

# Scripting with TCL, Part 2

**Axel Kohlmeyer**



Center for Molecular Modeling  
University of Pennsylvania

SBS 2007 @ JNCASR, Bangalore

# Lists in TCL

- Lists are a central construct in TCL
- Everything can be seen as a list of objects where each object can be represented as a string and could constitute either an integer, a floating point number, a string or a list
- Even a TCL program is a list of lines and each line is a list (command arg1 arg2 ...)
- A list is a string with whitespace separated objects and can be grouped with `{}` or `""`
- Lists are the closest equivalent to arrays in C

# Creating TCL Lists

- There are 3 basic ways of creating a list:

- Explicitly:

```
set lst {{item 1} {item 2}}
```

or:

```
set lst {item 1} {item 2}
```

Note: only in the second form items can be expanded from variables

- Using `split`:

```
set lst [split "item 1,item 2" ","]
```

- Using `list`:

```
set lst [list "item 1" "item 2"]
```

# Accessing TCL List Elements

- Use `lindex` to access a single list element:  

```
set lst [split abcdefg ]  
puts    first character is [lindex $lst 0]
```
- Note that indexing starts with 0 (C/C++ style)
- Use `llength` to determine the length of a list:  

```
puts    the word has [llength $lst] chars
```
- Use `foreach` to loop over the elements of a list:  

```
foreach c $lst {  
    puts    the next char is: $c  
}
```

# TCL List Manipulations

- Since lists are an important part of TCL there are many commands for list manipulations

```
set lst {a b {d e} {{f} {g h}} i}
```

- Appending a list elements: lappend

```
lappend lst {j k l} m n o pq
```

- Replacing list elements: lreplace

```
set nlst [lreplace $lst 1 3 xx \{ \}]
```

- Inserting list elements: linsert

```
set nlst [linsert $lst 0 1 2 3]
```

- Reassigning list element values: lset

- lset lst 1 B

# More TCL List Manipulations

- **Sorting lists:** `set s1st [lsort $1st]`  
sorts as ASCII string by default.  
With option `-unique` duplicates are removed:  
`lsort -unique [$sel get rename]`
- **Searching lists:** `lsearch {list} {pattern}`
- **Returns index of first match:** `lsearch $1st n`  
or list of all matches: `lsearch -all $1st x`
- **Combining a list into a string:** `join $1st ,`
- **Assigning list elements to variables (VMD only):**  
`lassign [lindex $coords 2] x y z`

# TCL File I/O

- To open a file use the `open` command.
- Syntax: `open {filename} {access}`
- Open returns a context dependent “unique file handle” so command substitution is required in scripts to keep them generic:  

```
set fp [open output.dat w]
```
- Subsequent operations on this file use `$fp`:  

```
puts $fp hello, file!  
close $fp
```
- File I/O is by default buffered.

# More TCL File I/O

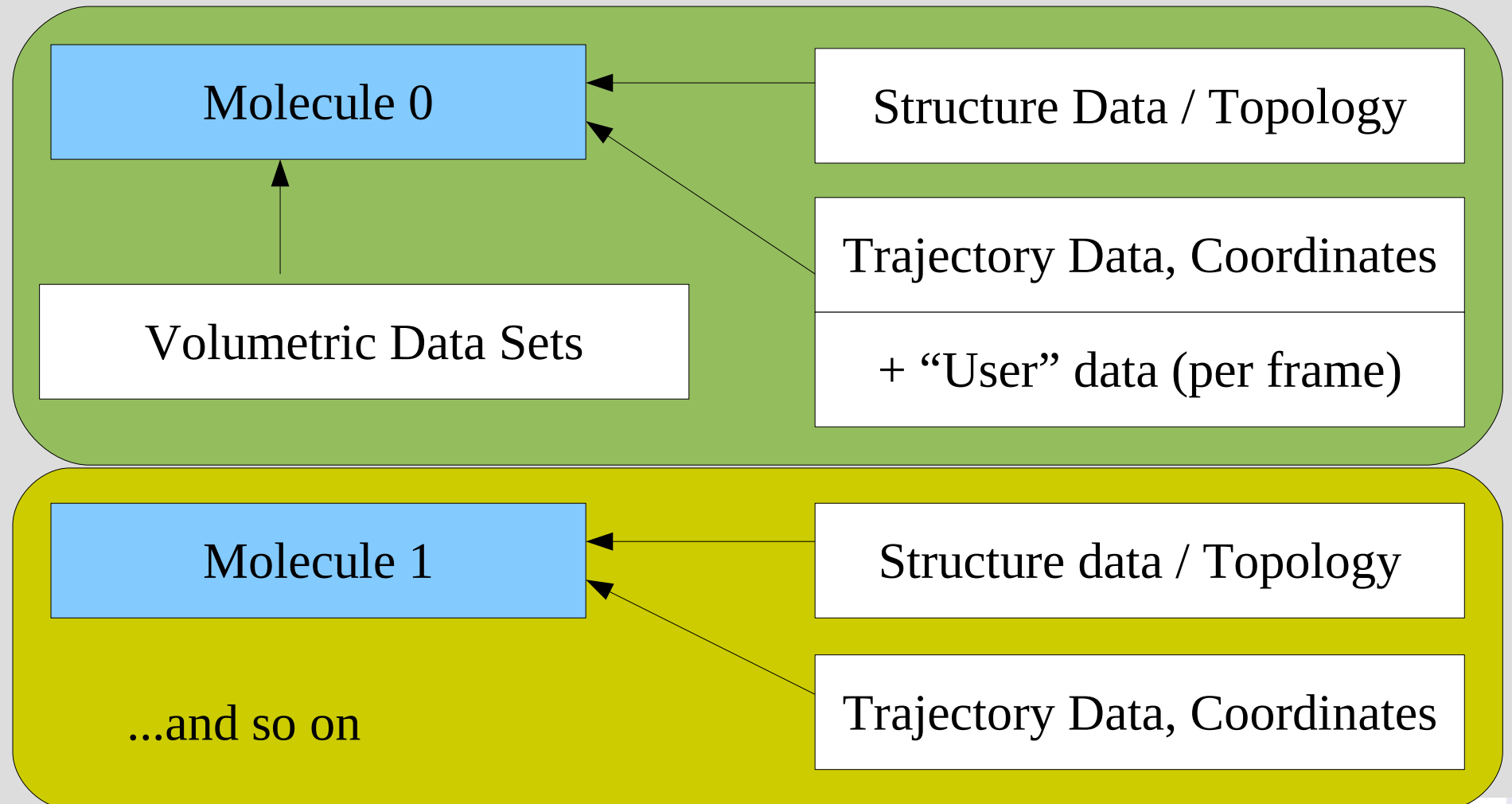
- Typical access: r : read, w : write, a : append
- Reading from file with gets (returns a line):  
`set line [gets $fp]`
- Alternatively read to variable (returns #chars):  
`gets $fp line`
- Pre-connected channels: stdout, stdin, stderr
- Read from keyboard with: `gets stdin`
- To sync I/O buffers to disk: `flush $fp`
- Always close files; finite number file handles
- Use of `$fp` invalid after close => `unset fp`

# Formatting Text in TCL

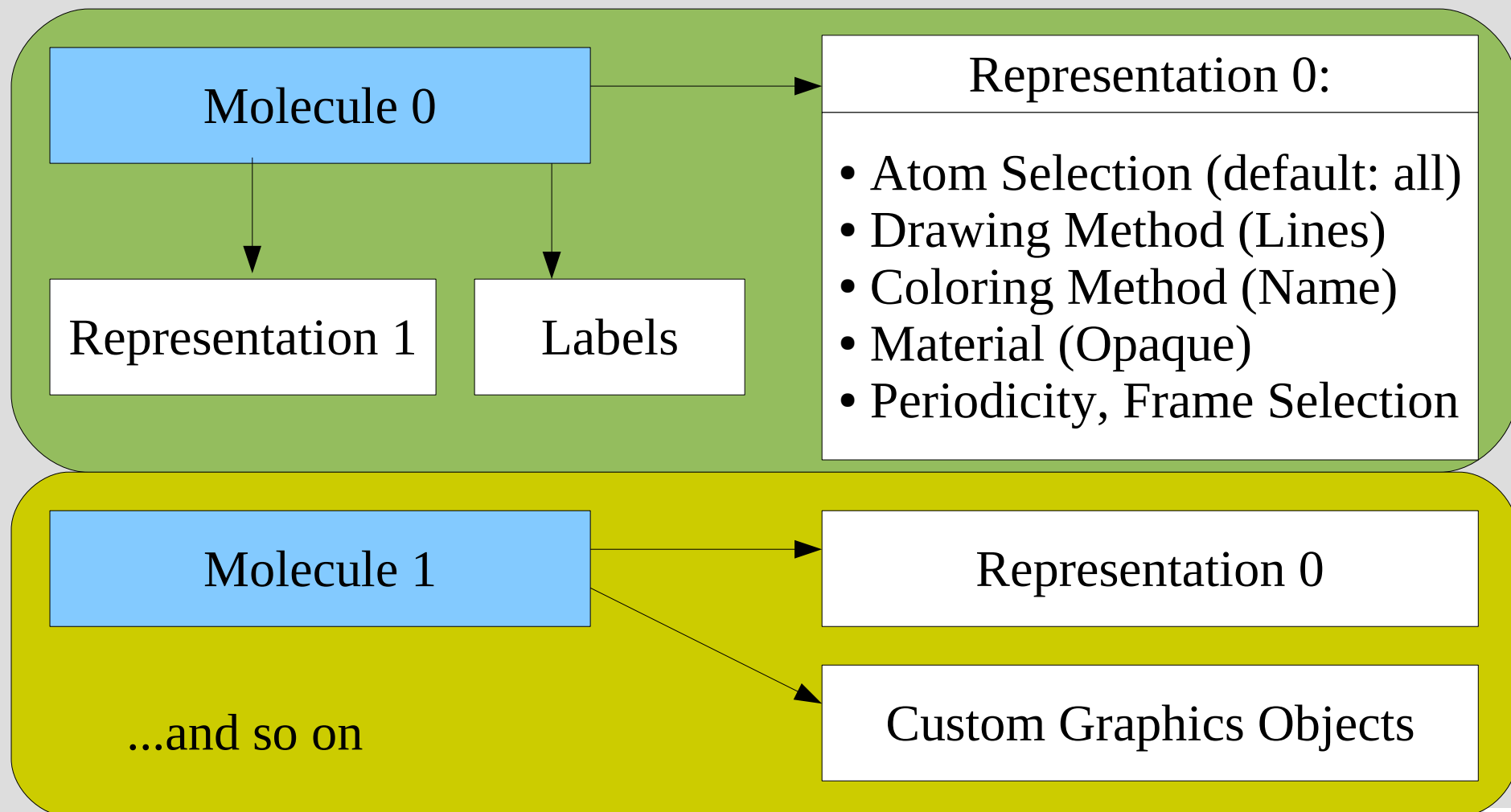
- printf() style formatting is done with `format`
- Syntax: `format {fmt} {a1} {a2} ...`
- Format conversions like in printf with % (`%d`, `%s`, `%g`, `%f`, ...),
- Each % requires an argument to format
- For I/O and assignments use TCL command substitution with `[ ]`. Examples:

```
puts [format value = %8.2f 11.1]
set out [format %-5s: %5d value 5]
puts [format line%d $i]: $out
```

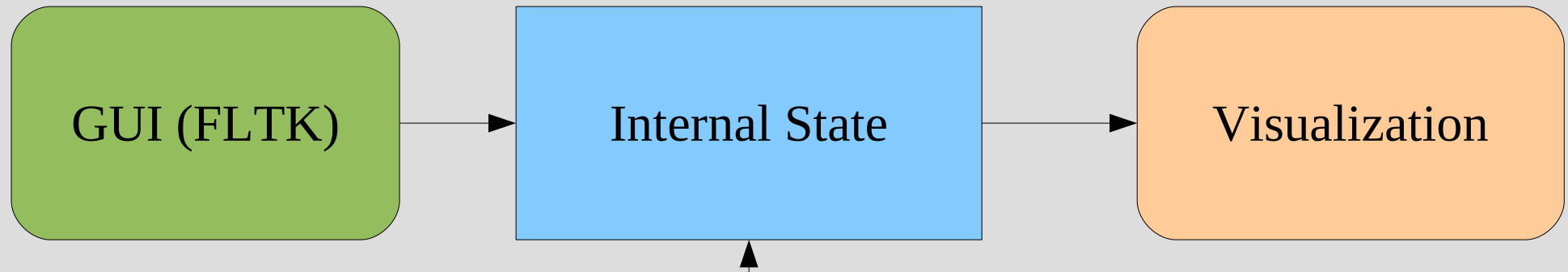
# The VMD Data Model



# The VMD Visualization Model



# The VMD Execution Model



Use `logfile console` to follow the changes of the internal state as VMD/TCL commands.  
`logfile name` writes the log to a file.  
`logfile off` turns logging off.

# The VMD atomselect command

- `atomselect` is the most powerful command in VMD and the cornerstone of most analysis or atom- or molecule-manipulation scripts
- Sadly, it is also the hardest to understand command and thus the root of many bugs
- `atomselect` is **special**: `atomselect` creates a new TCL command called `atomselect#` (`#` is a unique number)
- Name of this command is the return value
- `atomselect#` is **not** automatically deleted

# More on atomselect

- Syntax: `atomselect {molid} {seltxt}`
- Since the exact name of the command created by `atomselect` is not known in advance so command substitution and variables are used:  

```
set sel1 [atomselect 0 all]  
set sel2 [atomselect top {resname GLY}]
```
- Procedures created with `atomselect` have to be deleted explicitly to avoid memory leaks:  

```
$sel1 delete ; # $sel1 becomes invalid  
unset sel1 ; # removes variable. Not command
```

# Using Created Selections

- With the generated selections we can query and change properties of the selected atoms
- General options
  - Number of atoms in selection: `$sel num`
  - Indexes of atoms in selection: `$sel list`
  - Text used to create selection: `$sel text`
  - Molecule Id for selection: `$sel molid`

NOTE: selections are bound to specific VMD molecules and can not span two or more of them

  - Write selection to file: `$sel writepdb mysel.pdb`
  - NOTE: writepsf will create an incomplete .psf file.

# More General Selection Options

- Advance selection in trajectory:  
`$sel frame 10`
- Allows to monitor evolution of time dependent properties (coordinates).
- Recompute selection: `$sel update`
- Selections are by default not recomputed.  
`$sel frame #` will follow the same atoms, even if the selection conditions are not met anymore
- `$sel moveby {x y z}` (translate selected atoms)
- `$sel move {matrix}` (use transformation matrix)

# Get/Set with atomselect

- With `$sel get` you can query properties:  
`set coords [$sel get {x y z}]`  
`$sel get {resid residue resname}`
- Using `get` on an atom selection will always return a list of lists, even if the selection matches only one atom and you ask for only one property.
- When using `set` one needs to provide a list with a matching number of elements or a scalar (will be assigned to all elements)

# More on Get/Set with Selections

- Most properties are global for a VMD molecule  
Exceptions: `x`, `y`, `z`, `user` (from VMD 1.8.7 on  
also: `vx`, `vy`, `vz`, `user2`, `user3`, `user4`)
- Values from `user` can be used for colorization  
or in selection texts. Example:  

```
set sel [atomselect 0 {user == 3}]
```
- Creating and updating selections is time  
consuming, so it is frequently better to query  
for all desired properties, store them in a list  
and then process the list.

# Use of Selections in measure

- The measure command can use selections to define the set of atoms to work on.
- To get the (geometrical) center do:  
`set com [measure center $sel]`
- To get the transformation matrix for at best fit between two selections:  
`set T [measure fit $sel1 $sel2]`
- This can be use to align two molecules:  
`$all move $T`
- For more options see tutorial and user's guide

# Odds and Ends

- The `atomselect` command is not bound to and graphical representation and works as well in text mode => batch processing, analysis
- To start VMD executing a specific script at startup you can do: `vmd -e myscript.tcl`
- To write the trajectory of a selection (you cannot use `$sel writeXXX`) use:  
`animate write dcd slc.dcd sel $sel`