psfgen User's Guide

Version 2.0

João V. Ribeiro, Brian Radak, John Stone, Justin Gullingsrud,
Jan Saam, and Jim Phillips

May 5, 2020

Theoretical and Computational Biophysics Group
University of Illinois and Beckman Institute
405 N. Mathews
Urbana, IL 61801

1 Creating PSF Structure Files

The `psfgen` structure building tool consists of a portable library of structure and file manipulation routines with a Tcl interface. Current capabilities include

- reading CHARMM topology files
- reading psf files in X-PLOR/NAMD format
- extracting sequence data from single segment PDB files
- generating a full molecular structure from sequence data
- applying patches to modify or link different segments
- writing NAMD and VMD compatible PSF structure files
- extracting coordinate data from PDB files
- constructing (guessing) missing atomic coordinates
- deleting selected atoms from the structure
- writing NAMD and VMD compatible PDB coordinate files

We are currently refining the interface of `psfgen` and adding features to create a complete molecular building solution. We welcome your feedback on this new tool.

2 New commands and Functionalities

The version 2.0 of `psfgen` was extensively modified and improved to meet the current standards in the size of the structures, and the modern versions of additive CHARMM force field, and polarizable DRUDE force field (http://mackerell.umaryland.edu/charmm_ff.shtml).

New functionalities include:

- hydrogen mass repartition
- structure preparation for Drude force field
- structure preparation containing colinear lone pairs (halogen atoms in the latest additive CHARMM force field version)
- psfgen log file to store all the information printed to the console

To use the Drude force field, one only needs to load the Drude topology files and prepare the structures as per usual. Most commands are available for both lone pairs in the additive and polarizable force fields, although some operations are not yet available for Drude particles. Atom modification operations, e.g., psfset, and queries with the segment command on the drude particles, are not implemented. We advise the user to use VMD to assign beta and occupancy values during the structure preparation. writemol and readmol commands are not compatible with structure preparation for Drude Force field.

The new `psfgen` log file allows the user to save all the information regularly printed out during a `psfgen` execution script to a file. It is possible to open and close multiple log files in a `psfgen` script, but only one file is active at any given moment. An example of an application of multiple log files is to save the information of the loading process of the topology files to one log file and the rest of information of structure preparation to another file, as demonstrated below:

```
psfgen_logfile "load_topoplogy.log"

topology top_all22_prot.rtf
topology top_all36_carb.rtf
topology top_all36_lipid.rtf
topology top_all36_prot.rtf
topology top_all36_cgenff.rtf
topology toppar_water_ions.str

psfgen_logfile close

psfgen_logfile "structure_preparation.log"

segment BPTI {
 pdb output/6PTI_protein.pdb
}

patch DISU BPTI:5 BPTI:55
patch DISU BPTI:14 BPTI:38
patch DISU BPTI:30 BPTI:51

pdbalias atom ILE CD1 CD
coordpdb output/6PTI_protein.pdb BPTI

pdbalias residue HOH TIP3
segment SOLV {
 auto none
 pdb output/6PTI_water.pdb
}

pdbalias atom HOH O OH2
coordpdb output/6PTI_water.pdb SOLV

guesscoord

writepsf output/bpti.psf
writepdb output/bpti.pdb

psfgen_logfile close
```

3 Ordinary Usage

`psfgen` is currently distributed in two forms. One form is as a standalone program implemented as a Tcl interpreter which reads commands from standard output. You may use loops, variables, etc. as you would in a VMD or NAMD script. You may use psfgen interactively, but we expect it to be run most often with a script file redirected to standard input. The second form is as a Tcl package which can be imported into any Tcl application, including VMD. All the commands available to the standalone version of psfgen are available to the Tcl package; using `psfgen` within VMD lets you harness VMD's powerful atom selection capability, as well as instantly view the result of your structure building scripts. Examples of using `psfgen` both with and without VMD are provided in this document.

Generating PSF and PDB files for use with NAMD will typically consist of the following steps:

1. Preparing separate PDB files containing individual segments of protein, solvent, etc. before running `psfgen`.
2. Reading in the appropriate topology definition files and aliasing residue and atom names found in the PDB file to those found in the topology files. This will generally include selecting a default protonation state for histidine residues.
3. Generating the default structure using segment and pdb commands.
4. Applying additional patches to the structure.
5. Reading coordinates from the PDB files.
6. Deleting unwanted atoms, such as overlapping water molecules.
7. Guessing missing coordinates of hydrogens and other atoms.
8. Writing PSF and PDB files for use in NAMD.

3.1 Preparing separate PDB files

Many PDB files in the PDB databank contain multiple chains, corresponding to protein subunits, water, and other miscellaneous groups. Protein subunits are often identified by their chain ID in the PDB file. In `psfgen`, each of these groups must be assigned to their own *segment*. This applies most strictly in the case of protein chains, each of which must be assigned to its own segment so that N-terminal and C-terminal patches can be applied. You are free to group water molecules into whatever segments you choose.

Chains can be split up into their own PDB files using your favorite text editor and/or Unix shell commands, as illustrated in the BPTI example below. If you are using VMD you can also use atom selections to write pieces of the structure to separate files:

```
# Split a file containing protein and water into separate segments.
# Creates files named myfile_water.pdb, myfile_frag0.pdb, myfile_frag1.pdb,...
# Requires VMD.
mol load pdb myfile.pdb
set water [atomselect top water]
$water writepdb myfile_water.pdb
set protein [atomselect top protein]
set chains [lsort -unique [$protein get pfrag]]
foreach chain $chains {
  set sel [atomselect top "pfrag $chain"]
  $sel writepdb myfile_frag${chain}.pdb
}
```

3.2 Deleting unwanted atoms

The `delatom` command described below allows you to delete selected atoms from the structure. It's fine to remove atoms from your structure before building the PSF and PDB files, but you should never edit the PSF and PDB files created by `psfgen` by hand as it will probably mess up the internal numbering in the PSF file.

Very often the atoms you want to delete are water molecules that are either too far from the solute, or else outside of the periodic box you are trying to prepare. In either case VMD atom selections can be used to select the waters you want to delete. For example:

```
# Load a pdb and psf file into both psfgen and VMD.
resetpsf
readpsf myfile.psf
coordpdb myfile.pdb
mol load psf myfile.psf pdb myfile.pdb
# Select waters that are more than 10 Angstroms from the protein.
set badwater1 [atomselect top "name OH2 and not within 10 of protein"]
# Alternatively, select waters that are outside our periodic cell.
set badwater2 [atomselect top "name OH2 and (x<-30 or x>30 or y<-30 or>30
                              or z<-30 or z>30)"]
# Delete the residues corresponding to the atoms we selected.
foreach segid [$badwater1 get segid] resid [$badwater1 get resid] {
  delatom $segid $resid
}
# Have psfgen write out the new psf and pdb file (VMD's structure and
# coordinates are unmodified!).
writepsf myfile_chopwater.psf
writepdb myfile_chopwater.pdb
```

4 BPTI Example

To actually run this demo requires

- the program `psfgen` from any NAMD distribution,
- the CHARMM topology and parameter files `top_all22_prot.inp` and `par_all22_prot.inp` from http://mackerell.umaryland.edu/charmm_ff.shtml, and
- the BPTI PDB file `6PTI.pdb` available from the Protein Data Bank at http://www.pdb.org/ by searching for `6PTI` and downloading the complete structure file in PDB format.

Building the BPTI structure

In this demo, we create the files `bpti.psf` and `bpti.pdb` in the output directory which can then be used for a simple NAMD simulation.

```
# File: bpti_example.tcl
# Requirements: topology file top_all22_prot.inp in directory toppar
#               PDB file 6PTI.pdb in current directory

# Create working directory; remove old output files
mkdir -p output
rm -f output/6PTI_protein.pdb output/6PTI_water.pdb

# (1) Split input PDB file into segments}
grep -v '^HETATM' 6PTI.pdb > output/6PTI_protein.pdb
grep 'HOH' 6PTI.pdb > output/6PTI_water.pdb

# (2) Embed the psfgen commands in this script
psfgen << ENDMOL

# (3) Read topology file
topology toppar/top_all22_prot.inp
```

```
# (4) Build protein segment
segment BPTI {
 pdb output/6PTI_protein.pdb
}

# (5) Patch protein segment
patch DISU BPTI:5 BPTI:55
patch DISU BPTI:14 BPTI:38
patch DISU BPTI:30 BPTI:51

# (6) Read protein coordinates from PDB file
pdbalias atom ILE CD1 CD    ; # formerly "alias atom ..."
coordpdb output/6PTI_protein.pdb BPTI

# (7) Build water segment
pdbalias residue HOH TIP3   ; # formerly "alias residue ..."
segment SOLV {
 auto none
 pdb output/6PTI_water.pdb
}

# (8) Read water coordinaes from PDB file
pdbalias atom HOH O OH2     ; # formerly "alias atom ..."
coordpdb output/6PTI_water.pdb SOLV

# (9) Guess missing coordinates
guesscoord

# (10) Write structure and coordinate files
writepsf output/bpti.psf
writepdb output/bpti.pdb

# End of psfgen commands
ENDMOL
```

Step-by-step explanation of the script:

(1) Split input PDB file into segments. 6PTI.pdb is the original file from the Protein Data Bank. It contains a single chain of protein and some PO4 and H2O HETATM records. Since each segment must have a separate input file, we remove all non-protein atom records using grep. If there were multiple chains we would have to split the file by hand. Create a second file containing only waters.

(2) Embed the psfgen commands in this script. Run the psfgen program, taking everything until "ENDMOL" as input. You may run psfgen interactively as well. Since psfgen is built on a Tcl interpreter, you may use loops, variables, etc., but you must use `$$` for variables when inside a shell script. If you want, run psfgen and enter the following commands manually.

(3) Read topology file. Read in the topology definitions for the residues we will create. This must match the parameter file used for the simulation as well. Multiple topology files may be read in since psfgen and NAMD use atom type names rather than numbers in psf files.

(4) Build protein segment. Actually build a segment, calling it BPTI and reading the sequence of residues from the stripped pdb file created above. In addition to the pdb command, we could specify

residues explicitly. Both angles and dihedrals are generated automatically unless "auto none" is added (which is required to build residues of water). The commands "first" and "last" may be used to change the default patches for the ends of the chain. The structure is built when the closing } is encountered, and some errors regarding the first and last residue are normal.

(5) Patch protein segment. Some patch residues (those not used to begin or end a chain) are applied after the segment is built. These contain all angle and dihedral terms explicitly since they were already generated. In this case we apply the patch for a disulfide link three separate times.

(6) Read protein coordinates from PDB file. The same file used to generate the sequence is now read to extract coordinates. In the residue ILE, the atom CD is called CD1 in the pdb file, so we use "pdbalias atom" to define the correct name. If the segment names in the pdb file match the name we gave in the segment statement, then we don't need to specify it again; in this case we do specify the segment, so that all atoms in the pdb file must belong to the segment.

(7) Build water segment. Build a segment for the crystal waters. The residue type for water depends on the model, so here we alias HOH to TIP3. Because CHARMM uses an additional H-H bond we must disable generation of angles and dihedrals for segments containing water. Then read the pdb file.

(8) Read water coordinates from PDB file. Alias the atom type for water oxygen as well and read coordinates from the file to the segment SOLV. Hydrogen doesn't show up in crystal structures so it is missing from this pdb file.

(9) Guessing missing coordinates. The tolopogy file contains default internal coordinates which can be used to guess the locations of many atoms, hydrogens in particular. In the output pdb file, the occupancy field of guessed atoms will be set to 0, atoms which are known are set to 1, and atoms which could not be guessed are set to -1. Some atoms are "poorly guessed" if needed bond lengths and angles were missing from the topology file. Similarly, waters with missing hydrogen coordinates are given a default orientation.

Write structure and coordinate files. Now that all of the atoms and bonds have been created, we can write out the psf structure file for the system. We also create the matching coordinate pdb file. The psf and pdb files are a matched set with identical atom ordering as needed by NAMD.

Using generated files in NAMD.

The files bpti.pdb and bpti.psf can now be used with NAMD, but the initial coordinates require minimization first. The following is an example NAMD configuration file for the BPTI example.

```
# NAMD configuration file for BPTI

# molecular system
structure output/bpti.psf

# force field
paratypecharmm on
parameters toppar/par_all22_prot.inp
exclude scaled1-4
1-4scaling 1.0

# approximations
switching on
switchdist 8
```

```
cutoff 12
pairlistdist 13.5
margin 0
stepspercycle 20

#integrator
timestep 1.0

#output
outputenergies 10
outputtiming 100
binaryoutput no

# molecular system
coordinates output/bpti.pdb

#output
outputname output/bpti
dcdfreq 1000

#protocol
temperature 0
reassignFreq 1000
reassignTemp 25
reassignIncr 25
reassignHold 300

#script

minimize 1000

run 20000
```

5 Building solvent around a protein

The following script illustrates how `psfgen` and VMD can be used together to add water around a protein structure. It assumes you already have a psf and pdb file for your protein, as well as a box of water which is large enough to contain the protein. For more information on how atomselections can be used within VMD scripts, see the VMD User's Guide.

```
proc addwater { psffile pdbfile watpsf watpdb } {
# Create psf/pdb files that contain both our protein as well as
# a box of equilibrated water.  The water box should be large enough
# to easily contain our protein.
resetpsf
readpsf $psffile pdb $pdbfile
readpsf $watpsf pdb $watpdb

# Load the combined structure into VMD
writepsf combine.psf
writepdb combine.pdb
mol load psf combine.psf pdb combine.pdb
```

```
# Assume that the segid of the water in watpsf is QQQ
# We want to delete waters outside of a box ten Angstroms
# bigger than the extent of the protein.
set protein [atomselect top "not segid QQQ"]
set minmax [measure minmax $protein]
foreach {min max} $minmax { break }
foreach {xmin ymin zmin} $min { break }
foreach {xmax ymax zmax} $max { break }
    set xmin [expr $xmin - 10]
    set ymin [expr $ymin - 10]
    set zmin [expr $zmin - 10]
    set xmax [expr $xmax + 10]
    set ymax [expr $ymax + 10]
    set zmax [expr $zmax + 10]

# Center the water on the protein.  Also update the coordinates held
# by psfgen.
set wat [atomselect top "segid QQQ"]
$wat moveby [vecsub [measure center $protein] [measure center $wat]]
foreach atom [$wat get {segid resid name x y z}] {
foreach {segid resid name x y z} $atom { break }
coord $segid $resid $name [list $x $y $z]
}

# Select waters that we don't want in the final structure.
set outsidebox [atomselect top "segid QQQ and (x <= $xmin or y <= $ymin \
or z <= $zmin or x >= $xmax or y >= $ymax or z >= $xmax)"]
set overlap [atomselect top "segid QQQ and within 2.4 of (not segid QQQ)"]

# Get a list of all the residues that are in the two selections, and delete
# those residues from the structure.
set reslist [concat [$outsidebox get resid] [$overlap get resid]]
set reslist [lsort -unique -integer $reslist]

foreach resid $reslist {
delatom QQQ $resid
}

# That should do it - write out the new psf and pdb file.
writepsf solvate.psf
writepdb solvate.pdb

# Delete the combined water/protein molecule and load the system that
# has excess water removed.
mol delete top
mol load psf solvate.psf pdb solvate.pdb

# Return the size of the water box
return [list [list $xmin $ymin $zmin] [list $xmax $ymax $zmax]]
}
```

6 New Commands in the version 2.0

- `psfgen_logfile` <*file name*> [`close`]
  Purpose: Open or close a log file to store all information printed to the console.
  Arguments: <*file name*>: Valid file name in the current directory.
  `close`: Close the active log file. The file name should not be included in the closing command. Example above.
  Context: Any part of the script, context independent. May call multiple times.

- `hmassrepart` [*dowater <1 0>*] [*mass <target hydrogen mass>*]
  Purpose: Partition the mass of heavy atoms into the bonded hydrogen atoms.
  Arguments: `dowater`: 1 for true, 0 for false. Partition the water molecules. Default value 0.
  `mass`: Target for the hydrogen atoms' mass. Default value 3.024 amu.
  Context: After loading or preparing the structure.

- `vpbonds` [1 0]
  Purpose: Print the bonds between the virtual particles (drude particles and lone pairs) and their hosts.
  Arguments: 1 for true, 0 for false. Default value 1.
  Context: Before writing the psf file. May call multiple times. WARNING: To run simulations containing lone pairs or Drude particles on NAMD 2.13, set vpbonds to 0.

7 List of Commands

- `topology` [`list`] <*file name*>
  Purpose: Read in molecular topology definitions from file.
  Arguments: <*file name*>: CHARMM format topology file.
  `list`: Lists all currently specified topology files.
  `residues`: Return a list of the known residue topologies.
  `patches`: Return a list of the known residue patches.
  Context: Beginning of script, before segment. May call multiple times.

- `topology alias` <*alternate residue name*> <*existing residue name*>
  Purpose: Provide alternate names for residues found in topology file. An alternate name used to generate a residue will be used on output. Compare to "pdbalias residue" below, in which the real name is used on output.
  Arguments: <*alternate residue name*>: Desired residue name.
  <*existing residue name*>: Residue name found in topology file.
  Context: Before reading sequence with pdb. May call multiple times.

- `pdbalias residue` <*alternate name*> <*real name*>
  Purpose: Provide translations from residues found in PDB files to proper residue names read in from topology definition files. Proper names from topology files will be used in generated PSF and PDB files. Compare to "topology alias" above, in which the alias is is used as the residue name in generated files. This command also exists under the deprecated name `alias`.
  Arguments: <*alternate name*>: Residue name found in PDB file.
  <*real name*>: Residue name found in topology file or aliases.
  Context: Before reading sequence with pdb. May call multiple times.

- `segment` [`segids`] [`resids`] [`residue`] [`first`] [`last`] <*segment ID*> [*resid*] [*atom name*] { <*commands*> }
  Purpose: Build a segment of the molecule. A segment is typically a single chain of protein or DNA, with default patches applied to the termini. Segments may also contain pure solvent or lipid. Options [`segids`] [`resids`] [`residue`] [`first`] [`last`] are used to query information about the specified segment.

Arguments: `segids`: Return a list of segids for the molecule in the current context.
`resids`: Return a list of resids for the given segment in the current context.
`residue`: Return the residue name of the residue in the given segment with the given resid.
`atoms`: Return a list of atoms for the given segment with the given resid.
`coordinates`: Return x, y, z coordinates for the given atom.
`velocities`: Return x, y, z velocities for the given atom.
`mass`: Return the mass of the given atom.
`charge`: Return the charge of the given atom.
`atomid`: Return the one-based atomid of the given atom. These are only assigned/updated when writing a file. Therefore `writepsf`, `writepdb`, or `writemol` must be called to avoid returning old atomid values or zero.
`first`: Returns the name of the patch that was applied to the beginning of the specified segment.
`last`: Returns the name of the patch that was applied to the end of the specified segment.
<*segment ID*>: Unique name for segment, 1–4 characters.
<*commands*>: Sequence of commands in Tcl syntax to build the primary structure of the segment, including auto, first, last, residue, pdb, etc.
Context: After topology definitions and residue aliases. May call multiple times. Structure information is generated at the end of every segment command.

- `auto` [`angles`] [`dihedrals`] [`none`]
  Purpose: Override default settings from topology file for automatic generation of angles and dihedrals for the current segment.
  Arguments: `angles`: Enable generation of angles from bonds.
  `dihedrals`: Enable generation of dihedrals from angles.
  `none`: Disable generation of angles and dihedrals.
  Context: Anywhere within segment, does not affect later segments.

- `first` <*patch name*>
  Purpose: Override default patch applied to first residue in segment. Default is read from topology file and may be residue-specific.
  Arguments: <*patch name*>: Single-target patch residue name or `none`.
  Context: Anywhere within segment, does not affect later segments.

- `last` <*patch name*>
  Purpose: Override default patch applied to last residue in segment. Default is read from topology file and may be residue-specific.
  Arguments: <*patch name*>: Single-target patch residue name or `none`.
  Context: Anywhere within segment, does not affect later segments.

- `residue` <*resid*> <*resname*> [*chain*]
  Purpose: Add a single residue to the end of the current segment.
  Arguments: <*resid*>: Unique name for residue, 1–5 characters, usually numeric. <*resname*>: Residue type name from topology file. <*chain*>: Single-character chain identifier.
  Context: Anywhere within segment.

- `pdb` <*file name*>
  Purpose: Extract sequence information from PDB file when building segment. Residue IDs will be preserved, residue names must match entries in the topology file or should be aliased before pdb is called.
  Arguments: <*file name*>: PDB file containing known or aliased residues.
  Context: Anywhere within segment.

- `mutate` <*resid*> <*resname*>
  Purpose: Change the type of a single residue in the current segment.
  Arguments: <*resid*>: Unique name for residue, 1–5 characters, usually numeric. <*resname*>: New

residue type name from topology file.
Context: Within segment, after target residue has been created.

- `patch` [`list`] <*patch residue name*> <*segid:resid*> [...]
Purpose: Apply a patch to one or more residues. Patches make small modifications to the structure of residues such as converting one to a terminus, changing the protonation state, or creating disulphide bonds between a pair of residues.
Arguments: `list`: Lists all patches applied explicitey using the command 'patch'.
`listall`: Lists all currently applied patches including default patches.
<*patch residue name*>: Name of patch residue from topology definition file.
<*segid:resid*>: List of segment and residue pairs to which patch should be applied.
Context: After one or more segments have been built.

- `regenerate` [`angles`] [`dihedrals`]
Purpose: Remove all angles and/or dihedrals and completely regenerate them using the segment automatic generation algorithms. This is only needed if patches were applied that do not correct angles and bonds. Segment and file defaults are ignored, and angles/dihedrals for the entire molecule are regenerated from scratch.
Arguments: `angles`: Enable generation of angles from bonds.
`dihedrals`: Enable generation of dihedrals from angles.
Context: After one or more segments have been built.

- `regenerate` [`resids`]
Purpose: Remove insertion codes and minimally modify resids to retain uniqueness. No modifications will be made in segments that have monotonically increasing resids and do not contain insertion codes. Within a segment, no modifications will be made to residues preceeding the first non-increasing resid or residue with an insertion code.
Arguments: `resids`: Enable regeneration of resids to remove insertion codes.
Context: After one or more segments have been built.

- `multiply` <*factor*> <*segid*[*:resid*[*:atomname*]]> [...]
Purpose: Create multiple images of a set of atoms for use in locally enhanced sampling. The beta column of the output pdb file is set to 1...<*factor*> for each image. Multiple copies of bonds, angles, etc. are created. Atom, residue or segment names are not altered; images are distinguished only by beta value. This is not a normal molecular structure and may confuse other tools.
Arguments: <*factor*>:
<*segid:resid:atomname*>: segment, residue, or atom to be multiplied. If :resid is omitted the entire segment is multiplied; if :atomname is omitted the entire residue is multiplied. May be repeated as many times as necessary to include all atoms.
Context: After one or more segments have been built, all patches applied, and coordinates guessed. The effects of this command may confuse other commands.

- `delatom` <*segid*> [*resid*] [*atomname*]
Purpose: Delete one or more atoms. If only `segid` is specified, all atoms from that segment will be removed from the structure. If both `segid` and `resid` are specified, all atoms from just that residue will be removed. If `segid`, `resid`, and `atomname` are all specified, just a single atom will be removed.
Arguments: <*segid*>: Segment ID of target atom.
<*resid*>: Residue ID of target atom (optional).
<*atomname*>: Name of target atom (optional).
Context: After one or more segments have been built.

- `resetpsf`
Purpose: Delete all segments in the structure. The topology definitions and aliases are left intact. If you want to clear the topology and aliases as well, use `psfcontext reset` instead.
Arguments:
Context: After one or more segments have been built.

- `psfcontext` [*context*] [`new`] [`delete`]
  Purpose: Switches between complete contexts, including structure, topology definitions, and aliases. If no arguments are provided, the current context is returned. If <*context*> or `new` is specified, a new context is entered and the old context is returned. If `delete` is also specified, the old context is destroyed and "deleted <*old context*>" is returned. An error is returned if the specified context does not exist or if `delete` was specified and the current context would still be in use. *It may be possible to write robust, error-tolerant code with this interface, but it would not be easy. Please employ the following revised* `psfcontext` *usage instead.*
  Arguments: <*context*>: Context ID returned by psfcontext.
  Context: At any time.

- `psfcontext mixedcase`
  Purpose: Make context case sensitive by preserving case of all segment, residue, atom, and patch names on input.
  Arguments:
  Context: Before reading files requiring case sensitive behavior, normally as the first command.

- `psfcontext allcaps`
  Purpose: Make context case insensitive by converting all segment, residue, atom, and patch names to upper case characters on input. This is the default behavior and should match the behavior of versions prior to 1.5.0.
  Arguments:
  Context: Before reading files requiring case insensitive behavior, not needed in normal use.

- `psfcontext reset`
  Purpose: Clears the structure, topology definitions, and aliases, creating clean environment just like a new context.
  Arguments:
  Context: At any time.

- `psfcontext create`
  Purpose: Creates a new context and returns its ID, but does not switch to it. This is different from `psfcontext new` above, which switches to the newly created context and returns the current context's ID.
  Arguments:
  Context: At any time.

- `psfcontext delete` <*context*>
  Purpose: Deletes the specified context. An error is returned if the specified context does not exist or would still be in use. This is different from `psfcontext` <*context*> `delete` above, which switches to the specified context and deletes the current one.
  Arguments: <*context*>: Context ID returned by psfcontext.
  Context: At any time.

- `psfcontext eval` <*context*> { <*commands*> }
  Purpose: Evaluates <*commands*> in the specified context, returning to the current context on exit. This should be totally robust, returning to the orignal context in case of errors and preventing its deletion when nested.
  Arguments: <*context*>: Context ID returned by `psfcontext create`.
  <*commands*>: Script to be executed in the specified context.
  Context: At any time.

- `psfcontext stats`
  Purpose: Returns the total numbers of contexts that have been created and destroyed. This is useful for checking if a script is leaking contexts.

Arguments:
Context: At any time.

- `writepsf` [`charmm`] [`x-plor`] [`cmap`] [`nocmap`] [`nopatches`] <*file name*>
  Purpose: Write out structure information as PSF file. A simplified session log is listed in the REMARKS section of the PSF file.
  Arguments: `charmm`: Use CHARMM format (numbers for atom types).
  `x-plor`: Use X-PLOR format (names for atom types), the default format required by NAMD.
  `cmap`: Write cross-term entries to PSF file if present, the default.
  `nocmap`: Do not write cross-term entries to PSF file, even if present.
  `nopatches`: Do not write list of applied patches to PSF file header.
  <*file name*>: PSF file to be generated.
  Context: After all segments have been built and patched.

- `readpsf` <*file name*> [`pdb`] [*pdb file name*] [`namdbin`] [*namdbin file name*] [`velnamdbin`] [*velocity file name*]
  Purpose: Read in structure information from PSF file and add it to the structure. Optionally also read coordinates and insertion codes from a PDB file, assuming that the atom order is the same in both files. Optionally also read coordinates a NAMD binary file, assuming that the atom order is the same as the psf file. It is an error if any segments in the PSF file already exist.
  Arguments: <*file name*>: PSF file in X-PLOR format (names for atom types).
  `pdb`: Read coordinates and insertion codes from PDB file.
  <*pdb file name*>: PDB file with atoms in same order as PSF file.
  `namdbin`: Read coordinates from NAMD binary file.
  <*namdbin file name*>: NAMD binary file with atoms in same order as PSF file.
  `velnamdbin`: Read velocities from NAMD binary file.
  <*velocity file name*>: NAMD binary velocity file with atoms in same order as PSF file.
  Context: Anywhere but within segment.

- `pdbalias atom` <*residue name*> <*alternate name*> <*real name*>
  Purpose: Provide translations from atom names found in PDB files to proper atom names read in from topology definition files. Proper names from topology files will be used in generated PSF and PDB files. This command also exists under the deprecated name `alias`.
  Arguments: <*residue name*>: Proper or aliased residue name.
  <*alternate name*>: Atom name found in PDB file.
  <*real name*>: Atom name found in topology file.
  Context: Before reading coordinates with coordpdb. May call multiple times.

- `coordpdb` <*file name*> [*segid*] [`namdbin`] [*namdbin file name*]
  Purpose: Read coordinates from PDB file, matching segment, residue and atom names.
  Arguments: <*file name*>: PDB file containing known or aliased residues and atoms.
  <*segid*>: If specified override segment IDs in PDB file.
  `namdbin`: Read coordinates from NAMD binary file.
  <*namdbin file name*>: NAMD binary file with atoms in same order as PDB file.
  Context: After segment has been generated and atom aliases defined.

- `guesscoord`
  Purpose: Guesses coordinates of atoms for which they were not explicitly set. Calculation is based on internal coordinate hints contained in toplogy definition files. When these are insufficient, wild guesses are attempted based on bond lengths of 1 Å and angles of 109°.
  Arguments: None.
  Context: After stucture has been generated and known coordinates read in.

- `coord` <*segid*> <*resid*> <*atomname*> <{ *x y z* }>
  Purpose: WILL BE DEPRECATED AFTER VERSON 1.6 (use `psfset coord` instead) Set

coordinates for a single atom.
Arguments: <*segid*>: Segment ID of target atom.
<*resid*>: Residue ID of target atom.
<*atomname*>: Name of target atom.
<{ *x y z* }>: Coordinates to be assigned.
Context: After structure has been generated.

- `psfset` <*attribute*> <*segid*> [*resid*] [*atomname*] <*value*>
  Purpose: Set an attribute for a given segment, residue, or atom.
  Arguments: <*attribute*>: Segment attributes: `segid`: the name of the segment Residue attributes: `resname`: the name of the residue Atom attributes: `name`: the name of the atom, `mass`: the mass of the atom, `charge`: the charge of the atom, `beta`: the PDB bfactor of the atom, `coord`: the coordinates of the atom as {x y z}, `vel`: the velocity of the atom as {vx vy vz}
  <*segid*>: Segment ID of target segment, residue, or atom.
  <*resid*>: Residue ID of target residue or atom.
  <*atomname*>: Name of target atom.
  <*value*>: Value to be assigned.
  Context: After structure has been generated or read from file.

- `writepdb` <*file name*>
  Purpose: Writes PDB file containing coordinates. Atom order is identical to PSF file generated by writepsf (unless structure has been changed). The O field is set to 1 for atoms with known coordinates, 0 for atoms with guessed coordinates, and -1 for atoms with no coordinate data available (coordinates are set to 0 for these atoms).
  Arguments: <*file name*>: PDB file to be written.
  Context: After structure and coordinates are complete.

- `writenamdbin` <*file name*> [`velnamdbin`] [*velocity file name*]
  Purpose: Writes NAMD binary file containing coordinates. Atom order is identical to PSF file generated by writepsf (unless structure has been changed). Coordinates are set to 0 for atoms with no coordinate data.
  Arguments: <*file name*>: NAMD binary file to be written.
  `velnamdbin`: Also write velocities to NAMD binary file.
  <*velocity file name*>: NAMD binary velocity file to be written.
  Context: After structure and coordinates are complete.

8 Example of a Session Log

The command "writepsf" prints a simple session log as "REMARKS" at the beginning of the PSF file. The log contains information about applied patches and used topology files which not stored in the standard records of PSF files. These informations are also available after a PSF file was read by command "readpsf". Here'a a simple axample:

```
PSF

       1 !NTITLE
 REMARKS original generated structure x-plor psf file
 REMARKS 4 patches were applied to the molecule.
 REMARKS topology 1LOV_autopsf-temp.top
 REMARKS segment P1 { first NTER; last CTER; auto angles dihedrals }
 REMARKS segment O1 { first NONE; last NONE; auto none  }
 REMARKS segment W1 { first NONE; last NONE; auto none  }
 REMARKS defaultpatch NTER P1:1
 REMARKS defaultpatch CTER P1:104
```

```
 REMARKS patch DISU P1:10  P1:2
 REMARKS patch DISU P1:103  P1:6

    1704 !NATOM
       1 P1   1    ALA  N    NH3   -0.300000       14.0070           0
...
```

All patches that were applied explicitely using the “patch” command are listed following the keyword “patch”, but the patches that result from default patching like the first and last patches of a segment are marked as “defaultpatch”. Further the segment based patching rules are listed along with the angle/dihedral autogeneration rules.