the second triangle draw, that is, the second undo will leave only a single triangle in a group called "triangles".

The amount of drawing that is undone by each undo varies from one tool to the next. To find out how much is undone, try it.

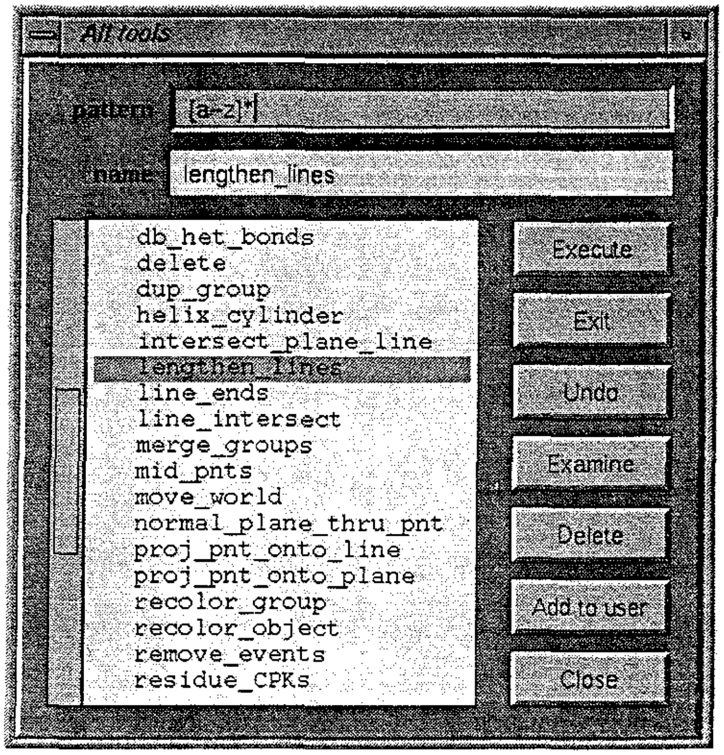


Figure 14

All tools panel

6.4 All tools panel

The **All tools** panel lists and lets you operate on any or all drawing tools in the library (figure 14). You may execute or examine the text of a drawing tool from this window as well as deleting tools from the library. You may also terminate the actions of the currently executing tool, or undo previous drawing operations.

The tools listed in the file list are determined by the pattern in the pattern box. Section 2.2.5 describes the use of patterns to choose particular sets of tools to be listed. As shown in figure 4, the **All tools** panel is initialized with the default pattern [a-z]* which lists all drawing tools in the system.

To operate on a tool using the **All tools** panel, you must perform two separate steps: select the tool, and then select the operation. Only one tool may be selected at a time; selecting a new tool will cause the previously selected tool to deselect (indicated by the brown highlight returning to white).

Desired Result	User Action	System Response
Selecting a tool	Click on the tool name in the file list	Tool will be highlighted in brown and its name displayed in the **name** area.

An operation is selected at the right side of the panel. The operations that apply to scripts are **Execute, Examine,** and **Delete.** The operation will be applied to the currently selected script. The operations **Exit** and **Undo** do not make use of the selected script.

Function	**System Response**
Execute	Starts executing the selected drawing tool. This will terminate the previously executing tool (except for any events defined, which may still be active).
Examine	Pops up an editor window containing the text of the selected tool. This text may be edited and/or executed from the editor window. See the document *VIEW Tool Definition Language - Development Environment* for details.
Delete	Removes the specified drawing tool from the library. The delete operation will request confirmation. Click on **Yes** to delete the tool, or on **No** to cancel the operation.

IMPORTANT NOTE: if a tool is added to the library using an editor outside the system (such as "vi"), the tool will not show up in the open **All tools** panel until the panel is updated. This occurs when the panel is next reopened or a new pattern is entered.]

6.5 User tools panel

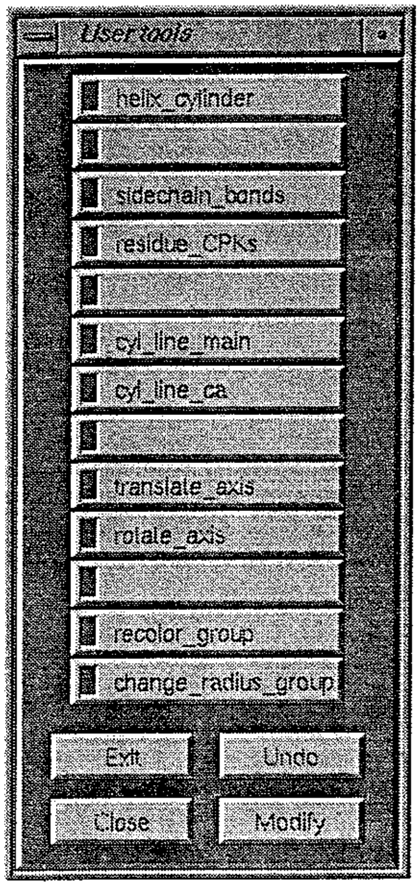


Figure 15

User tools panel.

The tools in the **User tools** panel are a subset of the tools available in the library (figure 15). You may choose the tools that are to go into this panel. Clicking on a tool button in the **User tools** panel will execute the tool; unlike the **All tools** panel, no operation selection is required. Clicking on a blank button (one containing no text) in the panel will cause the currently executing tool to exit.

6.5.1 Modifying the user tools panel

The simplest way to add a tool to the **User tools** panel is using the **Add to user** button in the **All tools** panel.

More extensive modifications to the **User tools** panel can be performed using the **Modify user tools** panel (figure 16). Clicking on **Modify** in the **User tools** panel will cause the **Modify user tools** button panel to appear.

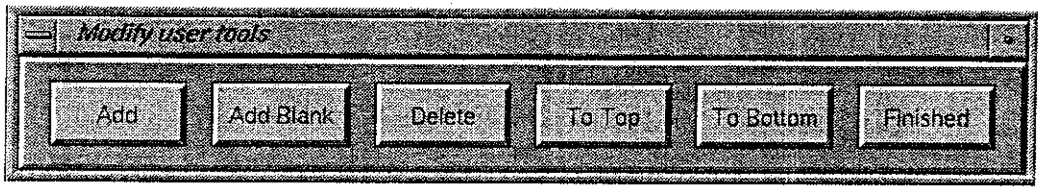


Figure 16

Modify user tools panel

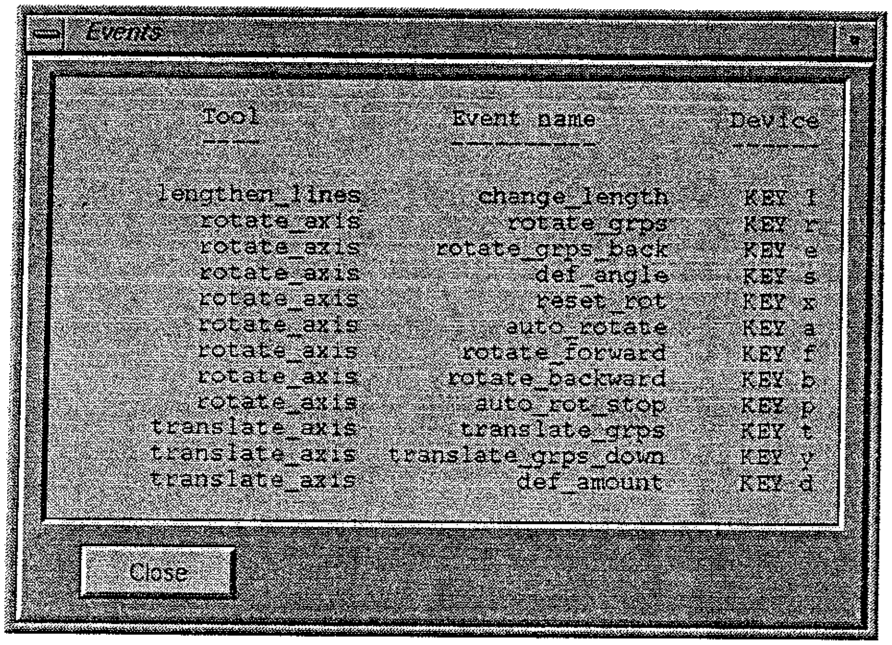
There are five operations that you can select in the **Modify user tools** panel:

- adding a tool to the **User tools** panel,
- adding a blank button to the **User tools** panel
- deleting a tool from the **User tools** panel,
- moving a tool to the top of the **User tools** panel,
- moving a tool to the bottom of the **User tools** panel,

These five operations are performed as described in the following table:

Desired Result	**User Action**	**System Response**
Adding a tool to the **Users tools** panel	Select the tool in the **All tools** panel. Click on **Add to user** in **All tools** panel. or Select the tool in the **All tools** panel. Click on **Add** in the **Modify user tools** panel.	A button containing the tool will appear at the bottom of the **User tools** panel
Adding a blank button to the **User tools** panel	Click on **Add blank** in the **Modify user tools** panel	A blank button will appear at the bottom of the **User tools** panel
Deleting a tool from the **Users tools** panel	Select the tool in the **User tools** panel. Click on **Delete** in the **Modify user tools** panel.	Tool will disappear from the **User tools** panel
Moving a tool to the top of the **User tools** panel	Select the tool in the **User tools** panel Click on **To Top** button in the **Modify user tools** panel.	Tool will move to the top of the **User tools** panel
Moving a tool to the bottom of the **User tools** panel	Select the tool in the **User tools** panel. Click on **To Bottom** button in the **Modify user tools** panel.	Tool will move to the bottom of the **User tools** panel.

The **User tools** panel is not able to execute tools so long as the **Modify** panel is displayed. When the **Modify** panel is closed by clicking on **Finished,** the **User tools** panel becomes active; clicking on a tool button will execute that tool. While the **Modify** panel is open, clicking on a tool in the **User tools** panel selects it for modification. Modify operations may be performed in any order.



```
            Tool               Event name            Device
            ----               ----------            ------

   lengthen_lines          change_length            KEY l
      rotate_axis            rotate_grps            KEY r
      rotate_axis       rotate_grps_back            KEY e
      rotate_axis              def_angle            KEY s
      rotate_axis              reset_rot            KEY x
      rotate_axis            auto_rotate            KEY a
      rotate_axis         rotate_forward            KEY f
      rotate_axis        rotate_backward            KEY b
      rotate_axis          auto_rot_stop            KEY p
   translate_axis         translate_grps            KEY t
   translate_axis    translate_grps_down            KEY y
   translate_axis             def_amount            KEY d
```

Figure 17

Show events panel

6.6 Show events panel

The **Show events** panel displays information about all currently active events (figure 17). For each event, the defining tool, the event name, and the triggering device is listed.

The **Show events** panel is not automatically updated when a tool defines new events or removes events. To update the panel, you must close and reopen it.

7. SAVING AND RESTORING THE SCREEN LAYOUT (SNAPSHOTTING)

At any time you may save a record of the VIEW interface by taking a *snapshot*. Positions of all panels, position of the display window, and, most important, the contents of the **User tools** panel will be recorded in the snapshot. Snapshots may be restored at any time, causing the screen to reconfigure to the layout in the snapshot. Note that snapshots do not capture the complete state of the system – they do not record the graphics in the display panel, nor the state of the currently executing tool and active events.

Desired Result	User Action	System Response
Recording a snapshot	Click on the **Create** button in the **Screen Snapshot** subpanel of the **Main** panel.	Requests the name to be assigned to the snapshot.
	Type in the name. Hit "return" or "tab" or click on the **OK** button.	Generates the snapshot.
Canceling a snapshot request	Click on the **Cancel** button.	Cancels snapshot request.
Restoring a snapshot	Click on the **Restore** button in the **Screen Snapshot** subpanel.	A file list displays all available snapshots.
	Click on a snapshot name.	Snapshot name will highlight in brown and will be displayed in the **name** area
	Click on the **Read** button	Restores the snapshot

Hint: a tree structure of snapshots may be generated by using qualified snapshot names. For example, if you're working on the molecule "2cna" and want to save a snapshot for that molecule as well as slightly varying snapshots for two visualization attempts, you might create snapshots with the names:

2cna
2cna_try1
2cna_try2

You may add as many qualifying fields as you wish in this manner. The names may become as long as you wish, although the **Snapshot restore** file list will only display the first twenty-two characters of the name.

8. DATABASE OPERATIONS

The subpanel in the main panel labeled **Database Ops** contains buttons used to invoke panels that provide for removing databases and displaying available database fields.

8.1 Database Remove

The **Database Remove** button panel (produced by clicking on **Remove** in the **Database Ops** subpanel of the **Main** panel) is used for permanently removing databases from the system (layout is identical to the **Group remove** panel, figure 11). Removing databases is useful when using tools that create new databases. Rerunning such a tool will cause the database that is previously created to be modified rather than regenerated. If you wish to start with a fresh database, delete the old one using this function. Once a database is removed, any display geometry that references that database will no longer have valid references. This may cause certain drawing tools that perform database accesses to fail (and popup a debugger) when operating on this geometry.

The remove operation is not undoable, and should be used with care. You may remove a subset of the currently defined databases, or all currently defined databases. Note that this function does not remove database files. This must be performed outside the system using UNIX commands.

Desired Result	User Action	System Response
Selecting a database for removal or	Click on the name of an unhighlighted database.	Button turns from gray to yellow.
Selecting all databases for removal	Click on **All** button at the bottom of the panel.	All database name buttons turn from gray to yellow.
Deselecting a previously chosen database	Click on a the name of a highlighted database.	Button turns from yellow to gray.
Confirming the selection of databases for removal	Click on **OK** button, at the bottom of the panel, when all selections are completed.	Selected databases are removed.

8.1 Field Display

This function produces a button panel that shows all the field names that are available for all currently loaded databases. It is strictly an information panel, useful for deciding which tools to use or for selecting field names in modifying/creating a tool.

The database fields that are automatically defined by VIEW molecular databases are described in section 4.3.10 of *VIEW Interactive Tool Definition Language – Language*

Description.

9. OTHER FUNCTIONS

Several other functions are available from the main panel.

9.1 Recenter function

This function centers the geometry in the display window by translating and scaling all on-screen geometry   Orientation of the geometry is not affected.

9.2 Exit VIEW function

This function permits you to terminate a VIEW session. The system will request confirmation when this button is pressed.

VIEW

Drawing Tool Library Description

Table of Contents

VIEW Drawing Tool Library Description

Larry Bergman
1/24/93

1. INTRODUCTION

This document describes the individual tools that are supplied with the VIEW system. Each tool is briefly described, and the sequence of actions you perform in using it are listed. Additionally, all events are described for each tool.

Prior to using this document, you should read *VIEW Exploratory Molecular Visualization System - Overview.* That document covers basic concepts of the VIEW system. This document will assume that you are familiar with these underlying ideas and definitions. I also suggest that you be familiar with the use of the VIEW system as described in the document, *VIEW User Interface Description.*

1.1 Geometry

Many of the tools create on-screen geometry. The geometry created consists of one or more *geometric primitives.* The primitives are: sphere, line, cylinder, triangle, and text. More complicated geometric forms are always drawn using these primitives. For example, the ***normal_plane_thru_pnt*** tool produces a quadrilateral. This quadrilateral is composed of two adjoining triangles.

Geometry is often created in pieces in this manner. The most common case is the creation of atomic bonds. Although a bond could be represented with a single line or cylinder, it is always created as two-half bonds (i.e. two adjoining line segments or cylinders). This allows a pointer to an atom record in the database to be associated with each half-bond. With geometry so constructed, we can select an atom by clicking on the half-bond (line or cylinder) near the desired atom position.

Operations on geometry are always at the level of a group or a primitive; it is not possible to operate on levels in between. Consider the ***delete*** tool which removes individual geometric primitives. If a quadrilateral produced by ***normal_plane_thru_pnt*** is to be removed using ***delete***, the two triangles must be removed separately; there is no way to remove the quadrilateral in a single step. Similarly, many tools will highlight selected primitives. Selecting a bond will cause one of the half-bonds that comprise it to highlight; you will not see the entire bond change color.

1.2 Relationships between geometry and databases

Many of the tools create geometry using information from a molecular database, often atom positions. Most of these tools associate pointers to the database with each geometric primitive created. Some do not, however (this will be indicated in the tool descriptions), usually when the created geometry is not unambiguously associated with an individual atom or bond. It is also possible to remove database pointers from the geometry by removing the database (using the **Database remove** operation described in Section 8.1 of *VIEW User Interface Description*).

Clicking on a geometric primitive that has no database pointers will produce no result when the system requires an atom selection.

When geometry which contains database pointers has been moved relative to other geometry (using ***rotate_axis***, for example), the location of the geometry and the atom positions in the database will no longer correspond. If the moved geometry is used to specify atoms for other drawing tools, the new geometry will be drawn at the original atom locations, not at the positions of the moved geometry. You may update the database to reflect the position of the on-screen geometry using the tool ***update_db***, thereby circumventing this problem.

1.3 Selections

Almost all of the drawing tools involve selections of one or more geometric primitives in the display panel. Selections are made by positioning the mouse cursor near the geometric object to be selected and then pressing the right mouse button.

Selections are either of atoms or geometric objects (such as spheres or lines). When atoms are to be selected, perform the selection on some element of geometry near the atom position. The system indicates a successful selection by displaying a red sphere at the atom position.

If an object is to be selected, the system either changes the color of the selected object, or performs some action that results from that choice. If a color change results, the new color displays until the screen is redrawn (the color then reverts to the original color of the object), either when you manipulate the geometry with the mouse, or when you perform the next selection.

Many of the object selections are used to define a location. In this case, the center of the object you select is used; usually you will wish to select a sphere. Watch the color change to make sure that you selected the object you wish – if not, use **undo** and then try again. Other selections require that you choose a line. In these cases, any linear object, either a line or a cylinder, may be selected; a cylinder may be thought of as a fat line.

Some of the object selections are used to specify a geometry group or a molecule to be operated on. In this case, the selection chooses the group or molecule by specifying one geometric element from it.

1.4 System prompts

There are two forms of system prompts – selection messages and queries. A *selection message* is a blue pop-up panel at the top of the screen containing text prompting you to make a selection in the display panel.

Queries are pop-up panels containing text areas in which you are to specify a number or string. Default values are provided in each query area. Queries are completed by pressing "return", "tab", or by clicking on **OK**. A more detailed description of the use of queries is contained in section 2.2.5 of the document *VIEW User Interface Description.* Generally, completion of a query does not result in a change in the display panel; queries usually change some tool parameter to be applied to future drawing operations.

1.5 Distance units

Some of the tools require that you define a distance (for example, the ***select_radius*** event in the ***arrow*** tool requires that you define a radius). The distance units are the units defined by the data that you are visualizing. If you are viewing a molecule, the units are Angstroms.

1.6 Events

A number of tools define events. *Events* are sequences of actions that are triggered by depressing a keyboard key or turning a dial. Some events are used to change parameters, others produce actions such as rotating or translating geometry. More information on events will be found in section of 4.4 of the document *VIEW Exploratory Molecular Visualization System, —Overview*

1.7 Tool descriptions in this document

The format of each tool description is as follows:

tool name

Brief tool description

Table detailing the actions you perform and the system prompts and responses that accompany them.

Optional event descriptions. For each event:

Event name followed by a brief event description

Table detailing the actions you perform and the system prompts and responses that accompany these actions.

The optional event descriptions contain an information block for each event. The block lists the name of the event (which may be viewed in the **Show events** panel of the interface), a brief description of what the event does, and then the sequence of user actions that are used to trigger and complete the event. Events are started when the tool that defines them is executed. Events may be stopped by executing the ***remove_events*** tool, described below.

Let us look at a sample, and describe in detail how the tool is used. The sample is the first tool in the library, ***arrow.***

arrow _atoms

Draws a series of arrows between pairs of atoms

	System Prompt	**User Action**	**System Response**
repeat as often as desired	Select atom for arrow tail	Click on an atom	Red ball appears at the atom position
	Select atom for arrow head	Click on a second atom	A red ball appears at the second atom position. Arrow is drawn connecting the atoms, then the red balls disappear.

The description tells us that the tool is used for drawing arrows and allows us to produce a series of them. The sequence of actions tells us that there are two selections to be performed and that the pair of selections may be performed repeatedly. The first selection, the arrow tail, is of an atom. Position the mouse cursor near an atom position (for example, near the place where two vector bonds meet), and click with the right mouse button. The system draws a red sphere. The second selection, the arrow head, is performed in the same way. After the second selection, the arrow is drawn, the spheres disappear, and then you may continue with a new atom selection.

Optional Events

select_length	Specifies whether the arrows are to connect the atoms, go beyond each atom, or fall short of each atom.

User Action	System Response
Press letter key "l"	Query panel appears with prompt: *Length scaling factor:* Displayed number is the previously defined scaling factor.
Choose a scaling factor of one (this is the default value when starting this tool) or	Future arrows will connect the two selected atoms together.
Choose a length scaling factor of less than one or	Future arrows will fall short of each of the two selected atoms.
Choose a length scaling factor of greater than one.	Future arrows will extend beyond each of the two selected atoms.

select_radius	Specifies the radius of the shaft for future arrows.

User Action	System Response
Press letter key "r"	Query panel appears with prompt: *Specify radius for shaft.* Displayed number is the previously defined radius.
Choose a number for the radius of the arrow shaft.	Radius of arrow shaft will change to the new value for future arrows.

The event section for this tool tells us that there are two events defined. The first, on the letter "l" key, is called *select_length*. This tool allows us to specify a scaling factor that lengthens or shortens the arrow. If we press the letter "l" key, a query appears requesting a value for the scaling factor. The initial value in the query is 1. We can change this value (or leave it unchanged) and then press return or click on **OK**; the query will disappear. The description tells us not to expect any immediate change to the graphics on-screen (the new factor only affects arrow we draw after changing the length factor). Similarly, we may use the *select_radius* event on the letter "r" key to specify a new radius for future arrows.

2. TOOL DESCRIPTIONS

2.1 Database management tools

Several tools construct simple geometry directly from a molecular database or update a database using on-screen geometry.

db_all_bonds, db_ca_sticks, db_h_bonds and db_het_bonds

This following four tools are similar and are grouped together for convenience. These routines, which generate no graphics on-screen until they are completed, may require quite a while to draw a large molecule (10-30 minutes).

The geometry generated by these tools may not display until the tool has completed. Also, the geometry may appear off-screen (the tools do not automatically rescale geometry to fit the screen). To see the geometry, click **Recenter** in the **Main** panel.

These tools create one or more groups with names indicating the contents, such as "main_chain" or "h_bonds". The groups produced will not have the name of the tool.

The results of these tools is:

db_all_bonds,	Draws all bonds of a molecule.
db_ca_sticks	Draws line segments connecting successive alpha carbons along the main chain of a molecule.
db_h_bonds	Draws the hydrogen bonds of a molecule.
db_het_bonds	Draws bonds between "het" (heterogenous) atoms of a molecule. Both atoms in the bond must be "het".

System Prompt	User Action	System Response
Query panel appears with prompt: *Enter database to be read.* Dummy file name appears as an example.	Enter full path name for a file.	Depending on the tool used, draws line segments to represent: all bonds of a molecule, or connections between sucessive alpha carbons of a molecule, or hydrogen bonds of a molecule, or bonds between "het" atoms of a molecule.

update_database

This tool updates molecular databases to reflect on-screen geometry. It is useful when geometry has been moved (using ***rotate_axis*** or ***translate_axis,*** for example). ***update_database*** requires that all on-screen geometry have geometry pointers and be associated with individual atoms (thus ***helix_cylinder*** cylinders would not be valid). This tool only uses cylinders, lines, and spheres to determine atom positions for the update; triangles and text are ignored.

No User Actions.

2.2 Simple molecular sketching tools

These tools are used for drawing simple shapes such as lines, triangles, and spheres based on atom positions.

connect_atoms

Draws a thin cylinder between a pair of atoms.

	System Prompt	User Action	System Response
repeat as often as desired	Select first atom.	Click on an atom.	Red ball appears at the atom position
	Select second atom.	Click on a second atom.	Second red ball appears momentarily. A cylinder connects the atoms together. Red balls disappear.

cyl_atom_sequence

Draws a series of cylinders connecting selected atom positions.

	System Prompt	User Action	System Response
	Select start atom.	Click on an atom.	Red ball appears.
repeat as often as desired	Select next atom.	Click on an atom.	Red ball appears. A cylinder connecting the previous two atom positions is drawn. The red ball at the start of the cylinder disappears.

select_atoms

Displays small spheres at selected atom positions. This is often used in conjunction with other drawing tools that require spheres, such as ***compute_distance*** or ***mid_points***.

	System Prompt	User Action	System Response
repeat as often as desired	Select an atom.	Click on an atom.	Colored sphere appears.

triangle_atoms

Draws a series of triangles connecting sets of three selected atom positions. The triangles have no database pointers assigned.

	System Prompt	User Action	System Response
repeat as often as desired	Select first atom.	Click on an atom.	Red sphere appears.
	Select second atom.	Click on another atom.	Another red sphere appears.
	Select third atom.	Click on third atom.	A third red sphere appears. A triangle is drawn. All three red spheres disappear.

2.3 Tools for displaying molecular geometry

A number of tools construct geometry for all or part of a molecule based on on-screen selections. These representations includes atomic CPK spheres, side and main chain bonds, alpha-carbon connections, backbone ribbons, and splined backbone tubes.

atom_CPKs

Displays CPK spheres at selected atom positions.

	System Prompt	User Action	System Response
repeat as often as desired	Select an atom.	Click on an atom.	Colored CPK sphere with Van der Waals radius appropriate to the atom type appears.

cyl_line_ca

Draws a series of cylinders connecting adjacent alpha carbons between two selected positions on the main chain of a protein.

	System Prompt	User Action	System Response
repeat as often as desired	Select start atom.	Click on an atom.	Red ball appears.
	Select end atom.	Click on a second atom.	Second red ball appears. A series of gold cylinders connect adjacent alpha carbons between the two chosen atoms. Red balls disappear as the cylinders are drawn over them.

cyl_line_main

Draws the mainchain bonds (including carbonyl oxygens) as a set of cylinders between two selected positions on the mainchain of a protein.

	System Prompt	User Action	System Response
repeat as often as desired	Select start atom.	Click on an atom.	Red ball appears.
	Select end atom.	Click on a second atom.	Second red ball appears. A series of white cylinders appear between the two chosen atoms showing main chain and carbonyl oxygen bonds. Red balls disappear.

helix_cylinder

Creates cylinders that lies along the axes of alpha helices. The cylinders have no database pointers.

	System Prompt	User Action	System Response
repeat as often as desired	Select atom at start of helix.	Click on an atom.	Red ball appears.
	Select atom at end of helix.	Click on a second atom.	Second red ball appears. purple cylinder within the helix is drawn. red balls disappear.

residue_CPKs

Displays the atoms as CPK spheres for individual residues, specified by selecting a single atom from that residue.

	System Prompt	User Action	System Response
repeat as often as desired	Select atom from residue to be displayed.	Click on an atom.	Red sphere appears. As residue is drawn, red sphere becomes a CPK sphere with Van der Waals radius appropriate to the atom type.

ribbon

Generates a backbone ribbon for the selected molecule. This tool displays a ribbon several minutes after the user initiates execution. The triangles comprising the ribbon contain no database pointers.

	System Prompt	User Action	System Response
repeat as often as desired	Select the molecule for ribbon.	Click on an object from a group that displays the desired molecule.	Selected object changes color. After several minutes, a ribbon is drawn. Selected object changes to original color.

ribbon_select

Generates backbone ribbon segments for selected portions of molecules. The triangles comprising the ribbon contain no database pointers.

	System Prompt	User Action	System Response
repeat as often as desired	Select start atom for ribbon.	Click on an atom.	Red sphere appears at selected atom.
	Select end atom for ribbon.	Click on an another atom.	Another red sphere appears at selected atom. Red spheres disappear when ribbon is drawn between the atoms.

select_subtree

Displays (as a series of cylinders) the connected set of bonds (subtree) attached to a particular bond. The display proceeds in the direction specified by selecting, in order, the two atoms comprising the start bond.

You must execute care in selecting the bond for this tool. If the selected bond is part of a cyclic structure (for example in a proline, within an aromatic ring, or part of a disulfide bond), the entire molecule may be selected.

	System Prompt	User Action	System Response
repeat as often as desired	Select first atom on start bond.	Click on an atom.	Red sphere appears.
	Select second atom on start bond.	Click on another atom.	Another red sphere appears. Subtree is drawn from first atom to another atom. Red spheres disappear

sidechain_bonds

Displays the bonds (as cylinders) for individual sidechains, specified by selecting a single atom from each sidechain.

	System Prompt	**User Action**	**System Response**
repeat as often as desired	Select atom from residue to be displayed.	Click on an atom.	Red sphere appears. After sidechain is drawn, red sphere disappears.

spline_tube_chain

Displays a spline-like tube between a pair of selected mainchain atoms. The tube follows atoms of a particular type specified by the user. The default type of atom is an alpha carbon. Spline tube geometry has no database pointers assigned.

	System Prompt	**User Action**	**System Response**
	Text area appears with prompt: "Enter atom type for chain following"	Type in an fully qualified atom type or press "return" or "tab" to keep the default of alpha carbon (CA).	No visual response.
repeat as often as desired	Select starting atom.	Click on an atom.	Red sphere appears.
	Select ending atom.	Click on another atom.	Red sphere appears at ending point. Starting sphere disappears and a tube is drawn from the starting point. The ending sphere disappears when the tube reaches it.

2.4 Tools for modifying parameters of existing geometry

A number of tools allow you to alter attributes of existing geometry including their color, size, shading, and position relative to other groups.

avg_tri_normals

Averages the normals of all triangles in a given group that are coincident at a vertex for each triangle and vertex in the group (all primitives in the group should be triangles). This is a technique that is commonly used to produce smoother shading when the surface normals at the vertices are not known (e.g. the tool that produces the triangles does not assign per-vertex normals). See section 4.1 of *VIEW User Interface Description* for more on the use of normals in controlling shading of triangles.

System Prompt	User Action	System Response
Select a triangle from the group to be averaged.	Click on a triangle from the group to be averaged	Triangle changes color. Triangles become smooth-shaded and the selected triangle reverts to its original color.

change_radius_group

Changes the radii of all objects in a group to the same new radius.

	System Prompt	User Action	System Response
	Select an object with initial radius	Click on an object	No visual response
	Query panel appears with prompt: *Radius to be applied to groups.* Number appearing is the radius of the selected object.	Specify the new radius.	No visual response
repeat as often as desired	Select an object from the group to be changed.	Click on an object	Radius of all objects in the group changes to the new radius.

change_radius_object

Changes the radii of individual objects to the same new radius.

	System Prompt	User Action	System Response
	Select an object with initial radius.	Click on an object	No visual response.
	Query panel appears with prompt: *Radius to be applied to objects.* Number appearing is the radius of the selected object.	Specify the new radius.	No visual response.
repeat as often as desired	Select an object to be changed.	Click on an object.	Radius of the object changes to the new radius.

Optional Events

change_rad_dial	Continuously changes the radius of the selected object by rotating a dial.

User Action	System Response
Rotate dial 7 clockwise, or	Radius continuously increases.
Rotate dial 7 counterclockwise	Radius continuously decreases.

recolor_group

Sets all objects in a geometry group to a specified color

	System Prompt	User Action	System Response
	Select object with initial color.	Click on an object	No visual response.
	A query with three text areas appears with the prompt: *(red, green, blue) color to be applied to groups* Numbers appearing in the these text areas denote the exisitng intensity of the red, green and blue components of the object selected. The scales is from 0 to 255.	Type in a new number for any or all of the three color components. Click on **OK** in the query when color specification is as you desire.	Query disappears. No visual response in the display panel. The new color specified will apply to subsequent groups selected.
repeat as often as desired	Select an object from the group to be recolored.	Click on an object.	Entire group containing the selected object changes color.

recolor_object

Sets all selected objects to a specified color

	System Prompt	User Action	System Response
	Select object with original color.	Click on an object.	No visual response.
	A query with three text areas appears with prompt: *(red, green, blue) color to be applied to objects* Numbers appearing in the these text areas denote the existing intensity of the red, green and blue components of the object selected. The scales are each from 0 to 255.	Type in a new number for any or all of the three color components. Click on **OK** in the query when color specification is as you desire.	Query disappears. No visual response in the display panel. The new color specified will apply to subsequent objects selected.
repeat as often as desired	Select object to be recolored.	Click on an object.	Selected object changes color.

rotate_axis

Rotates all geometry in selected groups around a selected axis.

	System Prompt	User Action	System Response
	Select the axis to rotate about.	Click on an line or cylinder.	Selected line or cylinder changes color.
repeat as often as desired	Select a group for rotation.	Click on an object in a group.	Selected object changes color and the group that contains it becomes part of the set of groups that respond to rotation events (listed below).

Optional Events

def_angle	Defines the rotation amount. Default is one degree.

User Action	System Response
Press key "s"	Query panel appears with prompt: *Enter rotation angle*
Type in a rotation angle. Positive numbers denote clockwise rotation angles. Negative numbers denotes counterclockwise rotation angles	No visual response until user presses other event keys.

Or:

User Action	Event Name	System Response
Press key "r"	*rotate_grps*	Rotates the geometry clockwise by the defined amount of rotation.
Press key "e"	*rotate_grps_back*	Rotates the geometry counterclockwise by the defined amount of rotation.
Press key "a"	*auto_rotate*	Automatically rotates the geometry back and forth 30 degrees.
Press key "p"	*auto_rot_stop*	Stops rotation of the geometry started by the *auto_rotate* event.
Press key "x"	*reset_rot*	Resets the geometry to its original position.
Press key "f"	*rotate_forward*	Rotates the geometry counter-clockwise 30 degrees. Once activated, this event will have no further effect on re-execution until the rotate_backward event is executed.
Press key "b"	*rotate_backward*	Rotates the geometry clockwise 30 degrees. Once activated, this event will have no further effect on re-execution until the rotate_forward event is executed.

translate_axis

Translates all geometry in selected groups along a selected axis.

	System Prompt	User Action	System Response
	Select the axis to translate along.	Click on an line or cylinder.	Selected line or cylinder changes color.
repeat as often as desired	Select a group for translation.	Click on an object in a group.	Selected object changes color and the group that contains it becomes part of the set of groups that respond to translation events (listed below).

Optional Events

def_amount	Defines the amount of movement of a group along a selected axis. The default is .1 world space units (angstroms if the group selected represents a molecule)

User Action	System Response
Press key "d"	Query panel appears with prompt: *Enter the translation amount*
Type in a translation amount or leave default.	No visual response until you presses other event keys.

Or:

User Action	Event Name	System Response
Press key "t"	*translate_grps*	Moves the groups along the selected axis by amount defined by key "d" or by the default.
Press key "y"	*translate_grps_down*	Moves the group along the selected axis in the opposite direction from key "t".

2.5 Euclidean construction tools

A set of tools is designed for performing 3-D Euclidean constructions. Such constructions include drawing a line between two points, drawing a plane normal to a line through a given point, projecting a point onto a line or a plane, displaying the intersection of a line and a plane, and displaying the closest approach of two 3-D lines.

Many of these tools treat on-screen geometry as representations of idealized geometry. Thus, a cylinder, even though of fixed length, may be used to represent an infinte line. For this reason, some of the tools produce intersection and projection points that do not lie within the boundaries of the geometry selected to represent a line or a plane.

None of the geometry created by Euclidean construction tools has database pointers assigned.

connect_points

Draws a thin cylinder between the center points of a pair of objects.

	System Prompt	User Action	System Response
repeat as often as desired	Select first object.	Click on an object.	Object changes color.
	Select second object.	Click on a second object.	Color of the second object changes momentarily. First object changes to original color. A cylinder connects the object centers.

intersect_plane_line

Displays a sphere at the intersection of a planar surface and a line. The planar surface may be a plane produced by the ***normal_plane_thu_sph*** tool (which consists of two triangles) or a triangle drawn by some other tool.

System Prompt	User Action	System Response
Select the plane	Click on a planar surface (a plane or a triangle).	If the surface chosen is a triangle, then the triangle changes color. If the surface chosen is a plane, then one of the triangles forming the plane changes color.
Select the line.	Click on a line or cylinder.	The line or cylinder momentarily changes color. Selected triangle changes to original color. A white ball appears at the intersection of the plane and the line.

lengthen_line

Lengthens or shortens selected lines segments (or cylinders) by a user-specified factor. Lines are lengthened or shortened equally beyond the original position at each end. Spheres at the ends of the newly lengthened lines are displayed on-screen and are placed in a group called "lengthen_line_sphs" (which is added to **Group Operations** panels). Note that this tool, unlike many of the other Euclidean construction tools, does not treat the selected geometric primitive as a representation of an infinite line.

	System Prompt	User Action	System Response
repeat as often as desired	Select the line.	Click on a line or cylinder.	Original line is lengthened and balls mark either end of the lenghtened line.

Optional Events

change_length	Specifies the scale factor to be used for lengthening. The default, 1.2 lengthens the line to 120% of the size of the original line Values less than one will shorten the line. On-screen geometry does not change when this event is completed; the new scaling factor will be applied to subsequent lines selected.

User Action	System Response
Press letter key "l"	Query panel appears with prompt: *Length scaling factor:*
Choose a length scaling factor of less than one, or	Future lines selected will be redrawn shorter than the original.
Choose a length scaling factor of greater than one.	Future lines selected will be redrawn longer than the original.

line_ends

Creates spheres to mark the ends of lines (or cylinders). This tool may be used in conjunction with other tools that use sphere centers as positions (such as *compute_distance*).

	System Prompt	User Action	System Response
repeat as often as desired	Select the line.	Click on a line or cylinder between two atoms.	New spheres mark either end of the line.

line_intersect

Displays the intersection point of two lines as a sphere. If the lines do not intersect, the points on each line that are closest to each other are marked with a sphere and a cylinder connecting the two spheres is drawn.

System Prompt	User Action	System Response
Select the first line.	Click on a line or cylinder	Selected line changes color.
Select the second line.	Click on a second line or cylinder	Selected line changes color momentarily. First line changes to original color. Either a single sphere is displayed (if the lines intersect), or a sphere for each line with a connecting cylinder is displayed

mid_pnts

Displays a sphere marking the midpoint between the center points of two selected objects.

	System Prompt	User Action	System Response
repeat as often as desired	Select the first object.	Click on a object.	Selected object changes color.
	Select: second object.	Click on a second object.	Selected object changes color momentarily. First object changes to original color. Mid-point sphere is displayed.

normal_plane_through_pnt

Constructs a plane through a selected object center perpendicular to a selected line (or cylinder).

System Prompt	User Action	System Response
Select the object to pass through.	Click on a object.	Object changes color.
Select the line.	Click on line or cylinder.	Selected line changes color momentarily. First object changes to original color. A plane is displayed that passes through the selected object center and that is perpendicular to the selected line.

proj_pnt_onto_line

Constructs a sphere representing the perpendicular projection of a selected object's center onto a selected line (or cylinder).

System Prompt	User Action	System Response
Select the object.	Click on a object.	Object changes color.
Select the line.	Click on line or cylinder.	Selected line changes color momentarily. First object changes to original color. A sphere is displayed that is the perpendicular projection of the selected object's center onto the selected line.

proj_pnt_onto_plane

Constructs a sphere representing the perpendicular projection of a selected object's center onto a planar surface. The planar surface may be a plane produced by the ***normal_plane_thu_sph*** tool (which consists of two triangles) or a triangle drawn by some other tool.

System Prompt	User Action	System Response
Select the object.	Click on a object.	Object changes color.
Select the plane.	Click on plane or triangle.	Selected triangle changes color momentarily. First object changes to original color. A sphere is displayed that is the perpendicular projection of the selected object's center onto the selected plane.

triangle_pnts

Draws a series of triangles connecting the centers of three selected objects.

	System Prompt	User Action	System Response
repeat as often as desired	Select first object.	Click on a object.	Selected object changes color.
	Select second object.	Click on another object.	Selected object changes color. First object changes to original color.
	Select third object.	Click on third object.	Selected object changes color momentarily. Second object changes to original color. A triangle is drawn.

2.6 Annotation tools

Several of the tools are for annotating a drawing. The two forms of annotation supported are arrows and text.

arrow_atoms

Draws a series of arrows between pairs of atoms. Arrow geometry has no database pointers.

	System Prompt	User Action	System Response
repeat as often as desired	Select atom for arrow tail	Click on an atom	Red ball appears at the atom position
	Select atom for arrow head	Click on a second atom	A red ball appears at the second atom position. Arrow is drawn connecting the atoms, then the red balls disappear.

Optional Events

select_length — Specifies whether the arrows are to connect the atoms, go beyond each atom, or fall short of each atom.

User Action	System Response
Press letter key "l"	Query panel appears with prompt: *Length scaling factor:* Displayed number is the previously defined scaling factor.
Choose a scaling factor of one (this is the default value when starting this tool) or	Future arrows will connect the two selected atoms together.
Choose a length scaling factor of less than one or	Future arrows will fall short of each of the two selected atoms.
Choose a length scaling factor of greater than one.	Future arrows will extend beyond each of the two selected atoms.

select_radius — Specifies the radius of the shaft for future arrows.

User Action	System Response
Press letter key "r"	Query panel appears with prompt: *Specify radius for shaft.* Displayed number is the previously defined radius.
Choose a number for the radius of the arrow shaft.	Radius of arrow shaft will change to the new value for future arrows.

arrow_pnts

Draws a series of arrows between the center points of a pair of selected objects. Arrow geometry has no database pointers.

	System Prompt	User Action	System Response
repeat as often as desired	Select object for arrow tail	Click on an object	Object changes color
	Select object for arrow head	Click on a second object	Object changes color momentarily. First object changes to original color. Second object changes to original color. Arrow is drawn connecting the object's center points.

Optional Events

The *select_length* and *select _radius* events are identical to those for *arrow_atoms.*

text_atom_wqual, text_atomnum, text_resname, text_resnum

These following four tools are similar and are grouped together for convenience. Execution of each of these tools results in the appearance of a text label near the atom/s selected. Text is of fixed size and orientation. It is always drawn in the plane of the screen. The text labels contain:

text_atom_wqual	Displays the fully qualified atom name (e.g. "CA" or "CG1").
text_atomnum	Displays the atom number.
text_resname	Displays the 3-letter residue name.
text_resnum	Displays the residue number.

	System Prompt	User Action	System Response
repeat as often as desired	Select atom to annotate.	Click on an atom.	Depending on the tool used, text labels appear with one of the following: atom name, or atom number, or residue name, or residue number.

text_userlabel

This tool is similar to the above tools, but the text label contains text that the user supplies.

	System Prompt	User Action	System Response
repeat as often as desired	Query appears with prompt: "Text to be displayed"	Type in text.	No visual response.
	Select atom.	Click on an atom.	Text label appears with text that user has chosen.

2.7 Measurement tools

Several tools allow you to measure angles and distances.

compute_angle

Computes the inner bond angle between two linear objects (lines or cylinders) that meet at a common point.

	System Prompt	User Action	System Response
repeat as often as desired	Select first line.	Click on a line or a cylinder	Selected line or cylinder changes color.
	Select second line.	Click on a second line or cylinder that is connected to the first line or cylinder. or	Selected second line or cylinder changes color. First object changes to original color. The message: *angle=n,* where n is the computed inner angle, is printed in the window from which VIEW was run.
		Click on a line or cylinder that is not connected to the first selected line or cylinder.	No visual response. No error message appears and no angle is computed.

compute_dihedral

Computes the dihedral angle between the center points of four selected objects. The dihedral is computed using the four points in the order selected.

	System Prompt	**User Action**	**System Response**
repeat as often as desired	Select first object.	Click on an object.	Object changes color.
	Select second object.	Click on a second object.	Selected second object changes color. First object changes to original color.
	Select third object.	Click on a third object.	Selected third object changes color. Second object changes to original color.
	Select fourth object.	Click on a fourth object.	Selected fourth object changes color. The message: *Dihedral angle = n* where n is the computed angle, is printed in the window from which VIEW was run.

compute_distance

Computes the distance between the center points of two selected objects.

	System Prompt	User Action	System Response
repeat as often as desired	Select first object.	Click on a object.	Object changes color.
	Select second object.	Click on a second object.	Selected second object changes color. First object changes to original color. The message: distance between points = n where n is the computed distance is printed in the window from which VIEW was run.

2.8 Geometry management tools

Several tools are provided for deleting geometry, duplicating groups of geometry, and merging geometry groups.

delete

Deletes geometric primitives (cylinders, lines, spheres, triangles, text) from the geometry group of which they are a part. See section 1.1 for a discussion of the relationship between geometry generated by tools and individual primitives.

	System Prompt	User Action	System Response
repeat as often as desired	Select object to delete.	Click on an object.	Object is deleted

dup_group

Creates a duplicate of a selected geometry group. Attempting to give the new group the same name as an already-existing geometry group causes the tool to fail.

System Prompt	User Action	System Response
Select object from group to be duplicated	Select an object.	Selected object changes color.
Query panel appears with prompt: *Enter the name for the duplicate group.* Dummy group name "dup" appears as an example to the user.	Enter group name, or leave the name "dup".	Object changes to original color. Duplicate group is added to **Group Operations** panels.

merge_groups

Combines geometry in two or more groups into a single group. One group (the first selected) is be designated as the merge group. This is the group that will contain all merged geometry. The other groups specified have their geometry moved into the merge group and are then deleted.

	System Prompt	User Action	System Response
	Select initial group.	Click on an object from the merge group.	Selected object changes color.
repeat as often as desired	Select group to merge.	Click on an object from the group to be merged.	Selected object changes color momentarily. Previous object changes to original color. Geomety from the group is transferred to the merge group. No visible effect on-screen.

2.9 Display and system management tools

A tool is available that creates a keyboard interface for geometry movement. Another repositions geometry in the center of the screen. Finally, a tool that removes all defined events is available.

move_world

Rotates, moves or scales all geometry (the world) using keys. This provides a keyboard interface which serves as an alternative to the mouse-based virtual trackball (see Section 2.1.2 of *VIEW User Interface Description* for a description of the virtual trackball). The tool itself produces no changes to the display and involves no user interaction; all movement using this tool is done through the use of optional events.

Optional Events

User Action	Event Name	System Response
Press key "h"	*trans_x_left*	Moves the geometry to the left on the screen.
Press key "l"	*trans_x_right*	Moves the geometry to the right on the screen.
Press key "j"	*trans_y_down*	Moves the geometry downwards on the screen.
Press key "k"	*translate_y_up*	Moves the geometry upwards on the screen.
Press key "i"	*translate_z_in*	Zooms in on the geometry
Press key "o"	*translate_z_out*	Zooms out on the geometry
Press key "a"	*rot_x_clock*	Rotates the geometry clockwise around the x-axis
Press key "f"	*rot_x_cc*	Rotates the geometry counter-clockwise around the x-axis
Press key "s"	*rot_y_clock*	Rotates the geometry clockwise around the y-axis
Press key "d"	*rot_y_cc*	Rotates the geometry counter-clockwise around the y-axis
Press key "w"	*rot_z_clock*	Rotates the geometry clockwise around the z-axis
Press key "e"	*rot_z_cc*	Rotates the geometry counter-clockwise around the z-axis
Press key "y:	*scale_down*	Reduces the amount of movement or zooming resulting from each keystroke.
Press key "u"	*scale_up*	Increases the amount of movement or zooming resulting from each keystroke.
Press key "r"	*reset*	Resets the amount of movement or zooming resulting from each keystroke to system default.

remove_events

Removes all defined events

No user actions

set_origin

Resets the origin of the world coordinate system (the center for virtual trackball rotations) to the center of a selected object. The selected center is moved to the center of the screen.

	System Prompt	User Action	System Response
repeat as often as desired	Select object to recenter on.	Click on an object.	Object momentarily changes color and is centered in the display panel.

Tool Index

VIEW

Interactive Tool Definition Language

Language Description

Table of Contents

VIEW Interactive Tool Definition Language - Language Description

Larry Bergman
1/24/93

1. INTRODUCTION

This document describes the language that defines drawing tools in the VIEW system. The language will be referred as the *tool language.* This document describes the syntax and semantics of the tool language. It does not describe how to run or debug a tool; the development environment for tools is described in the document, *VIEW Interactive Tool Definition Language - Development Environment.*

Users of the tool language are assumed to have programming experience. Knowledge of either FORTRAN or the C programming language is helpful. Furthermore, use of the development environment requires experience with an interactive debugger (such as dbx). Experience with a screen-based editor (such as MacWrite) will be very helpful.

Before using this document, you should have read *VIEW Exploratory Molecular Visualization System - Overview* That document covers basic concepts of the VIEW system. This document will assume that you are familiar with these underlying ideas and definitions. We will also assume that you have read *VIEW User Interface Description,* which describes operation of the VIEW system.

A drawing tool definition consists of a sequence of program statements. The basic types of statements are:

Category	Statement type
Variable value definition	assignment
	concatenation
	remove
Flow control	exit
	if-else
	iteration
	sleep
Input/output	report
	user_query
Display control	display
	undisplay
Object selection (picking)	select
Subroutine definition and control	parameter definition
	subroutine call
	subroutine return
Datatype manipulation	database manipulation
	geometry group manipulation
	geometric object manipulation
	set manipulation
Events	event definition
	event control

The syntax used in this document is as follows: a portion of a statement that will be used elsewhere or is described elsewhere (called a *non-terminal* in computer-science jargon) is enclosed in angle brackets, <>. For example, <simple variable>. Required keywords or symbols are displayed in **bold face** type. For example, **POINT** or **(**. Some statements have more than one form. In this case, the alternate forms are listed separated by a vertical bar, |. Some statements have optional parameters. These are enclosed in square brackets, []. If a command has a series of optional parameters at the end of the command, you must specify all parameters that precede the last parameter you choose to supply. Thus, if the statement is specified as:

SPHERE (<point> **,** <radius> [**,** <color>] [**,** <tesselation> **)**

You may specify

```
       SPHERE (pnt1, rad)
       SPHERE (pnt1, rad, color1)
    or SPHERE (pnt1, rad, color1, tess)
```

where *pnt1* is a point, *rad* is a numeric radius, *color1* is a color, and *tess* is a numeric tesselation. but not

```
       SPHERE (pnt1, rad, tess)
```

In this last example, the system will try to use the variable *tess* as the color.

The previous example uses variable names as they are used throughout the document. Our variable names are chosen to be descriptive of the datatypes they contain or of their use. Thus, in the first SPHERE example above, the variable *pnt1* was selected for the center point, and the variable *rad* was selected for the radius. When you write your own tools, any variable names that contain the appropriate datatypes may be used.

The term *object* will be used throughout this document to refer to data elements. An object might be a point, a number, a database, or any other datatype.

2. STATEMENTS

The syntax for language statements generally follows the C language. All statements are one of two forms (and a few types of statements may be either). The first is:

<statement> ;

where <statement> is one of a group of language statements, such as an **IF** statement, or an assignment statement. The tool definition language is slightly more restrictive than C — <statement> may not be null.

EXAMPLES:

```
a = b + 5;
IF (a < 3) c = 2;
grp_list &= grp;
```

The second form is:

<statement header> { <statement body> }

where <statement header> is the initial portion of certain statements such as **FOR** statements and **IF** statements. <statement body> is a single language statement or a sequence of language statements, one after another. Again, the tool definition language is more restrictive than C — <statement header> may not be null, nor may <statement body>.

EXAMPLES:

```
FOR (i=0; i<4; i=i+1) a &= b[i] + 5;


IF (EXISTS(a))
{       REPORT(a);
        b = SIN(a+PI/2);
}
```

The language supports comments. Comments always start with an exclamation mark; the remainder of the line following the exclamation mark is treated as a comment. The single exception is an exclamation mark immediately followed by an equals, which is the "is-not-equal" relational operator.

EXAMPLE:
```
a = 5;   ! comment - this is a simple assignment
```

Comments will be used throughout this document to annotate examples.

3. VARIABLES

Variables may store any of the datatypes in the language (enumerated in the next section). A statement such as:

```
pnt = POINT(1., 0., 1.);
```

defines a variable named *pnt* of type POINT. Type declarations are not required in the language, and any variable may store any type.

A simple variable (referred to later in this document by <simple variable>) consists of an arbitrary sequence of characters from the set [a-z][A-Z][0-9] and underscore, _ . A simple variable name must begin with an alphabetic character.

EXAMPLES:

```
num
sph_count2
```

There are reserved words in the language that may not be used as simple variables. Reserved words are all written entirely in uppercase. Since the language is case sensitive, if you use some lowercase letters in each variable name, you will never accidentally use a reserved word. The reserved words are:

ACOS	ALIAS	ALL
AND	ARRAY	ASIN
ASK_FILE	ASK_NUMBER	ASK_STRING
ATAN	BACKCOLOR	CENTERPOINT
COLOR	CONTAINS	COPY
COS	CROSS	CURR_TOOL
CYLINDER	DB	DB_ADD_RECORD
DB_ADD_SUBSET	DB_PTR	DB_READ
DB_REMOVE_FIELD	DB_REMOVE_RECORD	DB_WRITE
DEFAULT	DIAL	DIALNUM
DIALRATE	DIALVAL	DIRECTORY
DISPLAY	DISPLAY_GEOMETRY	DIST
DO	DOWN	ELSE
END	END_KEY	EOD
EVENT	EVENT_REMOVE	EXISTS
FALSE	FLATCAP	FOR
FOREACH	GEO_READ	GEO_WRITE
GROUP	GROUP_NAME	IF
IN	INDEX	IN_OUT
KEY	KEYCHAR	LEFT
LINE	LOOP	MOD
MOUSE	MOUSE_BUTTON	MOUSE_POSITION
NEXT_RECORD	NOCAP	NOREDRAW
NOT	OBJECT	ON
OR	ORIGIN	OUT
PARAMETERS	PAUSE	PI
POINT	PREV_RECORD	RECORD
REDRAW	REMOVE	REPORT
RESET_TRANSFORMATIONS	RETURN	RIGHT
ROTATE	ROTATE_SCREEN	ROUND
SCALE	SCALE_SCREEN	SCREEN
SEARCH3D	SELECT	SELECT_DB
SET	SHOWDICT	SIN
SPHERE	SPHERECAP	SQRT
START	START_EVENT	START_KEY
STOP	STOP_EVENT	TAN
TEXT	TO	TRANSLATE
TRANSLATE_SCREEN	TRIANGLE	TRUE

UNDEFINED	UNDISPLAY	UNDOABLE
UNINTERRUPTABLE	UNTIL	UP
VECTOR	WHILE	WITH

Variables may be modified to specify access of fields, attributes, records, and array elements. There are four forms that such a modification can take:

1) The variable name may be followed by an expression in brackets indicating indexed array access.

   EXAMPLE: a[3]

   indicates that the third element of the array *a* is to be used.

2) The variable name may be followed by an expression in parentheses indicating keyed array access.

   EXAMPLE: a("num")

   indicates that the element of the array *a* which has the string-valued key, "num", is to be used.

3) The variable name may be followed by an period and then a field or attribute name.

   EXAMPLE: rec.position

   indicates that a field (if *rec* is a database record) or an attribute (if *rec* is some other variable type) called "position" is to be accessed from the variable *rec*.

4) The variable name may be followed by a period and then a keyed access (as described in 2 above). This syntax specifies record access from a database.

   EXAMPLE: dbase.atom(5)

   indicates that the record in the "atom" subset with the key value 5 is to be accessed from the database referenced by *dbase.*

These forms may be combined liberally. Thus,

```
a[3].position                  ! retrieves position field from array element.

b[3][4]                        ! two-dimensional array access.

dbase.atom(num).conformation   ! retrieves database field "conformation" from
                               ! "num" record of "atom" subset.
```

```
    item.color.red                 ! retrieves red component of item's color
                                   ! attribute.
```

are all legal.

The term <variable> will be used throughout this document to refer to either a simple variable or a simple variable that has been modified as indicated.

4. DATA TYPES

Statements operate on several different data types. The major data types and their subtypes are:

- numeric
- string
- Boolean (or logical)
- array
  - keyed
  - indexed
- set
- color
- displayable geometry
  - sphere
  - line
  - cylinder
  - text
  - triangle
- internal geometry (not displayable)
  - point
  - vector
- geometry group
- database
- database record
- database field
- attribute

Each of these data types is discussed individually.

4.1 Containers

The tool definition language has four container datatypes. These are arrays, sets, databases, and geometry groups. Containers are used for storing other datatypes. An additional type, records, appears to fit this definition, but record fields are very similar to attributes, rather than objects stored within a container. Since records do not support iteration or concatenation, they will not be considered to be containers.

There are several characteristics shared by all containers. The first is that containers do not own the objects they contain. In other words, an object is pointed to by the container, it is not copied into it (the exception is when the **COPY** construct is used to add to a container). This has an important consequence. A change made to an object, even if through a different variable name, will change that entry in the container. Thus the code segment:

```
obj = SPHERE(pnt,rad,white);
grp &= obj;          ! adds obj to the geometry group grp
obj.COLOR = red;
```

will cause the group *grp* to contain a red sphere.

Secondly, the objects in any container may be accessed using the **FOREACH** iterator. This construct permits you to operate on each and every object within a container (**FOREACH** is described in section 7.6.4).

Containers vary from each other in several ways. The most important is the datatypes that may be stored within them. Databases may only store records; geometry groups may only store displayable geometric objects; while arrays and sets may store any datatype (including other containers).

A second way in which they differ is whether an object stored in a container may also be stored in another container of the same type. In other words, whether an object may have more than one *parent* of a given type distinguishes the containers. The rule is, containers which are restricted to objects of a single type (databases and geometry groups), may not share objects with another container of the same type. Thus a record may only have a single parent database, although the record may also be stored in multiple arrays (the "single parent" rule does not apply to other container types).

A third difference is that some of the containers (indexed array, geometry group) will allow the same object to be inserted more than once, keeping as many references as are inserted. Others (database, set, keyed array), will only store a single reference even if the object is inserted more than once.

A final difference lies in how objects are retrieved for the container. Some of the containers (sets, geometry groups) only support access of their contents through the FOREACH iterator. Others (arrays, databases) have alternate access methods using keys or indices.

The differences between containers are summarized in the table below.

Container type	**Datatype stored**	**Entry able to be storied in more than one container of this type?**	**Duplicate entries obtainable?**	**Individual entries obtainable without an iterator?**
Array	Any	Yes	Yes (indexed) No (keyed)	Yes
Database	Record	No	No	Yes
Geometry group	Displayable geometry object	Yes	No	No
Set	Any	No	Yes	No

4.2 Creation

Objects may be created using *object creators*. An explicit object creator is a built-in function that specifies the type of object that is to be created and describes its attributes (at least enough of its attributes to create it).

EXAMPLES:

```
pnt = POINT(0,0,0);
b = ARRAY();
c[2] = TRIANGLE(pnt1,pnt2,pnt3,tri_color);
tri.normal1 = VECTOR(1,0,0);
DISPLAY (TRIANGLE (POINT(0,0,0), POINT(1,1,1), POINT(0,1,0)));
sph = SPHERE(pnt,rad,COLOR(155,0,0));
```

The simplest types of objects - numeric, string, Boolean - do not have explicit creators. Objects of this type are created implicitly, by using an expression that evaluates to that type. Thus, statements such as:

```
b = TRUE;
rad = (5. 0 + PI)/20;
charstr = "a string";
```

will create variables of type Boolean, numeric, string respectively.

Some of the creators can be used *in-line*. For example the COLOR creator in the statement:

```
sph = SPHERE(pnt,rad,COLOR(155,0,0));
```

is an in-line creator. Creators of container objects (array, set, geometry group, database) would make no sense in-line and can only be used by themselves on the right hand side of an assignment.

Object types which have explicit creators can also be created implicitly. The expressions:

```
mid_pnt = (pnt1 + pnt2)/2;        ! pnt1 and pnt2 are points
crs_vec = CROSS(vec1,vec2);       ! vec1 and vec2 are vectors
```

create a point and a vector respectively.

Array objects may also be created implicitly, by assigning a value to an array element. This is discussed in section 4.3.4 below.

It is important to understand that object creators for those objects where no arguments are specified, are more than just type declarations; they actually produce a new object. An example will help to clarify this. Suppose that we wish to create a dictionary of user-defined terms, with an set of objects to be associated with each term. The code fragment below will prompt the user for each term's name, an then create an empty set for each dictionary entry (the dictionary is implemented as a keyed array).

```
dict = ARRAY();
term_name = "initial";      ! initialization required by ASK_STRING

LOOP
{       ASK_STRING("Enter the name for the next term: ", term_name);
        dict(term_name) = SET();     ! create an empty set for this entry

          . . . code to define set entries . . .

}
```

As many sets will be created as there are elements defined for *dict*.

The explicit creators for each object type are described in the next section.

4.3 Descriptions

4.3.1 Numeric

Numbers may be specified as integers or floats. The language does not restrict the intermingling of these types; all arithmetic is done in floating point, with automatic rounding where appropriate (e. g. when used as an array index); rounding is done using the C function "rint". Integers and floats are defined exactly as in C (or FORTRAN).

EXAMPLES:

```
124
-30
1.763
-1.5E-5
```

A special constant, **PI**, is defined having the value 3.141592654.

4.3.2 String

Strings are used to store sequences of text characters. A string is specified by an arbitrary sequence of characters from the set: space, [a-z][A-Z][0-9]-_. &=+*$#@!~|;:`'?/<>()[]{}%^ surrounded by double quotes , ".

EXAMPLES:

```
"fred"
"$*** this operation is not permitted!"
```

4.3.3 Boolean

A Boolean may take on the value **TRUE** or **FALSE**.

4.3.4 Array

Arrays come in two flavors - keyed and indexed. A keyed array is like a dictionary. A single key value is supplied for each entry. This key value is required both for storage and retrieval of values. For example, the program statements:

```
a("key1") = 4;
b = a("key1");
```

will result in the variable *b* taking on the value 4. The syntax for keyed array access is:

```
<variable name> ( <key value> )
```

where <key value> may be any <arithmetic expression> that evaluates to a numeric, a Boolean, a string, a color, a point or a vector.

EXAMPLES:

```
bonds_dict(num1 + 5)
bonds_dict(5)
pnt_group(POINT(1., 0., 1.))
```

An indexed array is exactly like an array in C.

The syntax for an indexed array is:

```
<simple variable> [ <index value 1> ] [ [ <index value 2> ] ...
    [ <index value n>] ]
```

where the <index value>s may be any <arithmetic expression>. Note that array indexing begins with 0 as in C.

EXAMPLES:

```
bonds[5]
val[num1 + 1][6]
```

If you access a keyed or indexed array entry that does not exist (because it has not been defined), a null value will be returned. These null values are handled as follows:

Context	Result
relational expression	all relational expressions containing a null evaluate to FALSE except for != expressions which are always TRUE.
REPORT, DISPLAY, UNDISPLAY or REMOVE statement	ignored
any other expression or statement	produces a run-time error

An (empty) array is specified by:

ARRAY ( )

No distinction is made in the creator between keyed and indexed arrays. The system determines the array type the first time that an element of that array is assigned.

Arrays can also be created implicitly, by assigning a value to an element of the array. Thus, statements of the form:

```
a[2] = "string one";
b("key") = 5;
```

when not preceded by an explicit creator for *a* or *b*, implicitly create an indexed and a keyed array respectively.

4.3.5 Set

A set stores objects of any type (with the exception of other sets) with the guarantee that no object will be duplicated. Operations that may be performed on sets include union, difference, and intersection. Objects within the set may be accessed through use of an iterator, described in section 7.6 below.

An (empty) set is specified by:

SET ( )

4.3.6 Displayable Geometry

There are several datatypes that represent geometric objects that are displayable in the graphics window. These are cylinder, line, sphere, triangle, and text.

Displayable geometry objects may have one or more database record pointers associated with them. This allows tools to access the database through on-screen geometry (usually using a selection). Setting and retrieving these pointers is described in section 7.16.13 below.

Descriptions of each of the displayable geometric object types follows.

4.3.6.1 Cylinder

A cylinder is specified by:

CYLINDER (<point1> , <point2> , <radius> [, <color>] [, <cap_type1> , <cap_type2>] [, <tesselation>])

where <point1> and <point2> are <point expressions> that define the ends of the cylinder axis. <radius> is a <numeric expression> that defines the cylinder radius. <color> is a <color expression> that will be used to assign a color to the cylinder. <cap_type1> and <cap_type2> specify for each end whether they are to have no cap, a flat cap, or a spherical cap. Specification is by using the keywords: **NOCAP**, **FLATCAP**, **SPHERECAP**. <tesselation> is a <numeric expression> (treated as an integer) that specifies the number of quadrilaterals that are to be used in tiling the cylinder (it does not specify the tesselation factor for spherical caps, which is fixed).

EXAMPLES:

```
CYLINDER(POINT(x1,y1,z1), POINT(x2,y2,z2), 5.0)
CYLINDER(pnt1, pnt2, rad, color, FLATCAP, SPHERECAP, 10)
```

4.3.6.2 Line

A line is specified by:

LINE (<point1> , <point2> [, <color>])

where <point1> and <point2> are <point expression>s that define the start and end points of the line respectively. <color> is a <color expression> used to assign a color to the line.

EXAMPLES:

LINE (pnt1, pnt2, line_color)
LINE (POINT(x1,y1,z1), POINT(x2,y2,z2))

4.3.6.3 Sphere

A sphere is specified by:

SPHERE (<point> , <radius> [, <color>] [, <tesselation>)

where <point> is a <point expression> that defines the center of the sphere. <radius> is a <numeric expression> that defines the sphere radius. <color> is a <color expression> used to assign a color to the sphere. <tesselation> is a <numeric expression> (treated as an integer) that specifies a tesselation factor for the sphere. Tesselation should be between 1 and 30, the higher the number, the finer the tesselation. The default tesselation is five. The more highly tessellated spheres will appear smoother, but will take longer to draw. If you have a lot of spheres to draw, you may wish to reduce the tesselation factor to retain interactive response of the display. The following table gives the number of triangles used to tesselate each sphere for selected values of <tesselation>.

<tesselation> value	**number of triangles per sphere**
1	8
2	36
3	80
4	140
5	216
6	308
7	416
8	540
9	680
10	836

EXAMPLES:

```
SPHERE(pnt1, rad)
SPHERE(POINT(x1,y1,z1), 2.0, COLOR(0,100,255))
```

4.3.6.4 Text

A text object is specified by:

TEXT (<string> , <point> [, <color>] **)**

where <string>, a <string expression>, is the string to be displayed. <point> is a <point expression> that specifies where the lower left-hand corner of the text is be located. <color> is a <color expression> used to assign the color of the text.

EXAMPLES:

```
TEXT("SHEET", pnt1, color)
TEXT(dbase.atom(atom_num1).res_name, POINT(x1,y1,z1))
```

Text is always displayed parallel to the screen at a fixed size; it does not change orientation or size when the display geometry is rotated or scaled. Text cannot be reliably selected.

4.3.6.5 Triangle

A triangle is specified by:

TRIANGLE (<point1> , <point2> , <point3> [, <color>] [, [<back_color>] **)**
TRIANGLE (<point1> , <point2> , <point3> , <normal1> , <normal2> , <normal3> [, <color>] [, <back_color>] **)**

where <point1>, <point2>, and <point3> are <point expression>s that define the vertices of the triangle. <color> is a <color expression> used to assign a color to the front face of the triangle. If <back_color> is specified, it assigns the back face color for the triangle. Otherwise, <color> is applied to the back face. <normal1>, <normal2>, <normal3> are <vector expression>s that specify the unit normals (i.e., these normals must be of length one) to be assigned to the vertices at <point1>, <point2>, <point3> respectively. (Remember: different unit normals may be assigned to triangle vertices so that they may be shaded to look like curved surfaces. See section 4.1 of *VIEW User Interface Description* for a more detailed discussion of the use of surface normals in creating smoothly-shaded geometry.

EXAMPLES:

```
TRIANGLE(pnt1, pnt2, pnt3, color)
TRIANGLE(pnt1, pnt2, pnt3, COLOR(0,100,255), COLOR(0,150,155))
```

TRIANGLE(pnt1, pnt2, pnt3, normal1, normal2, normal3)
TRIANGLE(pnt1, pnt2, pnt3, normal1, normal2, normal3, color)

4.3.7 Geometry Group

Geometry groups are containers that store displayable geometry objects. They allow a set of geometric objects to be treated as a unit. Groups may be toggled in the display, removed, written, or renamed through the user interface. Additionally, groups may have 3-D transformations applied to them (as described in section 7.17). Displayable geometry objects belong to one and only one group; no nesting of groups is supported.

Note that GROUP is a datatype, not an operator; you should not confuse the term with the "group" function in a drawing program such as MacDraw.

Each geometry group has an associated label. This label is displayed in the **Group Operations** main subpanel.

The group creator:

GROUP (<group label>)

will produce an empty group with the label <group label> (where <group label> is any <string expression>) if no group with that label already exists. If the group already exists when the creator is executed, the existing one will be retrieved; a new group will not be created. For example,

```
grp = GROUP("sphs");
```

The variable *grp* will contain an empty group which has the label "sphs". If the group "sphs" already existed when this statement was executed, the variable *grp* would be assigned that preexisting group.

Displayable geometric objects are added to a group the using the concatenation operator described in section 7.2.

4.3.8 Color

Colors store red, green, blue triples (rgb) used to assign colors to displayable geometric objects.

A color is specified by:

COLOR (<red> , <green> , <blue>)

where <red>, <green>, and <blue> are numeric expressions (rounded to integers by the system) with values between 0 and 255 that specify the red, green, and blue components of the color respectively.

EXAMPLES:

```
COLOR (0, 255, 155)
COLOR (r1, g1, b1)
```

4.3.9 Internal Geometry

Two types of geometric objects are available which are not directly displayable in the graphics window (they are displayable using debugger functions, but not directly from within a tool), but are useful for performing geometric calculations.

Points and vectors are very similar entities. Both can be expressed as a triple of numbers – in a point, the (x,y,z) triple represents a location, in a vector, it represents an offset. For this reason, VIEW is quite lax in enforcing a distinction between them; it is often possible to mix points and vectors freely in expressions or interchange them as arguments to functions. I recommend that you do not do this carelessly, however. Since they really do represent different mathematical entities, it is wise to think about which you wish to use, and write code accordingly. It is often beneficial to be able to mix points and vectors, but it should be done purposefully.

4.3.9.1 Point

a point is specified by:

POINT (<x> , <y> , <z>)

where <x>, <y>, and <z> are numeric expressions that specify the x, y, and z coordinates of the point respectively.

EXAMPLES:

```
POINT(1.5, 2.0, 10.0)
POINT(x1, y1, z1)
```

If you want to display a point, use a small sphere.

4.3.9.2 Vector

a vector is specified by:

VECTOR (<start point> , <end point>)

where <start point> and <end point> are numeric expressions that specify the starting and ending positions of the vector respectively.

EXAMPLES:

```
VECTOR(pnt1, pnt2)
VECTOR(POINT(1.5, 2.0, 10.0), POINT(3.5, 5.0, 5.5))
```

When vectors are printed (using REPORT or the display debugger function), they are printed as an (x,y,z) triple representing the difference between the start and end points.

If you want to display a vector, use a line or a cylinder.

4.3.10 Database

The most common use of databases in VIEW is for accessing molecular data. Molecular databases contain information about each atom and each bond in the molecule. The database datatype is much more general, however, and can be used to store any data that can be represented as a sequence of records.

Databases contain zero or more *subsets*. A subset contains a set of records, each containing the same fields. For example, a database may have an "atom" subset, each record of which contains an "x", "y", "z", "atom_num" and "res_num" field. Each field has a *name* (e. g. "x") and a *value* (e. g. 3.1). Each record in a subset has a single special field called the *key*. The key is used to access records within the subset. Values for this key must be numeric (automatically rounded to integer) or string values.

A database is specified by:

DB ( )

Program constructs that are used for adding subsets to databases and records to subsets are described in section 7.16.

Molecular databases supplied with the system each contain two subsets, one called "atom", which stores a record for each atom in the molecule, the other called "bond", which stores a record for each bond in the molecule. The *pdbtoview* program, which converts PDB datasets into VIEW database format, creates the following fields for each database record. All molecular field names are in lower case.

"atom" subset

atom_num	atom number
atom_type	atom type. "A" = atom, "H" = hetatom
atom_element	element
atom_wqual	fully qualified atom name. e. g. "CA", "CG1"
alt_conf	alternate conformation identifier. "A" - normal conformation,

	"B" - alternate conformation
res_name	three-letter residue name
res_chain_id	chain id. e. g. "A", "B"
res_num	residue number
x	x-coordinate of atom position
y	y-coordinate of atom position
z	z-coordinate of atom position
occupancy	occupancy
temp_fact	temperature factor
H_bond_donor	whether a hydrogen bond donor. "Y" = yes, "N" = no
conformation	conformation: "HELIX", "SHEET", "TURN" or "?"
atom_structure	alternate coding for conformation. "A" = helix, "B" = beta sheet, "C" = turn or unknown
bond_num1	bond number (in "bond" subset) of the first bond that this atom participates in. "0" if no such bond
bond_num2	bond number (in "bond" subset) of the second bond that this atom participates in. "0" if no such bond
bond_num3	bond number (in "bond" subset) of the third bond that this atom participates in. "0" if no such bond
bond_num4	bond number (in "bond" subset) of the fourth bond that this atom participates in. "0" if no such bond

"bond" subset

bond_num	bond number
atom_num1	atom number (in "atom" subset) of the first atom that comprises this bond
atom_num2	atom number (in "atom" subset) of the second atom that comprises this bond
bond_type	bond type. "C" = covalent, "H" = hydrogen

A special field named "position" is available for atom subsets. This field returns or sets a point at the "x","y","z" position (i.e. "position" is a shorthand for accessing "x", "y", and "z" individually).

4.3.11 Database record

Each subset in a VIEW database is comprised of *records*. Each record contains one or more *fields*. Each field has a string-valued *field name*. An alternate way to think of a subset is as a two-dimensional array. The records correspond to the rows of the array;

each column is labeled with a field name, and each entry in the array is an individual field within a record. A more detailed description of the records contained in VIEW molecular databases is given in the preceding section.

A database record is specified by:

RECORD ( )

Program constructs used for adding fields to a record are described in section 7.16.

4.3.12 Database field

A field is an individual value in a database record. The format for specification of an field is:

```
<record> . <field name>
```

EXAMPLES:

```
rec .position                 ! returns the "position" field from rec
dbase.atom(num).atom_type     ! returns the "atom_type" field from the
                              ! record with key "num" in the "atom"
                              ! subset of dbase
```

The field names that are available for molecular databases (produced using the *pdbtoview* program) are listed in section 4.3.10.

4.3.13 Attribute

An attribute contains some information about an object. Examples are the color of a displayable geometric object, the length of an array, or the name of a geometry group. Some (but not all) attributes may be set by placing them on the left hand side of an assignment statement. The format for specification of an object attribute is:

```
<variable> . <object attribute>
```

EXAMPLES:

```
item.color        ! item stores a displayable geometric object
list.length       ! where list stores an array, set, or group
item_col.red      ! where item_col stores a color
```

The attributes that are defined are listed below by object type. For each attribute, whether or not the attribute may appear on the left-hand side of an assignment is indicated (LHS), as well as the data type of the attribute. All attributes may be either in lower or upper case (but not mixed case).

	Attribute	Description	LHS	Type
	========	==========	===	====
Array				
	length	number of elements in the array	NO	numeric
Set				
	length	number of elements in the set	NO	numeric
Color				
	red	red component	YES	numeric
	green	green component	YES	numeric
	blue	blue component	YES	numeric
Displayable geometry (any type)				
	origin	geometric center of the object	YES	point
	group	geometry group this element is a member of	NO	group
	db	associated database	NO	database
	type	type of the geometric element possible values are: CYLINDER LINE SPHERE TEXT TRIANGLE	NO	string

database record pointer - described in section 7.16.13.

	Attribute	Description	LHS	Type
Sphere				
	center	center point of the sphere	YES	point
	radius	radius of the sphere	YES	numeric
	color	color of the sphere	YES	color
Line				
	vertex1	one endpoint of the line	YES	point
	vertex2	other endpoint of the line	YES	point
	color	color of the line	YES	color
Cylinder				

	vertex1	one endpoint of the cylinder	YES	point
	vertex2	other endpoint of the cylinder	YES	point
	radius	radius of the cylinder	YES	numeric
	color	color of the cylinder	YES	color
Text				
	color	color of the text	YES	color
Triangle				
	vertex1	one vertex of the triangle	YES	point
	vertex2	another vertex of the triangle	YES	point
	vertex3	another vertex of the triangle	YES	point
	normal1	normal at vertex1	YES	vector
	normal2	normal at vertex2	YES	vector
	normal3	normal at vertex3	YES	vector
	color	color of the triangle's front face	YES	color
	backcolor	color of the triangle's back face	YES	color
Point or *Vector*				
	x	x component of point or vector	YES	numeric
	y	y component of point or vector	YES	numeric
	z	z component of point or vector	YES	numeric
Geometry group				
	length	number of elements in the group	NO	numeric
	name	name of the group	YES	string
Database				
	name	name of the database	YES	string
Database record				
	db	associated database	NO	database
	key	record key (valid only if record is in a database)	NO	numeric or string

5. EXPRESSIONS

A number of language statements allow or require expressions (referred to by <expression>). An expression is one of three types: arithmetic expression, relational expression, or logical expression.

5.1 Arithmetic Expressions

An arithmetic expression consists of float or integer values, variables, arithmetic expressions, or arithmetic functions, either singly or combined using the operators:

+	(addition)
-	(subtraction or negation)
*	(multiplication)
/	(division)
**	(exponentiation)

Arithmetic expressions can be formed by combining simpler arithmetic expressions using any of the arithmetic operators or by enclosing an arithmetic expression in parentheses to indicate precedence.

EXAMPLES:

```
i + 5
((num + 5) / 4.3) ** 2.
SQRT(x*x + y*y + z*z)
4.74
MOD((num + 7, 5), 2)
(POINT(3. , 4. , 5. ) / length) * vector2
```

Operator precedence for all expression types follows that of C. The implementation of operator precedence is not very robust, however. I recommend that you use parentheses liberally.

For the purposes for future discussion, an arithmetic expression may be referred to by the datatype that it produces. Thus, an expression which always returns a numeric value will be referred to as a *numeric expression*, one which returns a vector as a *vector expression*, etc.

Operators may take point or vector valued arguments as follows (a <point expression> may be substituted for <vector expression> in all cases). The following table gives the argument types, the result types, and the operation name for all operators that apply to points or vectors.

<vector expression> + <vector expression>	-> <vector>	vector addition
<vector expression> - <vector expression>	-> <vector>	vector subtraction
<vector expression> * <vector expression>	-> <float>	dot product
<vector expression> * <numeric expression>	-> <vector>	vector scaling
<vector expression> / <numeric expression>	-> <vector>	vector scaling
- <vector expression>	-> <vector>	vector negation

Strings support a special use of the + operator – it indicates string concatenation. Thus the statement:

```
a = "/usr/tmp/" + file_name + ".dat";
```

would result in a having the value "/usr/tmp/myfile.dat" if the variable *a* contained the string "myfile".

5.2 Relational Expressions

A relational expression generates a Boolean value (**TRUE** or **FALSE**) and consists of two arithmetic expressions, logical expressions, or strings combined using the relational operators:

==	(equals)
!=	(not equals)
<	(less than)
>	(greater than)
<=	(less than or equal to)
>=	(greater than or equal to)

EXAMPLES:

```
((num - 5)*2) == 0
MOD(num1, 3) > num2
dbase.atom(a_num).atom_wqual != "CA"
```

5.3 Logical Expressions

A logical expression generates a Boolean value (TRUE or FALSE) and consists of a single relational expression or two Boolean values, variables, relational expressions or logical expressions combined using the relational operators:

OR	(logical inclusive or)
AND	(logical and)

or a single such expression preceded by the unary operator NOT.

EXAMPLES:

```
(dbase.atom(num).atom_wqual == "CA") OR
        (dbase.atom(num).atom_wqual == "C")
(num < 7) AND in_flag
(pos_flag == FALSE) AND (NOT out_flag)
x <= 5
```

6. FUNCTIONS

Functions are similar to those in C and FORTRAN. A function returns a single value. The following built-in numeric functions are available:

ROUND	– biased round to nearest integer (using C-library "rint" function)
SQRT	– square root
SIN	– sine
COS	– cosine
TAN	– tangent
ASIN	– arcsine
ACOS	– arccosine
ATAN	– arctangent
MOD	– modulus
DIST	– 3-D Euclidean distance

Each function (except MOD and DIST) takes a single argument in parenthesis which may be any numeric expression. The modulus (**MOD**) function takes two numeric arguments; the first is the number to be divided, and the second is the divisor. Both arguments are automatically rounded before use. The distance (**DIST**) function takes two point-valued arguments.

EXAMPLES:

```
SQRT (x*x + y*y + z*z)
MOD (num + 7, 5)
TAN (2*PI);
DIST (pnt1, pnt2);
```

All trigonometric functions take arguments in degrees; inverse trigonometric functions produce results in degrees.

The vector function, **CROSS**, computes the vector crossproduct of two vector-valued (or point-valued) arguments.

EXAMPLE: `CROSS(vec1, vec2)`

A logical function, **EXISTS**, checks for whether a variable or an array element has been defined. If the argument is defined, EXISTS returns TRUE, otherwise it returns FALSE;

EXAMPLES:

```
EXISTS(vec)
EXISTS(a[4])
EXISTS(b("key"))
```

EXISTS is only implemented for simple variables and arrays (indexed or keyed).

7. STATEMENT DESCRIPTIONS

7.1 Assignment

Variables are assigned using the syntax:

```
<variable> = <right hand side> ;
```

where <right hand side> is any <expression> or object creator:

EXAMPLES:

```
x2 = (item1.atom.x + item2.atom.x) / 2.;
tri.COLOR = element.BACKCOLOR;
tri = TRIANGLE(pnt1, pnt2, pnt3, color);
rad(num1) = atom_rad(num1) + 5.5;
pos[index] = dbase.atom(atom_num).position;
dbase.atom(num).x = 0.5;
is_true = a < 5;
```

The assignment operation causes the variable on the left to point to the right hand side; it does not create a copy. Another way of saying this is, the variable on the left becomes an *alias* for the right hand side.

Sometimes that is not what is desired. Consider this code sample:

```
tri1.COLOR = red;
tri2 = tri1;
tri2.COLOR = blue;
```

This code does not produce two triangles, a red one and a blue one (perhaps what is desired). tri2 and tri1 are the SAME triangle in this example and after the three statements have completed, that triangle is blue.

To allow for copying instead of aliasing, a special function, **COPY**, is available. **COPY** is permitted in place of any variable anywhere in the language (although it only makes sense as the right hand side of an assignment or a concatenation statement, or in the record argument to **DB_ADD_RECORD**). The single argument of **COPY** is the expression that is to be copied. **COPY** is only useful if its argument is a <variable> (if its argument is an expression, no aliasing is possible; an implicit copy is automatic).

The above example can be recoded as:

```
tri1.COLOR = red;
tri2 = COPY(tri1);
tri2.COLOR = blue;
```

resulting in two triangles, a red one and a blue one.

7.2 Concatenation

The concatenation operator allows you to add objects to container objects. Three container types may be concatenated to: arrays, sets, and geometry groups (the fourth container type, databases, has a special command, **DB_ADD_RECORD** for adding to it). The syntax is :

```
<container> &= <add_object> ;
```

where <container> is a <variable> of one of the mentioned container types, and <add_object> is a <variable>, expression, or function whose value is to be added to the container. Arrays must be indexed (this would not make sense for a keyed array, since no key is supplied); <add_object> will be placed in the location beyond the highest index that has been assigned prior to the concatenation. Concatenation is the only available method for adding an object to a geometry group. Note that concatenating an object to a set is functionally equivalent to unioning that object with the set.

Like assignment, concatenation aliases the object. Thus after execution, the statements:

```
obj.COLOR = COLOR(255,255,255);  ! white
grp &= obj;
obj.COLOR = COLOR(255,0,0);   ! red
```

will result in a single object in the group *grp*, colored red. Objects may be copied using the **COPY** function as part of the concatenation. The code:

```
obj.COLOR = COLOR(255,255,255);  ! white
grp &= COPY(obj);
obj.COLOR = COLOR(255,0,0);   ! red
```

Will result in two objects, one in *grp* colored red, and another (in whatever group it is in), colored white.

Displayable geometric objects may only belong to one group at a time. For this reason, concatenating an object to a group implicitly removes it from the group it was in.

7.3 Remove

The REMOVE statement is used to delete the definitions of objects. The syntax is:

```
REMOVE ( <remove variable 1> [ , <remove variable 2> , . . .
                    <remove variable n> ] );
```

each of the <remove variables> is a <simple variable> or an array element (indexed or keyed).

An object, once removed, is no longer available for use. If the object removed is a displayable geometric object, it will be deleted from the display panel. If the object is a geometry group, all objects within the group will be deleted from the display panel, and the group name will be removed from all **Group Operations** panels in the interface.

7.4 Exit

Execution may be terminated using the statement:

```
EXIT ;
```

7.5 If-else

The **IF - ELSE** statement is identical to the if-else statement in C. The syntax is:

```
IF ( <logical expression> ) <if-body> [ ELSE  <else-body> ]
```

where <if-body> and <else-body> have the same syntax as <iteration body> described under **Iteration** in the next section.

If the logical expression is **TRUE**, <if-body> is executed. If the logical expression is **FALSE**, and an **ELSE** clause is supplied, <else-body> is executed.

EXAMPLES:

```
IF (rad < 5. 0) DISPLAY sph1;
IF (element.atom.x < 5. 0)
{       count = count + 1;
        y = element.atom.y;
}
ELSE count = 0;
```

7.6 Iteration

There are several forms of iteration supported. **WHILE, DO-UNTIL,** and **FOR** are standard, C-like constructs. The **FOREACH** statement is an iterator that can be used to iterate on arrays, sets, databases, and geometry groups (any datatype that serves as a container, i.e., contains other objects, except records). The **LOOP** statement permits infinite iteration.

All iterations have an <iteration body> which consists of either a single statement, or a sequence of script language statements contained within a set of braces, {}. Thus, the iteration body is identical to the body of a "for" statement in C.

The examples given below for the **WHILE, DO-UNTIL,** and **FOR** statement are functionally equivalent.

7.6.1 While

The **WHILE** statement is used to perform a block of code while a specified condition is true. The syntax is:

WHILE (<logical expression> **)** <iteration body>

This statement executes the <iteration body> repeatedly as long as the <logical expression> evaluates to TRUE, and then stops. The evaluation precedes the execution of the <iteration body>.

EXAMPLE:

```
i = 0;
WHILE (i < 5)
{       REPORT("current index = ", i);
        i = i + 1;
}
```

This program segment yields the output:

```
current index = 0
current index = 1
current index = 2
current index = 3
current index = 4
```

7.6.2 Do-until

The **DO-UNTIL** statement is used to perform a block of code until a specified condition is true. The syntax is:

DO <iteration body> **UNTIL (** <logical expression> **) ;**

This statement executes the <iteration body> repeatedly until the <logical expression> evaluates to **TRUE**. The evaluation follows the execution of the <iteration body>. Note that the **DO-UNTIL** statement will always execute the iteration body at least once; the **WHILE** statement may not execute the iteration body at all (it will not if the <logical expression> is initially false).

EXAMPLE:

```
i = 0;
DO
{       REPORT("current index = ", i);
        i = i + 1;
} UNTIL (i >= 5);
```

7.6.3 For

This statement is identical to the C "for" statement (and similar to the FORTRAN "DO" statement). The syntax is:

```
FOR ( <initial statement> <logical expression> ; <increment statement> )
        <iteration body>
```

<initial statement> and <increment statement> may be any valid language statements. The terminating semicolon (;) is omitted for <increment statement>, however.

The **FOR** statement defines a loop structure. Prior to beginning the loop iteration, the <initial statement> is executed. On each iteration of the loop, the following sequence is performed: first the <logical expression> is evaluated and checked. If it evaluates to **FALSE**, the **FOR** statement terminates. Otherwise the <iteration body> is executed and then the <increment statement> is executed.

EXAMPLE:

```
FOR (i=0; i<5; i=i+1)
        REPORT("current index = ", i);
```

7.6.4 Foreach

The **FOREACH** statement is a special-purpose iterator that is used to iterate on container objects (arrays, groups, sets, databases). There are three forms of this statement:

1) For arrays, groups, and sets, the syntax is:

```
FOREACH ( <iteration variable> IN <container> [ <iteration phrases> ] )
    <iteration body>
```

<iteration variable> is a <variable name> that will take on the value of each of the objects in <container> during the iteration. <container> is a variable that references an array, a group, or a set. Iteration phrases are optional phrases that allow you to specify additional control of the iteration, or additional retrieval of information from each object in the container. The iteration phrase:

; INDEX = <variable name>

supported only for indexed arrays (the only one of these containers that can be accessed by index), will set the variable named by <variable name> to the index value of the <iteration variable> (the position in the array of <iteration variable>). Thus, in the iterator:

FOREACH (item IN list1; INDEX=n) { . . . body . . . }

the third time through the loop, *n* will have the value 2, and *item* will have the value stored at *list1*[2] (remember that array indexing begins at 0). The iteration phrase:

; KEY = <variable name>

supported only for keyed arrays (the only one of these containers that can be accessed by key), will set the variable named by <variable name> to the key value of the <iteration variable>. Note that items are retrieved in ascending key order.

2) To iterate on a database, the syntax is:

FOREACH (<iteration variable> **IN** <database> **.** <subset name>
[<iteration phrases>] **)** <iteration body>

<database> is a variable that contains a database and <subset name> is the name of the subset to be accessed. Each iteration will return a database record in <iteration variable>. Iteration phrases supported for this form include:

; START_KEY = <key value>

and

; END_KEY = <key value>

where <key value> is a key value (or a variable containing a key value) of a record in the specified data subset. The **START_KEY** phrase specifies the starting position in the database for the iteration; the **END_KEY** phrase specifies the ending position. Records are retrieved in the order they are stored on the file.

EXAMPLE:

```
FOREACH (rec IN dbase.atom; START_KEY=40; END_KEY=80)
    { . . . body . . . }
```

The database iterator also supports the **KEY** iteration phrase described above. The value produced by this phrase will be the record's key value.

3) The third form of the **FOREACH** command is used for iterating on geometry that is displayed on-screen. Any geometry that is toggled Off, will not be retrieved.

FOREACH ([<type specifier>] <iteration variable> **IN**
DISPLAY_GEOMETRY [**WITH DB**]) <iteration body>

This form of the **FOREACH** statement does not support any <iteration phrases>. The <type specifier>, which may be specified as either **OBJECT** or **GROUP**, determines whether each pass of the iteration will produce a single object or a group. The two constructs given below are functionally identical.

```
1) FOREACH (OBJECT obj IN DISPLAY_GEOMETRY) { . . . body . . . }

2) FOREACH (GROUP grp IN DISPLAY_GEOMETRY)
       { FOREACH (obj IN grp) { . . . body . . . } }
```

If no type specifier is supplied, it will default to **OBJECT**. The **WITH DB** phrase is used to restrict the iteration to those geometric objects that have a database associated with them (see section 7.16.13). The phrase is only applicable if the <type specifier> is **OBJECT** (either implicitly or explicitly).

7.6.5 Loop

The LOOP statement is used to loop forever. The syntax is simply:

LOOP <iteration body>

EXAMPLE:

```
LOOP
{       SELECT(item);
        REPORT(item.COLOR);
}
```

The LOOP statement will be terminated by exiting the tool, either explicitly using the Exit function in the interface, or implicitly by starting the execution of another tool.

7.7 Sleep

Tools may be forced to pause in their execution for a specified period of time using the **SLEEP** statement. The syntax is:

SLEEP(<time> **) ;**

Where <time> is a numeric expression that specifies the amount of time in seconds that the tool is to pause.

7.8 Report

The **REPORT** statement generates a written report containing one or more items. The report is produced in the window that the VIEW program is run from. The syntax is:

REPORT (<expression> [**,** <expression> . . .] **) ;**

EXAMPLES:

```
REPORT ("res name =", (item.atom.res_name), "rad = ", rad);
REPORT ("value =", ((x+5)/20));
```

7.9 User Query

Two statements are available that request information from the user, **ASK_NUMBER** and **ASK_STRING**.

7.9.1 Ask number

This statement produces a query that requests one or more numeric values from the user (see section 2.2.5 of the document *VIEW User Interface Description* for a discussion of queries and their use). The syntax is:

ASK_NUMBER (<title> **,** <simple variable 1> [**,** <simple variable 2> ...
, <simple variable n>] **);**

where <title> is a <string expression> to be printed at the top of the query, and <simple variable x> specifies a numeric value to be supplied by the user. Each <simple variable> must have a value defined prior to executing the **ASK_NUMBER**; these values will be displayed as initial values in the query.

EXAMPLE:

```
red = 0;  green = 0;  blue = 255;
ASK_NUMBER ("enter new color", red, green, blue);
```

7.9.2 Ask_string

This statement produces a query that requests one or more string values from the user The syntax is:

ASK_STRING (<title> , <simple variable 1> [, <simple variable 2> ...
, <simple variable n>]);

where <title> is a <string expression> to be printed at the top of the query, and <simple variable x> specifies a string value to be supplied by the user. Each <simple variable> must have a value defined prior to executing the **ASK_STRING**; these values will be displayed as initial values in the query.

EXAMPLE:

```
res_name = "GLY";
ASK_STRING ("enter a residue name", res_name);
```

7.10 Display

The **DISPLAY** statement ensures that geometric objects (either geometry groups or displayable geometric objects) are drawn in the display window. Its actions in doing so depend on the type of objects it is passed (in its parameter list) and the state of the system:

1) An argument that is a displayable geometry object is added to the *current display group*. The current display group is a group that has the same name as the tool being run. If no such group exists, the **DISPLAY** command will create one. If the group exists, but is turned Off, it will be toggled On.

2) An argument that is a group not in the interface (i.e. not listed in the **Group Operations** panels) is added to the interface and toggled On.

3) An argument that is a group in the interface that is turned Off is toggled On.

The syntax is:

DISPLAY (<variable 1> [, <variable 2> ... , <variable n>]) ;

EXAMPLES:

```
DISPLAY (obj1, obj2, obj3);
DISPLAY (grp);
```

7.11 Undisplay

The **UNDISPLAY** statement turns off groups in the display. It is functionally equivalent to toggling a group Off using the **Group Toggle** panel. The syntax is:

UNDISPLAY (<variable name 1> [**,** <variable name 2> ... **,** <variable name n>] **) ;**

EXAMPLE:

```
UNDISPLAY (grp1, grp2, grp3);
```

7.12 Select

The **SELECT** statement is used to specify picking of on-screen geometry. There are two forms of select:

SELECT (<object name> [**,** < select string>] [**; UNDOABLE**] **) ;**

Specifies that an object is to be picked on-screen and assigned to the variable name <object name>.

SELECT_DB (<object name> [**,** <select string>] [**; UNDOABLE**] **) ;**

Same as the previous, but the picked object must have a database associated. The statement will not return with a valid pick until an object with a database is hit.

The optional <select string> is a <string expression> to be displayed in a message box at the top of the screen when the system is waiting for the selection. If and only if this parameter is not supplied, the box will contain the string, "Select".

The **UNDOABLE** keyword specifies that a checkpoint is to be generated at this statement for the **Undo** system function. If an **Undo** button is pressed after this statement has completed, the system will backup execution to this statement and will pause waiting for a selection.

EXAMPLES:

```
SELECT(item1, "Select an atom");
SELECT_DB(item2;UNDOABLE);
```

7.13 Parameter definition

A single parameter definition statement may be supplied as the first statement of a script. This statement supplies information about parameters that are to be passed if the script is used as a subroutine. The syntax is:

PARAMETERS ([<parameter list> **]) ;**

where <parameter list> consists of one or more <parameter>s separated by commas.

<parameter> has the form:

```
<variable name>  <parameter type>
```

<parameter type> is one of the values:

IN	- the parameter is an input parameter
OUT	- the parameter is an output parameter
IN-OUT	- the parameter is both an input and an output parameter

EXAMPLES:

```
PARAMETERS (radius IN, distance OUT);
PARAMETERS (marker IN-OUT);
```

If the subroutine is to be called with no parameters, the parameter statement may be supplied with an empty parameter list, or may be omitted.

7.14 Subroutine call

The syntax of a subroutine call is identical to that for C or FORTRAN.

```
<subroutine name> ( [ <parameter 1> , <parameter 2> , . . . , <parameter n> ] ) ;
```

<parameter x> may be any variable name or expression. Note that in VIEW no sharp distinction is made between subroutines and main routines. Any tool (with or without a **PARAMETER** statement) may be invoked as a main routine by executing it (this usually only makes sense if the routine has no input parameters). Subroutines have no return values, all values to be returned must pass through the parameter list.

EXAMPLES:

```
Select_atom (obj1, pnt, sph);
Tetrahedron ( );
```

In the tool library supplied with the system, those routines that are used only as subroutines start with a capital letter; those that are directly executable as well do not. We suggest that you adopt this practice; it allows the **All tools** panel to list only those routines which are directly executable, using the default pattern ([a-z]*) .

7.15 Subroutine return

Return from a subroutine is specified using the statement:

```
RETURN ;
```

Unlike the return statement in "C", no argument may be specified (subroutines in the tool language have no return values).

7.16 Database manipulation

7.16.1 Defining subsets

Subsets are added to a database using the **DB_ADD_SUBSET** statement:

DB_ADD_SUBSET (<variable name> **.** <subset name>, <string> **) ;**

where <variable name> specifies a variable that contains a database, <subset name> is the name of the subset to be added, and <string> is a string expression that specifies the name of the key field for the subset.

EXAMPLE: DB_ADD_SUBSET (dbase.residue, "res_name");

7.16.2 Reading a database from a file

The **DB_READ** function is used to read a database from a file. The syntax is:

<variable> **= DB_READ (** <file_name> **) ;**

<file_name> is a string expression containing the full pathname of the file to be read. <variable> will be of type database when this statement completes.

EXAMPLE:

```
dbase = DB_READ ("/my/path/view/Database/molec");
```

A system-defined constant **DIRECTORY** is available to facilitate coding file names for DB_READ, DB_WRITE, GEO_READ, and GEO_WRITE. **DIRECTORY** is a string containing the path specified for the *-directory* parameter of the *iview* command (See *Running the VIEW System* for information on this parameter). The above example could be coded as:

```
dbase = DB_READ( DIRECTORY + "Database/molec");
```

Note use of the + string concatenation operator.

7.16.3 Writing a database to a file

The **DB_WRITE** function is used to write a database to a file. The syntax is:

DB_WRITE (<file name> **,** <database> **) ;**

<database> is a <variable> containing the database to be written. <file_name> is a string expression containing the full pathname of the file to be written. If a file by this name already exists, it will be overwritten.

EXAMPLE:

```
DB_WRITE("/my/path/view/Database/molec", dbase);
```

7.16.4 Retrieving database records

A language phrase that specifies record retrieval will be referred to a *record specifier*. A record specifier may specify retrieval from either a database or from a displayable geometric object. The latter allows you to retrieve database records that are "pointed to" by the object. The record pointer will have been added to the geometric object using the **DB_PTR** attribute described in section 7.16.13.

A database record may be retrieved from a database using the record specifier:

```
<database> . <subset name> ( <key> )
```

<database> is a variable that contains a database. <subset name> specifies the subset that the record is located in. <key> is a <numeric expression> (converted to integer by the system) or a <string expression> specifying the value of the database *key* (Note that the datatypes allowed for database keys are restricted from those allowed as keys for keyed arrays).

EXAMPLES:

```
dbase.atom(5)
db2.residue("LYS")
```

The second example shows retrieval from a user-defined subset called "residue".

There are two special-purpose database keys that may be used in this form of record specifier. **START** will return the first record in the subset. **END** will return the last record.

EXAMPLES:

```
dbase.atom(START)
db2.residue(END)
```

A database record may be retrieved from an displayable geometric object using the record specifier:

```
<geometry> . <subset name>
```

<geometry> is a <variable> that contains a displayable geometric object. <subset name> is the name of the subset that the record is located in.

EXAMPLES:

```
item1.atom
element[2].bond
```

Additionally, there are two functions that retrieve records in an ordered fashion from a database. The **NEXT_RECORD** and **PREV_RECORD** functions each take a record specifier as its single argument. **NEXT_RECORD** returns the record in the subset immediately following the argument, **PREV_RECORD** returns the previous record in the subset.

EXAMPLES:

```
new_rec = NEXT_RECORD(rec);
prev_rec = PREV_RECORD(dbase.atom(num));
rec = NEXT_RECORD(rec);
```

The last example shows the most common use of these functions, for iterating on a database. These functions may be initialized using special record keys **START** or **END**. These will retrieve the first and last record in a subset respectively.

EXAMPLE: `rec = dbase.atom(START);`

When either of these functions attempts to retrieve beyond the limits of the subset (NEXT_RECORD beyond the last record, PREV_RECORD prior to the first record), it will return a special value, EOD (end-of-data). This value may be checked for. For example, the code segment:

```
rec = dbase.atom(START);
WHILE (rec != EOD)
{       rec = NEXT_RECORD (rec);
        ... more code ...
}
```

Will completely iterate through the "atom" subset of the database stored in *dbase*. This code segment is functionally identical to:

```
FOREACH (rec IN dbase.atom)
{  ... more code ...  }
```

7.16.5 Retrieving record fields

Fields may be retrieved from database records using the syntax:

```
<record> . <field name>
```

where <record> is either a variable that contains a record or a record specifier as described in the previous section.

EXAMPLES:

```
rec.atom_num                    ! rec is a record)
item1.atom.atom_num             ! item1 is a displayable
                                !    geometric object
dbase.atom(5).atom_num          ! dbase is a database
```

7.16.6 Creating database records

Database records are created by using the **RECORD** creator (described in section 4.3.11) and assigning the record to a variable.

EXAMPLE: rec = RECORD ( );

7.16.7 Adding new fields to a record

Record fields are defined by assigning them using the syntax:

<record> . <field name> = <expression>

where <record> is either a variable that contains a record or a record specifier , <field name> is the name of the field that is being created, and <expression> is any expression of type numeric or string.

EXAMPLES:

```
rec.x = 5.0;             ! assigns the "x" field of rec.
rec.name = "res1";       ! assigns the "name" field of rec.
```

7.16.8 Replacing fields in a record

Record fields are replaced by reassigning them as described in the previous section.

7.16.9 Defining record keys

Record keys are set just like any other field; there is no special syntax for defining the key value. Thus if the key name is "name", the statement:

rec.name = "res1";

will define the key field to have the value "res1". The name of the key to be used in accessing the database is defined by the **DB_ADD_SUBSET** command (described in section 7.16.1). This command may be issued either before or after key fields are set for records to be inserted.

7.16.10 Adding records to a database

Records are added to a database using the **DB_ADD_RECORD** statement. The syntax is:

DB_ADD_RECORD (<database> **.** <subset name> **,** <record> **);**

where <database> is a <database expression>, <subset name> is the name of the subset to add to, and <record> is a <record expression>.

EXAMPLE: DB_ADD_RECORD(dbase.atom, rec);

Note that the record must contain the key field named in the **DB_ADD_SUBSET** statement that created the subset in order to insert it into that subset.

If a record already exists in the subset with the key value specified in <record>, it will be replaced. Otherwise, <record> will be appended to the end of the subset.

7.16.11 Adding new fields to a database

The first record that is added to an empty subset implicitly defines the fields for that subset. All records subsequently added to that subset must contain exactly the same fields (all of which must be present). It is possible to add a new field to all records in a subset, however, with the **DB_ADD_FIELD** statement. The syntax is:

DB_ADD_FIELD (<database> **.** <subset name> **,** <field name> **) ;**

where <database> is a <database expression>, <subset name> is the name of the subset to add to and <field name> is a <string expression> that contains the name of the field to be added.

EXAMPLE: DB_ADD_FIELD(dbase.atom, "conf");

The newly created field will contain a special value (9.9999E50) for all existing records, indicating that the value has not yet been assigned.

7.16.12 Deleting fields from a database

The **DB_REMOVE_FIELD** statement is used to remove a field from all records in a subset. The syntax is:

DB_REMOVE_FIELD (<database> **.** <subset name> **,** <field name> **) ;**

where <database> is a <database expression>, <subset name> is the name of the subset to be removed from and <field name> is a <string expression> that contains the name of the field to be removed.

EXAMPLE: DB_REMOVE_FIELD(dbase.atom, "conf");

7.16.13 Defining and using record pointers for displayable geometry

Displayable geometric objects may have one or more database record pointers associated with them. This allows you to retrieve database information based by querying an element of geometry (particularly useful when selecting). A record pointer is assigned to an object using the special attribute, "db_ptr". For example, the statement:

```
sph.db_ptr = rec;
```

where *sph* is a sphere and *rec* is a record from a database, would associate the record with the sphere.

These record pointers may be retrieved by specifying the subset that they point to (see section 4.3.10 for a discussion of subsets). Thus,

```
rec = sph.atom;
```

will place a database record in *rec* if sph has a record pointer that points to an "atom" subset. The syntax is identical to record access from a database, except that no key is specified. Note that although multiple record pointers may be assigned to an object, they must all be to the same dataset and only one record per subset may be retrieved. The pair of program statements:

```
sph.db_ptr = dbase.atom(num1);
sph.db_ptr = dbase.atom(num2);
```

where *dbase* contains a database are legal. However, the second pointer assigned will never be retrievable . The statements:

```
sph.db_ptr = dbase.atom(num1);
sph.db_ptr = dbase.bond(bnum);
```

will allow future reference of either an atom or a bond record.

The following example shows creation of a sphere representing the first atom in a database and assignment of a database pointer.

```
rec = dbase.atom(START);
sph = SPHERE (rec.position,0.5);
sph.db_ptr = rec;
DISPLAY (sph);
```

This next code segment provides for on-screen selection of this sphere, and prints its atom number.

```
SELECT (item);
REPORT ("atom number is -", item.atom.atom_num);
```

7.17 Geometry group manipulation

This section describes input-output operations for geometry groups. A number of transformations that may be applied to geometry groups as well as displayable geometric objects are described in section 7.18.

7.17.1 Reading geometry groups from a file

The **GEO_READ** function is used to read one or more geometry groups from a file. The syntax is:

GEO_READ (<file_name>) ;

<file_name> is a string expression containing the full path name of the file to be read.

EXAMPLE: GEO_READ("/my/path/view/Geometry/molec_geo");

The geometry groups within the specified file will automatically be added to all **Group Operations** panels and may be accessed with the tool using the **GROUP** statement.

7.17.2 Writing geometry groups to a file

The **GEO_WRITE** function is used to write one or more geometry groups to a file. The syntax is:

GEO_WRITE (<file_name> , <geometry group 1> [, <geometry group 2> , . . . , <geometry group n>) ;

Each <geometry group> is a <variable> containing the geometry group to be written. <file_name> is a string expression containing the full pathname of the file to be written. If a file by this name already exists, it will be overwritten.

EXAMPLE:

GEO_WRITE(DIRECTORY + "/Geometry/molec_geo", grp1, grp2);

where *grp1* and *grp2* are variables each containing a geometry group.

7.18 Geometric object manipulation

Displayable geometric objects and geometry groups may be manipulated using several transformation operations that are supplied. These are: rotation, translation, scaling and setting the origin.

VIEW uses two coordinate systems. The *world coordinate system* is defined by the values used to generate the geometry. World coordinates are usually based on data from a database; if the on-screen geometry represents molecular data, the world coordinate system will be the coordinate system of the molecular data and the units will be Angstroms. The world coordinate system is right-handed. The *screen coordinate system* is aligned with the display window (x is horizontal, y is vertical, z is out of the screen). The virtual trackball (described in *VIEW User Interface Description*) allows you to change the orientation, position, or size of the world coordinate system with respect to the screen coordinate system.

The transformation operations, **ROTATE, TRANSLATE** and **SCALE** perform rotation, translation, and scaling of geometry in the world coordinate system. Since the world coordinate system may not align with the screen, these transformations will, in general, be skew to the screen axes.

The transformation operations **ROTATE_SCREEN** and **TRANSLATE_SCREEN** are available for rotating, translating, and scaling geometric objects in the screen coordinate system (scaling is handled using **TRANSLATE_SCREEN** as described below). This coordinate system allows viewing rotations and translations to be specified in the plane of the screen (x-axis = horizontal, y-axis = vertical, z-axis = out of screen).

By specifying all on-screen geometry, these commands allow a tool to perform the same operations that the virtual trackball or the dials perform from the interface.

Applying transformation operations to a geometry group is functionally equivalent to iterating on the group and applying the operation to each object within it. For brevity in the discussion below, the term *geometric object* will be used when either *displayable geometric object* or *geometry group* might apply.

7.18.1 Rotation

The **ROTATE** command rotates a geometric object about its center point. The syntax is:

```
    ROTATE ( <geometric object> , <x> , <y> , <z> , <angle> ) ;
  | ROTATE ( <geometric object> , <vector expression> , <angle>) ;
```

These two forms are equivalent: <x>, <y>, <z> are <numeric expression>s that comprise <vector expression> in the second format. <angle> is a <numeric expression> that specifies the amount to rotate (in degrees) about the axis specified by <vector>).

EXAMPLES:

```
ROTATE (grp, 0.5, 0.5, 0, 20);
ROTATE (obj, vec, rot_angle);
```

Rotation is performed about the object's origin. By default, this is the point (0,0,0), although this may be changed by resetting the origin as described below.

Rotations are cumulative. Thus, applying a rotation of 20 degrees followed by a rotation of 30 degrees about the same axis, is the same as applying a single rotation of 50 degrees about that axis. Rotations about different axes also accumulate.

7.18.2 Translation

The **TRANSLATE** command moves a geometric object with no change of size or orientation. The syntax is:

```
      TRANSLATE ( <geometric object> , <x> , <y> , <z> ) ;
    | TRANSLATE ( <geometric object> , <vector expression> ) ;
```

The two forms are equivalent: <x>, <y>, <z> are <numeric expression>s that comprise <vector expression> in the second format. The vector specifies the amount to translate in the x, y, and z directions. The units of translation are the units that define the geometry. If the geometry being displayed represents a molecule, the units are probably Angstroms.

EXAMPLES:

```
TRANSLATE (grp, 0.5, 0.5, 0);
TRANSLATE (obj, vec);
```

Translations are cumulative. Thus, applying a translation of 2 units along the x-axis followed by a translation of 3 units along the same axis, is the same as applying a single translation of 5 units.

7.18.3 Scaling

The scale command changes the size of a geometric object. The syntax is:

```
      SCALE ( <geometric object> , <number> ) ;
    | SCALE ( <geometric object> , <x> , <y> , <z> ) ;
    | SCALE ( <geometric object> , <vector expression> ) ;
```

The first form is used for uniform scaling in all directions. <number> is a multiplicative scale factor. Thus a value of 0.5 indicates that the object is to be shrunk by 50%, a value of 2 indicates a doubling of the object's size.

The second and third forms are used for non-uniform scaling and are equivalent: <x>, <y>, <z> are <numeric expression>s that comprise <vector expression> in the second format. The vector specifies the amount to scale in the x, y, and z directions.

EXAMPLES:

```
SCALE (grp, 0.5);
SCALE (grp, 0.5, 0.5, 1);
SCALE (obj, vec);
```

Scaling is cumulative. Thus, applying a scale of 0.5 to an object (halving it) followed by another scale of 0.5 is the same as applying a single scale of 0.25 (0.5 * 0.5).

7.18.4 Screen rotation

The **ROTATE_SCREEN** command rotates a geometric object about its center with the rotation specified in screen space. The syntax is:

ROTATE_SCREEN (<geometric object>, <x>, <y>, <z>, <angle>) ;
| **ROTATE_SCREEN** (<geometric object>, <vector expression>, <angle>) ;

The arguments are identical to those for the **ROTATE** command.

An alternate form of the command substitutes the keyword **ALL** for <geometric object>. This form will rotate all geometry currently being displayed about the screen center.

EXAMPLES

```
ROTATE_SCREEN (obj, vec, 4 5.);
ROTATE_SCREEN (ALL, 0, 1, 0, 180.);
```

7.18.5 Screen translation

The **TRANSLATE_SCREEN** command moves all geometry in the plane of the screen or scales it (if "z" is specified).

TRANSLATE_SCREEN (<geometric object>, <x>, <y>, <z>) ;
| **TRANSLATE_SCREEN** (<geometric object>, <vector expression>) ;

The arguments are identical to those for the **TRANSLATE** command. The "z" component is used to scale the geometry.

An alternate form of the command substitutes the keyword **ALL** for <geometric object>. This form will translate (and/or scale) all geometry currently being displayed in the plane of the screen.

EXAMPLES

```
TRANSLATE_SCREEN (obj, vec);
TRANSLATE_SCREEN (ALL, 0, 1, 0);
```

7.18.6 Screen scaling

Screen scaling is performed using the "z" component of the **TRANSLATE_SCREEN** command described above.

7.17.7 Setting the origin

The origin of a geometric object may be changed by setting the **ORIGIN** attribute for that object.

```
        EXAMPLE:        obj.ORIGIN = POINT (0.5, 0.5, 0.5);
```

This origin (in the object coordinate system) becomes the center of rotation for the **ROTATE** or **ROTATE_SCREEN** command. It has no other effect or use.

7.18.8 Resetting geometric transformations

The **RESET_TRANSFORMATIONS** command cancels the effect of all previous geometric transformations (rotation, translation, or scaling) on an object or group. The format is:

RESET_TRANSFORMATIONS (<geometric object> **);**

This command has no effect on the origin setting for the object.

This command does NOT support the **ALL** keyword.

7.19 Set manipulation

The standard set operations: union, intersection, and difference are available.

7.19.1 Union

The + operator is used to specify set union. The original set is to the left of the +. To the right of the + may be either another set to be unioned, or a variable of any other datatype which will be added to the set.

EXAMPLES:

```
        new_set = set1 + set2;
        add_set = set1 + num;  ! add an element
```

7.19.2 Intersection

The * operator is used to specify set intersection. The original set is to the left of the *. To the right of the * may be either another set to be intersected, or a variable of any other datatype to be intersected.

EXAMPLES:

```
intersect_set = set1 * set2;
intersect_element = set1 * num;  ! intersect an element
```

Note that intersection may produce an empty set (the null set). This will be a set with a length of zero.

7.19.3 Difference

The - operator is used to specify set difference. The original set is to the left of the -. To the right of the - may be either another set to be removed, or a variable of any other datatype to be removed. Any elements in the set to be removed not contained in the original set are ignored.

EXAMPLES:

```
intersect_set = set1 - set2;
intersect_element = set1 - num;  ! remove an element
```

Difference may also produce the null set.

7.20 Events

7.20.1 Event definition

An event may be defined on either a key or a dial. An event consists of a header and a body. The header specifies the event name, the device that will activate the event, and whether the **UNDO** function is to backup to the beginning of this event or not. The body of the event is the code to be executed when the event is *triggered* (i.e., when the device specified in the header is activated). The syntax is:

EVENT (<event name> **; ON DIAL** <dial number> [**; UNDOABLE**])
        <event body>
| **EVENT** (<event name> **; ON KEY** <key> [**; UNDOABLE**]) <event body>

where <event name> is a string that names the event (must be unique for each event within a tool), <dial number> is an integer with values between 0 and 7 (inclusive) which selects a dial (see figure 1), <key> is a string that contains a single character naming the keyboard key that is to trigger the event, and <event body> is one or more tool language statements contained in a pair of braces, {}.

The optional **UNDOABLE** phrase specifies that a checkpoint is to be created just before the event is executed. Clicking on the **UNDO** button after the event is executed will cause the system to back up to just before the event was executed (assuming that the **UNDOABLE** keyword is not encountered within the event body), undoing the effects of the event.

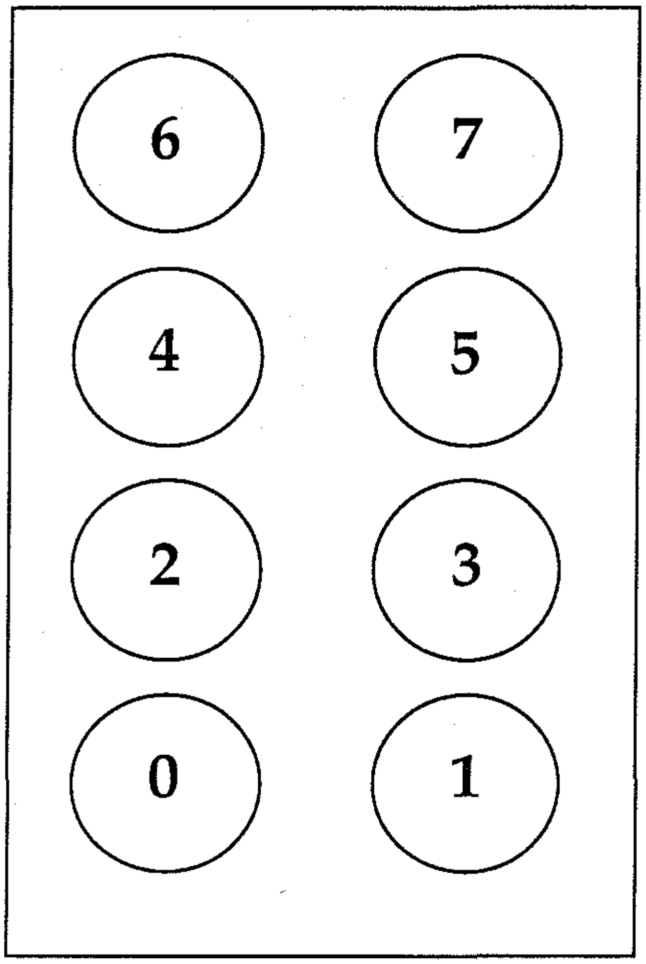


Figure 1
Dial assigments for interactive events

EXAMPLES:

```
EVENT ("change_rad"; ON KEY "r")
{       obj.RADIUS = 0. 5; }

EVENT ("rotate_item"; ON DIAL 7; UNDOABLE)
{       ROTATE (grp, vec, del_angle);
        ROTATE (grp2, vec, del_angle);
}
```

The events within a tool have access to all variables defined by that tool prior to the event's execution. Even after the tool has completed execution, these variables are still "alive" and available to the event.

An event may be triggered at any time, once the tool that defines it had begun executing. This may cause unexpected behavior. Consider the following language segment:

```
EVENT ("report_a"; ON KEY "a")
{       REPORT ("a = ",a); }

FOREACH (rec in dbase. atom)
{. . . code with no a references . . . }

a = 5;

. . . more code . . .
```

If the variable *a* is undefined prior to the **FOREACH** loop, and the "report_a" event is triggered while the loop is executing, an error will be generated (since *a* will be undefined). On the other hand, if "report_a" is triggered after the loop has completed and the assignment to *a* has been executed, then the program will print "a = 5". In this particular case, the **EXISTS** statement could be used to safeguard the event:

```
EVENT ("report_a"; ON KEY "a")
{       IF (EXISTS(a)) REPORT("a = ", a);
        ELSE REPORT ("cannot report a, not yet defined");
}
```

7.20.2 Control of events

Two statements are available for controlling the execution of events. An event may be deactivated using the statement:

STOP_EVENT (<event name>);

When an event has been stopped using this statement, the dial or key defined for this event will no longer respond by executing the event. Once stopped, an event may be reactivated using the statement:

START_EVENT (<event name>);

Events are automatically active when defined.

START_EVENT and **STOP_EVENT** may only be used to control events within the same tool; it is not possible to activate or deactivate events defined by other tools.

7.20.3 System constants for use in dial events

Several system-defined constants are available that provide information on dial status. **DIALNUM** contains the integer index number of the dial that was most recently turned. **DIALVAL** contains an integer that indicates dial position. **DIALVAL** is always between -50 and 50. When turning the dial clockwise, the numbers will increase until 50 is reached, and the value will then reset to zero. When turning counter-clockwise, the numbers will decrease until -50 is reached, and the value will then reset to zero. Fifty units correspond to about a one-quarter turn of the dial. **DIALRATE** contains an integer that indicates the speed of dial rotation. Values are approximately between one and five for slow rotations and between ten and thirty for fast ones. These constants may be used outside of events (this is most useful for **DIALNUM**).

VIEW

Interactive Tool Definition Language

Development Environment

Table of Contents

VIEW Interactive Tool Definition Language - Development Environment

Larry Bergman
1/25/93

1. INTRODUCTION

This document describes the development environment used to build, modify, and test drawing tools in the VIEW system. This document does not describe the tool language; the syntax and semantics of tool definitions are described in the document, *VIEW Interactive Tool Definition Language - Language Description.*

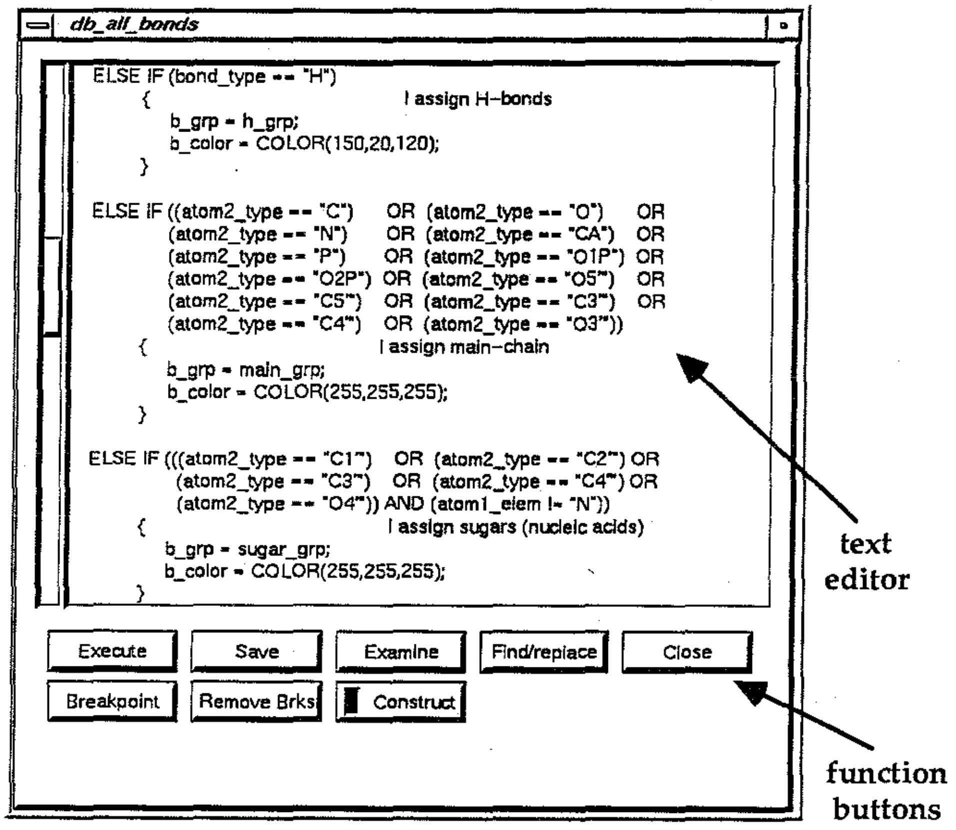
Users of the tool development environment are assumed to have experience with an interactive debugger (such as dbx), and be familiar with the use of breakpoints, stepping, and variable display. Experience with an screen-based editor (such as MacWrite or Word) will be very helpful.

Prior to using this document, you should have read *VIEW Exploratory Molecular Visualization System - Overview.* That document covers basic concepts of the VIEW system. This document will assume that you are familiar with these underlying ideas and definitions. Additionally, we assume that you are familiar with the use of the system interface, as described in *VIEW User Interface Description.*

2. TOOL CREATION

New tools may be created in two ways. The first is by using a debugger within VIEW. Debuggers are windows which contain a text editor within them. The editor allows you to alter a tool definition written in the tool definition language. Tools are usually produced by editing the text of some pre-existing tool. Our philosophy is, "New tools from old"; there is no mechanism for starting with an empty editor.

Tools may also be written or modified outside the VIEW system using a standard text editor such as vi or emacs. If you have much editing to do, we recommend this approach — the editor within VIEW lacks many of the features that make these editors easy to use. If you add a new tool to the library using an editor outside the VIEW system, you will need to update the **All tools** panel (either by reopening the panel or by entering a new pattern) in order to see the new tool name in that panel.



```
ELSE IF (bond_type == "H")
      {                                 ! assign H-bonds
         b_grp = h_grp;
         b_color = COLOR(150,20,120);
      }

ELSE IF ((atom2_type == "C")    OR (atom2_type == "O")    OR
         (atom2_type == "N")    OR (atom2_type == "CA")   OR
         (atom2_type == "P")    OR (atom2_type == "O1P")  OR
         (atom2_type == "O2P")  OR (atom2_type == "O5*")  OR
         (atom2_type == "C5*")  OR (atom2_type == "C3*")  OR
         (atom2_type == "C4*")  OR (atom2_type == "O3*"))
      {                                ! assign main-chain
         b_grp = main_grp;
         b_color = COLOR(255,255,255);
      }

ELSE IF (((atom2_type == "C1*")   OR (atom2_type == "C2*") OR
          (atom2_type == "C3*")   OR (atom2_type == "C4*") OR
          (atom2_type == "O4*")) AND (atom1_elem != "N"))
      {                                 ! assign sugars (nucleic acids)
         b_grp = sugar_grp;
         b_color = COLOR(255,255,255);
      }
```

Figure 1

A VIEW Debugger

2.1 Text Editors

An *editor* (figure 1) is a subpanel within a debugger in which tool description text is displayed. The text in an editor may be modified using a Macintosh-like interface.

If the text to be displayed is larger than the editor window, a scroll area will be displayed to the left of the text. This scroll area is used as described for file list scrolling in Section 2.2.4 of the document *VIEW User Interface Description.*

Several editing operations may be performed by using a combination of mouse and keyboard as indicated in the following table:

Desired Result	User Action	System Response
Positioning the text cursor	Position the mouse cursor at the place that the text cursor is to be positioned. Click the left mouse button.	A vertical blue cursor will appear in the text at the mouse cursor position (or at the start or end of the line if the cursor is outside the text).
Entering text	Position the text cursor. Type the new text with the mouse cursor inside the editor.	New text will appear at the text cursor position.
Selecting text	Position the mouse cursor at the place in the text that the selection is to start or end. Hold down the left mouse button and drag the mouse over the text.	Text will be highlighted in yellow.
Selecting single words	Position the mouse over the word. Click the left mouse button while depressing the alt key.	Selected word will appear in yellow.
Deleting selected text	Press the backspace or delete key. or	Highlighted text will disappear.
	Press the middle mouse button (to *cut* text)	Highlighted text will disappear, but be held in a paste buffer.

Desired Result	User Action	System Response
Replacing selected text	Type in replacement text. or	New text will replace highlighted text.
	Press the right mouse button (to *paste* text)	Contents of paste buffer will replace highlighted text.
Moving text	Select text to be moved	Text will be highlighted in yellow.
	Press the middle mouse button.	Highlighted text will disappear, but be held in a paste buffer.
	Position the text cursor at the insertion point. Press the right mouse button.	Previously removed text will appear at insertion point.

Certain keyboard keys have special functions within VIEW editors:

key	function
backspace	Deletes the character that precedes the text cursor
delete	Deletes the character that follows the text cursor
return	Moves text to the right of the cursor onto a new line. If there is no text to the right of the cursor, a blank line will be created.

The "tab" key has no effect in VIEW editors.

Selection may be made forward/downward in the text or backward/upward. If the cursor reaches the bottom (or the top) of the text window, the text will scroll to allow additional text to be highlighted. When as much text as you wish to select is highlighted, release the mouse. Selections are always contiguous areas of text. Starting a new selection will remove the definition of the previous selection (indicated by the previous selection dehighlighting).

A text selection may be deleted, cut, or replaced (by pasting over it or typing over it), as outlined above. The paste buffer can be modified only by cutting into it; each cut completely replaces the contents of the buffer. Cutting text from one window and pasting into another is supported. Note that the contents of the paste buffer may be used as many times as desired.

2.1.1 Find/replace

Find/replace operations may be invoked by clicking on the button labeled "Find/replace" below the editor text window. Clicking on this button causes a **Find/replace** panel (figure 2) to pop up on the screen. Within the **Find/replace** panel are two text areas, labeled "Find" and "Replace" and five function buttons.

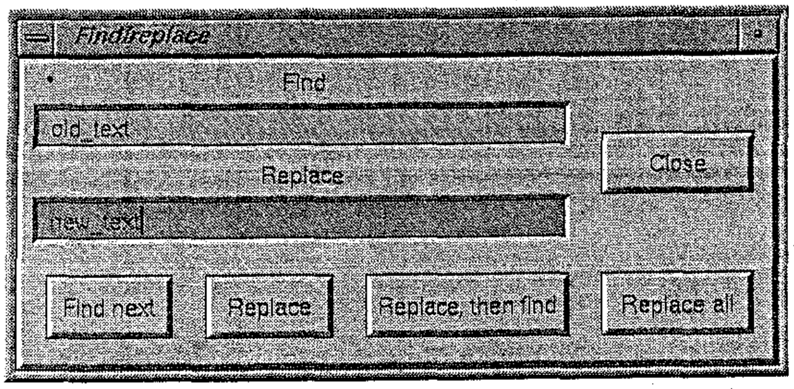


Figure 2
Find/replace panel

When the **Find/replace** panel appears, a text cursor (vertical blue bar) will be displayed in the **Find** area indicating that this area is ready for editing. Text within the **Find** and **Replace** areas may be edited in the same way as other text areas (see Section 2.2.3 of the document *VIEW User Interface Description* for more details). Either the "return" or "tab" key will move the text cursor to the other text area.

The use of each button in the **Find/replace** panel is described below.

2.1.1.1 Find next

Locates the next occurrence in the editor text (after the current text cursor position in the editor) of the text in the **Find** area. All text in routine being edited will be searched, not just the portion showing in the editor window.

If no cursor in the editor–	Locates the first occurrence of the text starting at the top and highlights the text in yellow.

If there is no occurrence of the specified text following the cursor, a confirmation panel will appear notifying you of this condition. Click on **OK** to make this panel disappear and to continue working.

2.1.1.2 Replace

Replaces the currently highlighted text in the editor with the text in the **Replace** area.

If no text is highlighted	– Replacement text will be inserted at the cursor position
If no cursor in the editor	– Replacement text will be inserted at the start of the text.

2.1.1.3 Replace, then find

This function behaves exactly the same as clicking on **Replace** followed by clicking on **Find**. It is useful for performing a series of replacements of a given string with visual confirmation of each replacement.

2.1.1.4 Replace all

This function will locate all strings in the editor text that match the string in the **Find** area and replace each with the string in the **Replace** area including the text following that in the window. If no matches are found, this function will have no effect. On completion, the last replaced string will be selected.

Note that **Replace all** will replace all matching entries in the routine being edited, not just those currently displayed in the editor window.

3. DEBUGGING

3.1 Types of Debuggers

Three types of debuggers are available in VIEW. *Examine, running,* and *error* debuggers.

Examine debuggers are used for viewing code. Running debuggers display code of executing tools and may be used for stepping through code. Error debuggers are produced when errors are detected in a tool.

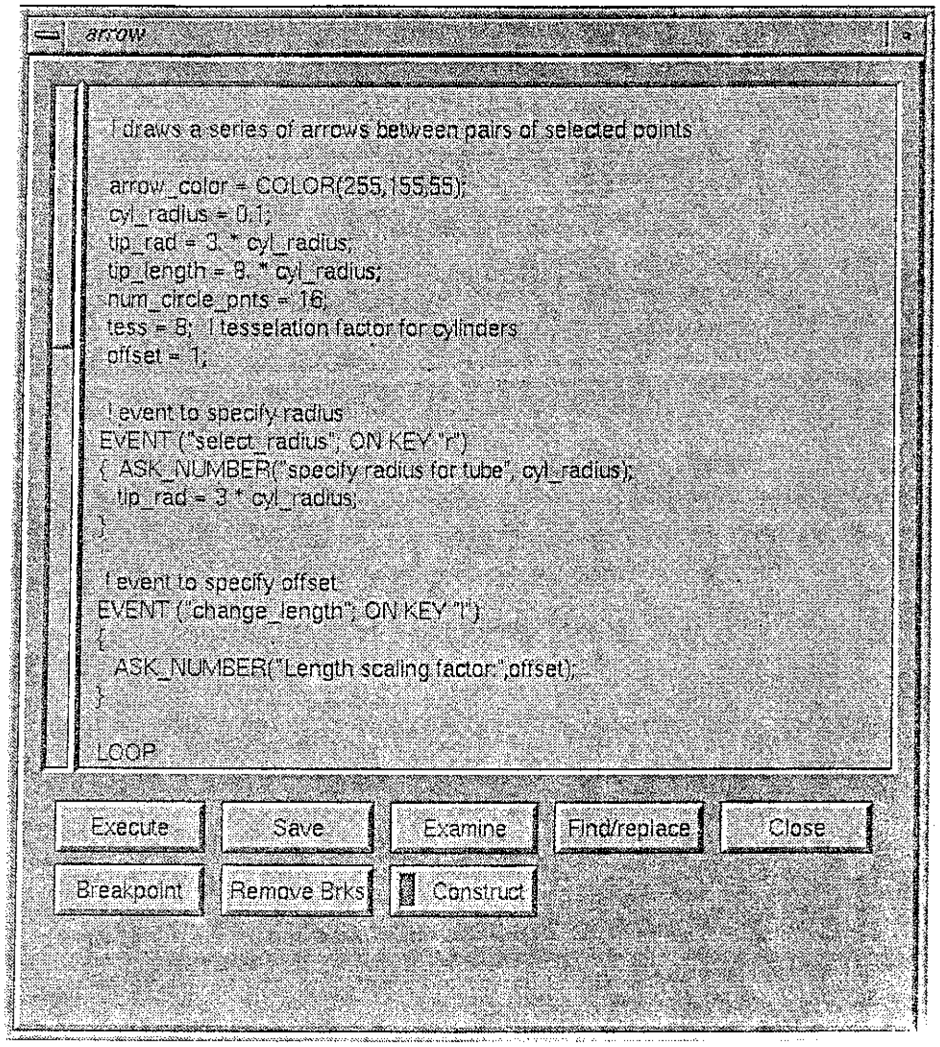
All debuggers have a top panel which contains a text editor. Each debugger has a set of function buttons below the editor. Running and error debuggers have additional subpanels (described with each below) beneath the function buttons.

The function buttons differ for the three editor types. The buttons for examine debuggers are common to all three types. Error debuggers contain additional buttons for displaying variable values. Running debuggers, in addition to these variable display buttons, contain buttons that control execution of the code when execution is paused. This information is summarized in the following table. X indicates that the function is not available for that type.

Information on which functions are available for each debugger type is summarized in the following table. An X indicates that the function is not available

	Examine Debugger	Running Debugger	Error Debugger
Execute			
Save			
Examine			
Find/replace			
Exit	X		X
Breakpoint			
Remove Brks			
Construct			
Display	X		
Undisplay	X		
Step	X		X
Next	X		X
Continue	X		X

Under certain conditions, buttons will be colored dark gray indicating that their function is not available. For example, a running debugger that is not at a breakpoint, will have disabled (gray) **Step**, **Next**, and **Continue** buttons, since these functions are only valid for a tool that is stopped at a breakpoint.



```
 ! draws a series of arrows between pairs of selected points

 arrow_color = COLOR(255,155,55);
 cyl_radius = 0.1;
 tip_rad = 3. * cyl_radius;
 tip_length = 8. * cyl_radius;
 num_circle_pnts = 16;
 tess = 8;  ! tesselation factor for cylinders
 offset = 1;

 ! event to specify radius
EVENT ("select_radius"; ON KEY "r")
{ ASK_NUMBER("specify radius for tube", cyl_radius);
  tip_rad = 3 * cyl_radius;
}

 ! event to specify offset
EVENT ("change_length"; ON KEY "l")
{
  ASK_NUMBER("Length scaling factor:",offset);
}

LOOP
```

Figure 3

Examine debugger

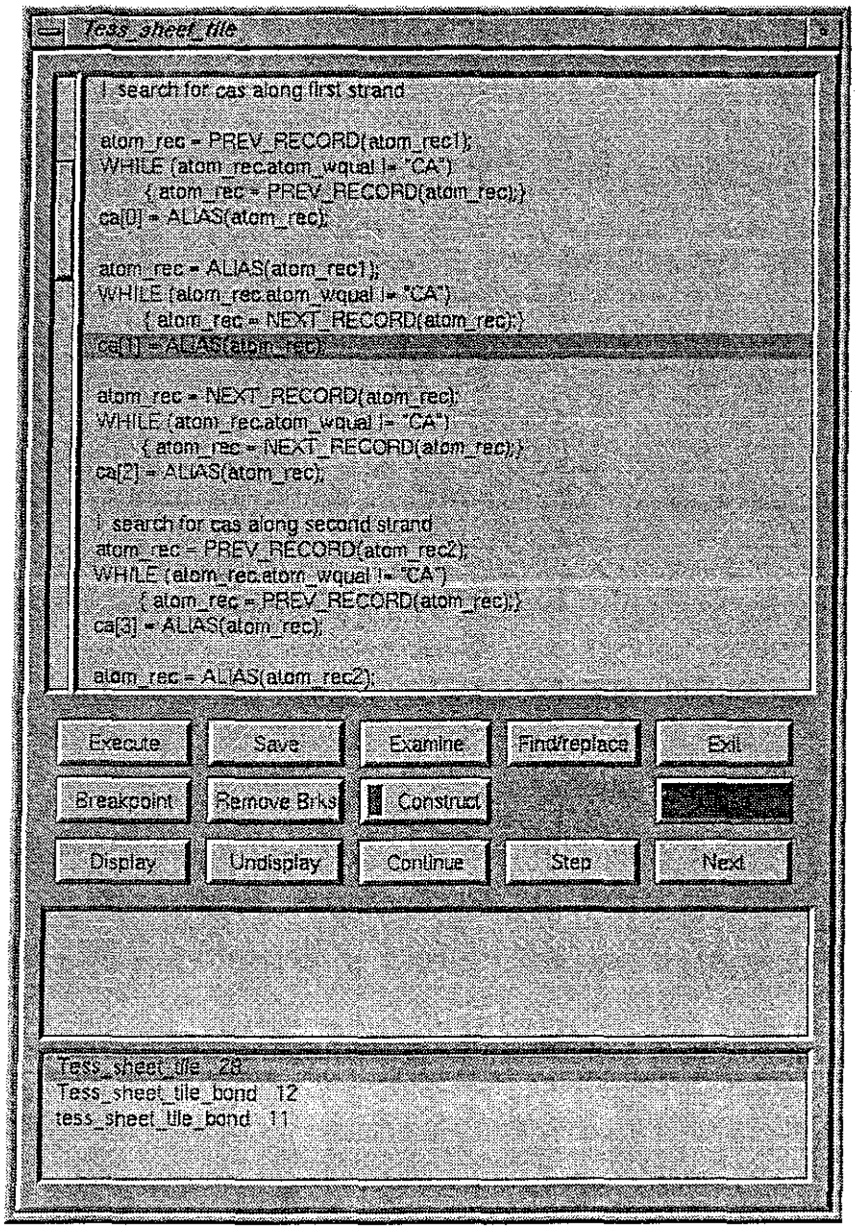
3.1.1 Examine Debugger

Examine debuggers (figure 3) are distinguished by gray panel backgrounds. An examine debugger can be invoked by selecting a tool in the **All tools** panel, and then clicking on **Examine** in that panel. An examine debugger containing text for a subroutine can be invoked by selecting the subroutine name within the text editor of a debugger (either by click-and-drag with the mouse, or by clicking over the name of the routine with the "alt" key depressed), and then clicking on **Examine** in that debugger. Examine debuggers are also produced when running debuggers are exited, either explicitly using **Exit** or implicitly by executing a different tool. In either case, all running debuggers will automatically change into examine debuggers. This information is summarized in the following table:

Type of debugger	Background color	Condition producing the debugger
Examine	Gray	Invoked from the **Examine** function in the **All tools** panel or
		Invoked from within another debugger by selecting the routine name within the editor and then clicking on **Examine** below the editor or
		Replaces running debugger when tool is exited (either with system **Exit** function, or by executing a new tool).

The buttons listed are found below the editor subpanel of an examine debugger. These same buttons are found in all debugger types. The button functions are described in more detail in section 3.2.

Button	Function
Execute	Executes this tool. If the code has been modified since the last **Execute** or **Save**, you will be prompted for a new tool name.
Save	Saves the tool. If the code has been modified since the last **Execute** or **Save**, you will be prompted for a new tool name.
Examine	Allows you to examine the text of subroutines as described above
Find/replace	Invokes a find/replace panel for locating text and making text substitutions
Close	Causes the debugger to disappear. All breakpoints are retained when the debugger is closed.
Breakpoint	Used for setting textual or graphical breakpoints (described in section 3.2.6).
Remove brks	Removes all currently defined breakpoints for this tool.
Construct	Activates or deactivates the "construction" debugging facility (described in section 3.2.8).



```
! search for cas along first strand

atom_rec = PREV_RECORD(atom_rec1);
WHILE (atom_rec.atom_wqual != "CA")
     { atom_rec = PREV_RECORD(atom_rec);}
ca[0] = ALIAS(atom_rec);

atom_rec = ALIAS(atom_rec1);
WHILE (atom_rec.atom_wqual != "CA")
     { atom_rec = NEXT_RECORD(atom_rec);}
ca[1] = ALIAS(atom_rec);

atom_rec = NEXT_RECORD(atom_rec);
WHILE (atom_rec.atom_wqual != "CA")
     { atom_rec = NEXT_RECORD(atom_rec);}
ca[2] = ALIAS(atom_rec);

! search for cas along second strand
atom_rec = PREV_RECORD(atom_rec2);
WHILE (atom_rec.atom_wqual != "CA")
     { atom_rec = PREV_RECORD(atom_rec);}
ca[3] = ALIAS(atom_rec);

atom_rec = ALIAS(atom_rec2);
```

```
Tess_sheet_tile  28
Tess_sheet_tile_bond  12
tess_sheet_tile_bond  11
```

Figure 4

Running Debugger

3.1.2 Running Debugger

Running debuggers (figure 4) have tan backgrounds. A running debugger is produced when syntactically correct code is executed from either an examine debugger or an error debugger. A running debugger will also pop up when either a breakpoint is encountered in a routine which does not already have a running debugger, or when you **Step** into a subroutine from a running debugger. This information is summarized in the following table:

Type of debugger	Background color	Condition producing the debugger
Running	Tan	Invoked when syntactically correct code is executed from any debugger. or
		Invoked when a breakpoint is encountered while executing a tool (see section 3.2.6 for a description of how to set breakpoints). or
		Invoked when a subroutine is stepped into using the **Step** function (described in section 3.2.11).

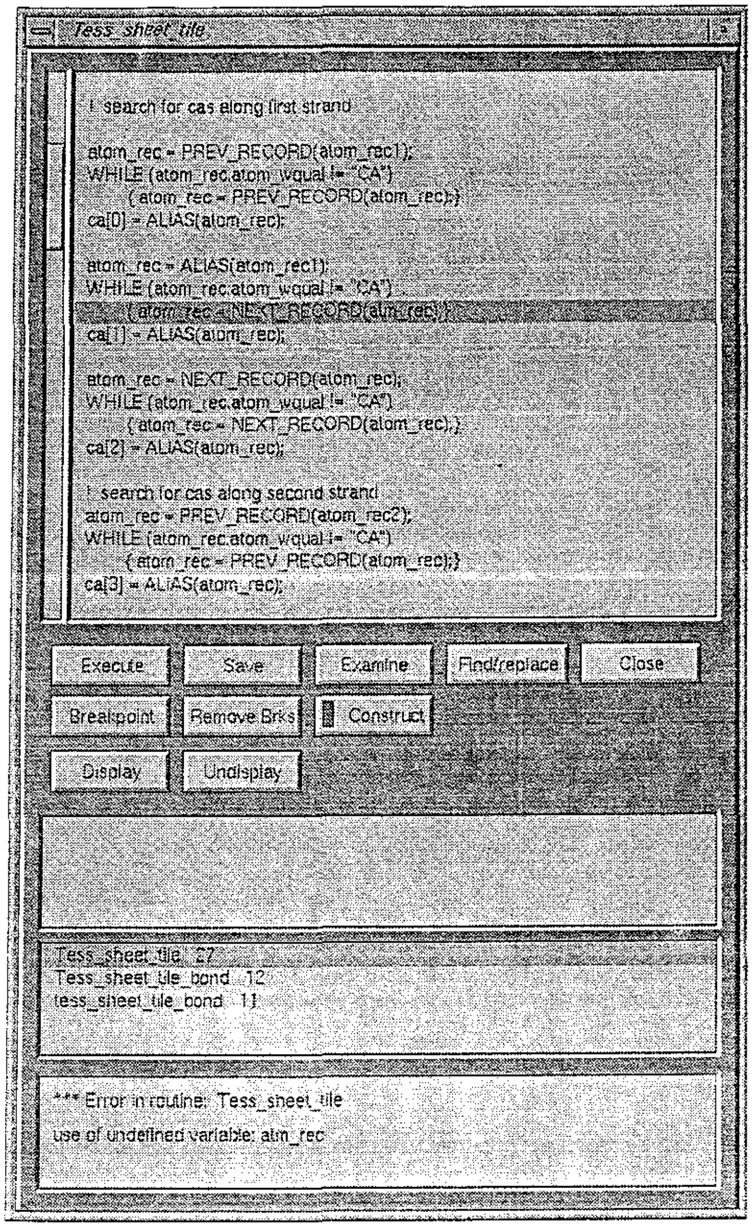
A running debugger may automatically convert to one of the other types as follows. If an error is encountered in the routine, the debugger will automatically become an error debugger. If the tool is exited, either explicitly when you click on **Exit**, or implicitly by starting execution of another tool, the running debugger will convert to an examine debugger.

In addition to the buttons found in an examine debugger, Running debuggers contain:

Button	Function
Display	Allows you to examine variables by seeing printed values and, if possible, by seeing a representation in the display panel.
Undisplay	Causes the color of any object in the display panel altered by **Display** to revert to its original color.
Step	Causes the code to advance forward a single statement. If the current statement is a subroutine call, the subroutine will be entered; its code will be displayed within a new running debugger stopped at the first statement.
Next	Causes the code to advance forward a single block. Subroutines called within the current block will be executed entirely with no pause for debugging.
Continue	Causes the tool to continue execution. Used when the tool is stopped for debugging.

In addition to the text editor subpanel, running debuggers have the following subpanels below the function buttons

Position	Name	Function
Top	Variable Display	Displays values of variables when the Display function is selected.
Bottom	Call Stack	Displays the sequence of function calls, with the last-called function at the top. Each function call has the current line number to the right of it. Clicking on any of the routine names in the call stack area will cause that function to highlight in blue and its text to appear in the call area



```
! search for cas along first strand

atom_rec = PREV_RECORD(atom_rec1);
WHILE (atom_rec.atom_wqual != "CA")
      { atom_rec = PREV_RECORD(atom_rec);}
ca[0] = ALIAS(atom_rec);

atom_rec = ALIAS(atom_rec1);
WHILE (atom_rec.atom_wqual != "CA")
      { atom_rec = NEXT_RECORD(atm_rec);}
ca[1] = ALIAS(atom_rec);

atom_rec = NEXT_RECORD(atom_rec);
WHILE (atom_rec.atom_wqual != "CA")
      { atom_rec = NEXT_RECORD(atom_rec);}
ca[2] = ALIAS(atom_rec);

! search for cas along second strand
atom_rec = PREV_RECORD(atom_rec2);
WHILE (atom_rec.atom_wqual != "CA")
      { atom_rec = PREV_RECORD(atom_rec);}
ca[3] = ALIAS(atom_rec);
```

```
Tess_sheet_tile  27
Tess_sheet_tile_bond  12
tess_sheet_tile_bond  11
```

```
*** Error in routine:  Tess_sheet_tile
use of undefined variable: atm_rec
```

Figure 5

Error debugger

3.1.3 Error Debuggers

An error debugger has a red background (figure 5). An error debugger is produced when an error is encountered in a tool. The debugger will pop-up with the line where the error occurred (or where the system thinks the error occurred) highlighted in blue in the text editor. This information is summarized in the following table:

Type of debugger	Background color	Condition producing the debugger
Error	Red	Invoked when an error is encountered while executing a tool. Errors may be encountered while scanning the text (syntax errors) or while executing it (runtime errors).

Errors may be of two types. Syntax errors result from illegal use of the tool definition language. Run-time errors are produced when legal statements are used improperly.

In addition to the buttons found in an examine debugger, Running debuggers contain:

Button	Function
Display	Allows you to examine variables by seeing printed values and, if possible, by seeing a representation in the display panel (see Section 3.2.9 for more detail).
Undisplay	Causes the color of any object in the display panel altered by Display to revert to its original color.

An error debugger produced by a syntax error will have an inoperative **Display** function; clicking on **Display** will produce no results in the **Variable Display** subpanel.

Error debuggers have three subpanels below the function buttons. The top two are identical to those in a running debugger. The bottom-most has the following function:

Position	Name	Function
Bottom	Error Message	Describes the error condition encountered.

For certain kinds of error conditions, the system is unable to report accurately. For example, an unmatched set of braces anywhere in the program is likely to be reported with the last line of the program highlighted and the message: "null character". Occasionally, an error will be accompanied by a message in the window that VIEW is run from giving additional information. Whenever you are uncertain about an error, it is always worth checking this window.

3.2 Debugger functions

The following sections describe each of the debugger functions.

3.2.1 Execute

Clicking on **Execute** will execute the tool specified by the text in the text area, or the main routine, if the text is a subroutine not at the top of the call stack. If the text has not been modified, the tool will execute immediately. If you have modified the text (without saving it), the system will save the routine you are currently editing prior to execution. It will do this by requesting a name for saving the new text as described in the next section. Canceling the save will also cancel the execute operation. This save operation will delete all currently defined breakpoints prior to executing the tool.

Note that normal operating procedure from within an error debugger is to change the code and attempt it again, by selecting **Execute**.

3.2.2 Save

If the tool text has been modified, this function will save the modified tool in the library.

Desired Result	User Action	System Response
Saving a tool	Click on **Save** button	Requests a name for text to be saved.
	Enter the new name for the tool (or leave the name unchanged to overwrite the current tool definition) The default is the original name of the tool whose code you are modifying.	
Confirming intention to save or	Press "return" or click on **OK** button.	Saves the text.
Canceling request to save text	Click on **Cancel** button.	Cancels the save operation.

3.2.3 Examine

Displays a subroutine called from the routine currently being examined.

Desired Result	User Action	System Response
Viewing the text of a subroutine	Drag the mouse over the subroutine name in the text editor, or While depressing the "alt" key, click the left mouse button with the mouse cursor positioned over the subroutine name.	Highlights the name in yellow
	Click on **Examine** button.	Pops up new examine debugger containing the subroutine's text

3.2.4 Find/replace

This function is used for editing text in the text window. The Find/replace function is described in section 2.1.

3.2.5 Exit

This function terminates the execution of the currently active tool or the currently active event. It has no effect on which events are available for activation using keys or dials.

3.2.6 Breakpoint

This function allows you to define or remove breakpoints. There are two types of breakpoints: textual breakpoints and graphical breakpoints.

Textual breakpoints allow you to select lines in the text where the system is to pause execution, allowing you to examine variable values (using **Display**) and/or manually control the execution (using **Step** or **Next**). These are the "standard" breakpoints supported by interactive debuggers such as UNIX's dbx. Textual breakpoints are set in the text in the editor window. When an executing tool reaches a statement with a breakpoint set, a running debugger is popped up (if one is not already displayed), and the breakpoint statement is highlighted in aqua indicating where execution is paused.

The following table describes how to set and remove breakpoints in the text.

Desired Result	User Action	System Response
Set textual breakpoint	Click on **Breakpoint** button.	Changes the mouse cursor shape to a stop sign.
	Click on text line in editor where breakpoint is to be set.	Highlights the selected text line in purple. Changes the mouse cursor back to an arrow.
Remove textual breakpoint	Click on **Breakpoint** button.	Changes the mouse cursor shape to a stop sign.
	Click on text line in editor where breakpoint is to be removed.	Removes the purple highlight from the selected line. Changes the mouse cursor back to an arrow.

If you set or remove a breakpoint at a statement that is highlighted in aqua (part of the current block), you will see no visual effect; the aqua highlight has priority over the purple highlight.

A *graphical breakpoint* allows you to specify a pause in tool execution based on creation of particular objects. You specify a graphical breakpoint by selecting an object created by a drawing tool in the display panel. When the tool is rerun, and that object is about to be displayed, the tool will be presented in a running debugger with the execution paused on the appropriate text line (just like a textual breakpoint). Once any graphical breakpoints have been set for a tool, reexecution of that tool will cause all objects created by previous executions of the tool to be deleted from the display window, except for objects that have graphical breakpoints specified.

Graphical breakpoints are set and removed as follows:

Desired Result	User Action	System Response
Set graphical breakpoint	Click on **Breakpoint** button.	Changes the mouse cursor shape to a stop sign.
	Click on object in the display panel for which a breakpoint is to be set.	Changes the color of the selected object.
Remove graphical breakpoint	Click on **Breakpoint** button.	Changes the mouse cursor shape to a stop sign.
	Click on object in the display panel for which a breakpoint is to be removed.	Restores the original color of the selected object

3.2.7 Remove Brks

This function removes all breakpoints, both textual and graphical.

3.2.8 Construct

This button turns display of *construction objects* on and off. Construction objects are geometrically representable objects created during tool execution – points and vectors. Points are represented by small white spheres, vectors by thin white cylinders. These objects only affect display; they do not affect the tool execution.

For example, suppose the system executes the fragment of tool code given below. Each program statement is followed by a line number for ease of reference.

```
pnt1 = POINT(1,1,1);          !  (1)
pnt2 = POINT(0,0,0);          !  (2)
pnt3 = POINT(0,1,0);          !  (3)
vec1 = VECTOR(pnt1,pnt2);     !  (4)
vec2 = VECTOR(pnt3,pnt2);     !  (5)
xvec = CROSS(vec1,vec2);      !  (6)
```

If the construction function is turned on, execution of statements (1), (2) and (3) will result in small spheres being displayed at positions (1,1,1), (0,0,0), and (0,1,0). Execution of statements (4) and (5) will result in the display of thin cylinders between the specified points. Statement (6) is a cross-product; the resulting vector will display as a thin cylinder.

The objects created when construction is on are placed in a special geometry group called "debug". This group is like any other geometry group, and may be toggled, removed, renamed, or written to a file.

If the rectangle on the **Construct** button is yellow, construction objects will be created during tool execution; if it is gray, construction is turned off and no construction objects will be generated. Clicking on this button will toggle back and forth between the two states. With construction On, construction objects will be displayed even if the debugger is closed.

Automatic creation of construction objects is particularly useful if combined with debugger functions such as **Step** or **Next,** although such combination is not required.

3.2.9 Display

The **Display** function combines the print function found in traditional debuggers with a graphical display function for geometric entities. Names of variables to be displayed are selected in the text window (by dragging with the mouse or by using alt/left_mouse word selection). Only simple variable names may be displayed (there's currently no support for displaying array elements, record fields, or attributes).

If the variable is *printable* (see table below), clicking the Display button with the mouse will result in the variable's value being printed in the variable display area of the debugger. If the variable is *displayable,* a geometric element will be added to the display panel, or if one already exists, its color will change to either white or red. These new geometric elements (spheres for points, cylinders for vectors) are placed in a special group called "debug" (which also contains construction objects, as described in section 3.2.8).

Variable type	Printable	Displayable
Number	YES	NO
Boolean	YES	NO
String	YES	NO
Point	YES	YES
Vector	YES	YES
Displayable geometric object	NO	YES
Color	YES	NO
Array	NO	NO
Set	NO	NO
Group	NO	NO
Database	NO	NO
Record	NO	NO

3.2.10 Undisplay

This function causes any geometric element whose color has been changed by the **Display** function to revert to its original color.

3.2.11 Step

This is the standard "step" function provided by most debuggers. It is usable only when the debugger is stopped at a statement. Clicking on **Step** will cause the execution (and highlighted text) to proceed to the next statement. If the next statement is on the same line as the current statement, the highlighting will not advance. If the current statement is a subroutine call, clicking on **Step** will cause a debugger to pop up for that routine with the first statement of the routine highlighted. Debugging (using **Step**, **Next**, or **Continue**) will then be active within the subroutine's debugger. When the last statement of the subroutine is reached, clicking on **Step** (or **Next**) will pop up and activate the calling routine's debugger.

Note that statement highlighting includes the complete body of a statement. Thus, if the debugger is stopped at a **FOR** statement, the complete statement including the loop body will be highlighted.

3.2.12 Next

The **Next** function operates somewhat differently than that found in most debuggers. Clicking on **Next** will cause the debugger to execute the currently highlighted statement without stopping within it. Thus, if the current statement is a **FOR** loop, clicking on **Next** will cause completion of the body of the loop with no pause for debugging. The next statement to highlight will be the next statement to be executed after the loop completes. If the highlighted statement when **Next** is clicked contains a subroutine call, the text of the subroutine will not pop up although the subroutine will be executed.

3.2.13 Continue

Continue causes execution of the tool to resume. No further debugging will be possible until a breakpoint (textual or graphical) is encountered.

VIEW

Data File Formats

Table of Contents

VIEW Data File Formats

Larry Bergman
1/26/93

1. INTRODUCTION

This document describes the format of two file types used by the VIEW system – database files (which is the format used for storing molecular data), and geometry files. Each format will be described by first presenting an annotated sample file, and then by giving a symbolic description of the format.

The sample files are annotated here by including bold-face numbers within parenthesis. These are NOT part of the file; they are merely pointers to comments that follow the sample file.

The blank lines in the sample files have been inserted are for clarity. VIEW ignores all blank lines in both database and geometry files. VIEW database files are free-format. You may include as much white space (blanks, tabs, carriage-returns) as you wish between fields.

The syntax used to describe the format follows that used in *VIEW Interactive Tool Definition Language – Language Description*. A portion of the file that will be described elsewhere (called a *non-terminal* in computer-science jargon) is enclosed in angle brackets, <>. For example, <header>. Some portions of a file are optional. These are enclosed in square brackets, []. For example, [<subset 2>]. All text within the file format description that is not enclosed in angle brackets or square brackets is to be entered in the file exactly as shown.

2. DATABASE FILE FORMAT

2.1 Sample database file

```
mhr_h1              (1)

atom                (2)
20                  (3)
atom_num            (4)
i    atom_num       (5)
s    atom_type
```

```
s    atom_element
s    atom_wqual
s    alt_conf
s    res_name
s    res_chain_id
i    res_num
f    x
f    y
f    z
f    occupancy
f    temp_fact
s    H_bond_donor
s    conformation
s    atom_structure
i    bond_num1
i    bond_num2
i    bond_num3
i    bond_num4

1 A N  N   A GLU ? 19  14.4  39.0 23.1 1.0  20.77  N  ? C 1  0  0  0        (6)
2 A C  CA  A GLU ? 19  14.0  40.1 23.9 1.0  23.23  N  ? C 1  2  3  0
3 A C  C   A GLU ? 19  14.9  41.2 23.9 1.0  22.88  N  ? C 2  4  0  0
4 A O  O   A GLU ? 19  14.5  42.4 23.6 1.0  23.59  N  ? C 4  0  0  0
5 A C  CB  A GLU ? 19  13.5  39.8 25.3 1.0  28.41  N  ? C 3  5  0  0
6 A C  CG  A GLU ? 19  12.5  40.9 25.8 1.0  38.85  N  ? C 5  0  0  0

END_SUBSET            (7)

bond                  (8)
4                     (9)
bond_num
i    bond_num
i    atom_num1
i    atom_num2
s    bond_type

1    1   2     C      (10)
2    2   3     C
3    2   5     C
4    3   4     C
5    5   6     C

END_SUBSET            (11)
```

(1) "mhr_h1" is the name that will assigned to the database once it is read into

VIEW.

(2) "atom" is the name of the first subset in this database.

(3) The atom subset has 20 fields per record.

(4) "atom_num" is the name of the key field for the atom subset

(5) The first header record for the atom subset. There is one header record per field. Each record has a type designator (i = integer, f = float, s = string) followed by a field name.

(6) The first data record for the atom subset. The record has 20 fields each of the type indicated by the header.

(7) The END_SUBSET record terminates the atom subset.

(8) "bond" is the name of the next subset in this database.

(9) The bond subset has 4 fields per record

(10) The first data record for the bond subset.

(11) The end of the bond subset.

2.2 File format description

The structure of a database file is:

```
<database name>

  <subset 1>
 [ <subset 2> ]
      .
      .
      .
 [ <subset n> ]
```

<database name> is a string 80 characters long or less that defines the internal name to be used when the database is read.

The format of each <subset> is:

```
<header>

<record 1>
```

```
[ <record 2> ]
      .
      .
      .
[ <record m> ]

END_SUBSET
```

The <header> is as follows:

```
            <subset name>

      <number of fields per record>

              <key name>

          <field 1 definition>
        [ <field 2 definition> ]
                   .
                   .
                   .
        [ <field x definition> ]
```

<subset name> is a string (up to 80 characters in length) that will be used to reference this subset.

<number of fields per record> is an integer which specifies how many fields are to be contained in each record, and how many <field definition> records will follow.

The <key name> is the name of the field that will be used for key accesses to the data records. There MUST be a <field definition> record that has a <field name> matching <key name>. The <type code> for this record must be integer (i).

Each <field definition record> has the following format:

```
<type code>    <field name>
```

<type code> indicates the data type of the field. It has one of the following values:

f – float
i – integer
s – string

<field name> is a string that names the field.

The <record>s each contain <number of fields per record> fields. Each is of the type indicated in the corresponding <field definition>.

3. GEOMETRY FILE FORMAT

3.1 Sample geometry file

```
z                                               (1)
helix_sticks                                    (2)
1                                               (3)
/glycine/grip13/view/data/Database/2mhr_h1h2.dat      (4)

l                                               (5)
1                                               (6)
0 0 2                                           (7)
0 255 255 255                                   (8)
15.26     37.89     21.15                       (9)
14.63     39.03     22.53
l                                               (10)
1
0 0 14
0 255 255 255
14.635    39.034    22.5305
14.002    40.175    23.908

z                                               (11)
atom_positions
2
/glycine/grip13/view/data/Database/2mhr.dat
/glycine/grip13/view/data/Database/2mhr_h1h2.dat
s                                               (12)
1
0 0 432
0 255 0 0
15.632    47.806    17.783
0.4
5

c                                               (13)
0
0 255 200 170
26.835    51.326    18.373
26.2465   50.3795   19.9195
0.15
8
1 0 0 0

t                                               (14)
```

```
0
0 225 225 225
1
200 20 150
29.58     56.66     27.44     -0.1926     0.9806  0.03534
11.30     52.97     30.31     -0.1926     0.9806  0.03534
8.364     53.06     11.67     -0.1926     0.9806  0.03534

x                                         (15)
0
0 255 255 255
LYS
27.135    51.626    18.373
```

(1) Each geometry file contains one or more geometry groups. Each geometry group starts with a record containing the single character, "z".

(2) "helix_sticks" is the name of the first geometry group.

(3) This group has pointers to one database in its geometry.

(4) The single database file for this group.

(5) Start of the first geometric primitive for this group. "l" indicates that this is a line.

(6) This primitive contains one database pointer.

(7) The database pointer for this primitive. The pointer is to the first database (index 0), the first subset in the database (index 0 which is "atom" for a molecular database), the record with key 2 (atom number 2).

(8) The color record for this primitive. Color records always start with 0 (indicating RGB color). The color for the line primitive is white, (255,255,255).

(9) Start of the type-specific information for this primitive. Data for a line primitive consists of two points, each with an x, y, and z component.

(10) Start of the second line in this geometry group.

(11) Start of the second geometry group in this file.

(11) Start of a sphere definition.

(13) Start of a cylinder definition. The cylinder has no database pointers.

(14) Start of a triangle definition.

(15) Start of a text definition.

3.2 File format description

The structure of the file is:

```
<geometry group 1>
[ <geometry group 2> ]
        .
        .
        .
[ <geometry group n> ]
```

The format of each <geometry group> is:

```
z

<number of databases>

[ <database file name 0> ]
[ <database file name 1> ]
        .
        .
        .
[ <database file name m> ]

<geometric primitive 1>
[<geometric primitive 2>]
        .
        .
        .
[<geometric primitive p>]
```

The <number of databases> is an integer and is followed by that number of fully qualified database file names. Note that if <number of databases> is zero, no file names follow.

The format of each <geometric primitive> is:

```
<type code>

[ <database pointer 1> ]
[ <database pointer 2> ]
        .
        .
        .
```

[<database pointer r>]

0 <red> <green> <blue>

<type specific data>

<type code> is a single character as follows:

c – cylinder
l – line
s – sphere
t – triangle
x – text

The format of each <database pointer> is:

<database index> <subset index> <key>

<database index> is the index number of the database in the list of databases at the start of the group. Note that the first database has index number zero. Similarly <subset index> is the index number of the subset in the database, also starting with zero for the first. Molecular databases have two subsets, "atom" (index 0) and "bond" (index 1). The <key> is an integer key that indicates the record. For "atom" records, <key> will be the atom number.

<red>, <green>, <blue> are the components of the color for the primitive. Each must be an integer value between 0 (no color) and 255 (fully saturated).

The <type specific data> for each geometric primitive type is presented below:

cylinder:

<end point 1>
<end point 2>
<radius>
<tessellation factor>
<end point 1 sphere cap code>
<end point 2 sphere cap code>
<end point 1 flat cap code>
<end point 2 flat cap code>

<end point 1> and <end point 2> each consist of an x, y, and z component (separated by white space) for one of the two end points of the cylinder axis.

<radius> is the cylinder radius.

<tessellation factor> is an integer which specifies how many facets the cylinder should be drawn with (cylinders are drawn using rectangular facets that run the length of the cylinder).

The <sphere cap code>s indicate for each end whether a sphere is to be drawn at that end (to cap the cylinder). 1 = draw a sphere, 0 = no sphere. Similarly, the <flat cap code>s indicate for each end whether a flat cap is to be drawn. If neither a sphere or flat cap is drawn for a given end, that end will be open.

line:

<end point 1>
<end point 2>

<end point 1> and <end point 2> each consist of an x, y, and z component (separated by white space) for one of the two end points of the line.

sphere:

<center point>
<radius>
<tessellation factor>

<center point> consists of an x, y, and z component (separated by white space) for the center of the sphere.

<radius> is the sphere radius.

<tessellation factor> is an integer between 1 and 30 which specifies the level of detail for rendering the sphere. The higher the number, the finer the tessellation. Highly tessellated spheres will appear smooth, but will take longer to draw then spheres with lower tessellation. Section 4.3.6.3 of *VIEW Interactive Tool Definition Language – Language Description* presents a table showing how the tessellation factor affects the number of polygons used to actually draw the sphere.

text:

<text string>
<text position>

The <text string> is the text to be display.

<text position> consists of an x, y, and z component (separated by white space) for the lower left corner of the text.

triangle:

```
<backcolor flag>
[ <backcolor> ]
<vertex 1> <vertex 1 normal>
<vertex 2> <vertex 2 normal>
<vertex 3> <vertex 3 normal>
```

<backcolor flag> is either 0, indicating that no backcolor is specified for the triangle, or 1 indicating that one is. If no backcolor is specified, the color specified for the triangle will be applied to both front and back faces. <backcolor> (if specified) consists of a red, green, and blue component, specified as described above.

<vertex n> consists of an x, y, and z component (separated by white space) for a vertex of the triangle. <vertex n normal> consists of the x, y, and z components of a surface normal at this vertex. The vertex normals MUST be of unit length.

VIEW Data Directory Structure

Larry Bergman
1/22/93

The files used by the VIEW system are stored in a fixed subdirectory structure within a common directory area. The four types of files handled by VIEW as well as PDB files (used by the *pdbtoview* program) are stored in subdirectories as follows.

Subdirectory name	**Types of files stored**
Database	VIEW database files
Geometry	VIEW Geometry files
Pdb	PDB files
Snapshot	VIEW Snapshot files
Tool	VIEW Tool files (scripts)

The directory under which these subdirectories is located is provided as a parameter to the *runview* command, used to run the VIEW system (although it is not required by the alternate *iview* command). It is also provided to the *pdbtoview* program used to convert PDB files to VIEW database format.

VIEW System Known Bugs

Larry Bergman
1/25/92

System crashes

On occasion the VIEW system may crash. As you use the system in ways that we did not anticipate, it is possible that the system may fail. System failure will be seen as a complete lack of response from the system, and a failure message in the window that VIEW was run from. This message will usually say either "Segment fault" or "Bus error".

If the system crashes, try to recreate the condition that caused the failure. If you are able to reproduce the failure with a sequence of actions that you can describe, please report the bug. To report the bug, send me mail at bergman@cs.unc.edu and describe the failure conditions. Please also mail any databases, geometry files and tools that contribute to the failure. I would also appreciate reports of any odd behaviors that you noticed, even if they do not cause complete system failure.

System hangs

Occasionally, VIEW gets stuck trying to open a new window. The only symptom of this is that the system will not respond (in fact, it looks just like a system crash, but no error message is reported). If the system hangs, you must kill the VIEW process (by typing your kill character, often cntl-c in the window VIEW was run from).

If a red stop message is displayed, be patient, the system is probably in the midst of loading a file, and is not hung. Sometimes file loads can take several minutes. Some other system operations, such as popping up the **All tools** panel, also take a short while to complete, during which time the system will be unresponsive.

System slow-downs

After you have run VIEW for a while and loaded several geometry and/or database files, you may notice the system response becoming slower. This is because VIEW is beginning to run out of memory. The only cure for this is to save your work (by writing geometry and/or snapshotting), exiting VIEW, and restarting the system.

User interface

Occasionally the virtual track ball will get stuck. That is, when you release the mouse button(s), mouse movement will still rotate, scale or translate the geometry. To release the geometry, click the left, middle, or both mouse buttons with the cursor inside the display panel (try all combinations until one of them works).

Occasionally pressing one of the virtual buttons (by positioning the mouse cursor over the button and then depressing one of the mouse buttons) will fail to have any effect – you will not see the button depress as it normally does. To resolve this problem, move the mouse cursor completely outside the panel that contains the button, then move the mouse back to the button and try again.

The system automatically maintains a clipping volume that contains all on-screen geometry. On occasion, an operation will cause geometry to stray outside that volume, and the geometry will be clipped out of the display. If the geometry is unexpectedly clipped, press **Recenter** in the main display, which will update the clipping volume.

Selection

When a number of geometric primitives are candidates for a selection, you may be unable to select. This may happen when a number of triangles are coincident at a point, for example, the tips of arrows drawn by *arrow_atoms* or *arrow_pnts*. To overcome this difficulty, either toggle off some groups (e.g. turn off the arrows and select using other geometry), or zoom in on the area where you are trying to select.

Tool language

Removing geometric primitives from a group within an iterator on that group will not work properly. For example, the code:

```
FOREACH (obj IN grp)
   REMOVE(obj);
```

where grp is a geometry group should be avoided. It will not work properly and may cause the system to crash.

Do not try to concatenate objects to themselves. Code such as:

```
grp &= grp;
```

may cause the tool to go into an infinite loop. If this happens (you will only know because the tool will not terminate), press EXIT to stop the tool.