# MMB 2.14

UPPSALA UNIVERSITET

# Reference guide

December 30, 2013

# Table of Contents

# 1     What's new?

In release 2.14 I improved a few features and added several others.

First, *Physics where you want it* was not including any monoAtoms. Now all atoms created with this command are automatically added to the physics zone. You can read in ion positions from a PDB file also, you just need to be careful about atom and residue names in that file. For example, the atom name for the magnesium ion should be `Mg+2`, and its residue name should be `MG`.

I have introduced a new residue type, a disulphide bridging cysteine. You can specify it using the single letter code 'X'. You can also infer it from the input PDB file, if you name the residue in that file CYX, rather than CYS (the latter will be interpreted as a normal, non-bridged cysteine). Thus it is compatible with both the `protein` and `loadSequencesFromPdb` commands. To actually bridge the sulphurs, you would use the new `addRingClosingBond` command. An example of this is supplied, see the new `commands.disulphide-bonded-cysteines.dat` . There is a new figure to explain this, in the *Mobilizers and constraints* chapter; the usage is explained in the *Forces* chapter.

There were some silly limitations to how one could mix user defined variables, numbers, and +/- operators, which have now been removed. You can now mix these more or less as you wish, though absolute residue numbers still have to be leftmost in the expression. You can also use a leading '- ' in a residue number, and this will correctly be interpreted as part of the residue number rather than an operator.

Have you ever simply wanted to impose a certain secondary structure in a certain region of your model? Say, make a helix-turn-helix into a single continuous helix? Well, with the new `setPhiPsiAngles` command you can do just that. It sets the phi, psi, and peptide dihedral angles to the defaults for `Alpha`, `ParallelBeta` and `AntiParallelBeta` secondary structures, overriding whatever values these dihedrals may have taken from the input structure file.

The new `psiPhiMobility` command is used for specifying the bond mobility for the bonds connecting the N to CA, and the CA to C on the protein backbone, for a given stretch of residues. It is equivalent to issuing the `singleBondMobility` command for the two mentioned bonds, for each residue in the range.

Quite importantly, we have brought back the `matchGapped` macro. When issued, it sets `guessCoordinates` to `True`, meaning that if you have missing atoms or even entire residues in your input PDB file, it will guess positions for the missing atoms, and still perfectly match the remaining (non-missing) atoms. Even the default `matchFast` behavior guesses side chain positions, but with `matchGapped` you can leave out backbone atoms. You will probably get back bond geometries at the boundary between the known and missing backbone atoms, which you can correct using the `singleBondMobility .. Free` command.

We also have improved the `mobilizeInterfaces` command, which lets you set the bond mobility for residues at a certain interface. Firstly, it now bases the definition of "interface" on all atoms, not just Cα and C3* atoms. Secondly, it is now much more economical. Lastly, you can define the interface differently .. specifying one or more chains that form a complex, and then asking for all interfaces with *other* chains.

Lastly, we have a new method of adding residues to the *physics zone*. Just set the new parameter `physicsRadius` to some value greater than zero. It will look for all "flexible" atoms – defined as atoms in a mobilizedBody of mass < 40, and then include all residues within `physicsRadius` of these, in the *physics zone*.

*For any published work which uses MMB, please cite one or more of the following:*

Turning limited experimental information intio 3D models of RNA, by Samuel C Flores and Russ B Altman, *RNA* 16(9):1769-78 (2010).

Predicting RNA structure by multiple template homology modeling, by Samuel C. Flores, Yaqi Wan, Rick Russell, and Russ B. Altman (2010) *Proceedings of the Pacific Symposium on Biocomputing.*

Fast flexible modeling of RNA structure using internal coordinates, by Samuel C. Flores, Michael Sherman, Chris Bruns, Peter Eastman, Russ Altman (2011) *Transactions in Computational Biology and Bioinformatics* 8(5): 1247-57.

# 2 Biopolymers and monoAtoms

In this Appendix, we describe how to instantiate biopolymers (RNA, protein), as well as single atoms such as counterions. Note that the number of biopolymers and series of single atoms is limited by the number of characters available as chain identifiers.

## 2.1 Biopolymer sequences and first residue numbers

MMB can instantiate RNA chains using the following syntax:

```
RNA <chain ID> <first residue #>  <sequence in single letter code>
```

The RNA sequence uses the single letter code (A,U,G,C). Similarly, you can instantiate DNA chains like this:

```
DNA <chain ID> <first residue #>  <sequence in single letter code>
```

The DNA sequence uses the single letter code (A,T,G,C). You can instantiate a protein chain as:

```
protein <chain ID> <first residue #> <sequence in single letter
code>
```

The protein chains use the 20 canonical amino acid alphabet for specifying the sequence.

Note that as of release 2.12, you can make the chain ID as long as you wish. This means that you are not limited to the 144 printable ASCII characters. In the output PDB file, the following tag will indicate the long chain ID:

```
REMARK-SimTK-long-ChainID mySuperLongNameForThisChain
```

.. where of course "`mySuperLongNameForThisChain`" will be replaced by whatever chain ID you specified. This will be followed by the corresponding ATOM records. The ATOM records will have a blank (" ") in  column 22 (the chain ID column).  Finally, after all the ATOM records for that chain have been printed, there will be another tag, like this:

```
REMARK-SimTK-long-ChainID
```

.. followed by nothing.  This "turns off" the long chain ID specification.  The next chain may be a normal chain (single-character chain ID in column 22) or there may be another chain with a long ID bracketed by "`REMARK-SimTK-long-ChainID`" tags as before.

There is one more way to instantiate sequences, which works for protein, RNA, and DNA. You can issue the command:

```
loadSequencesFromPdb
```

And MMB will go to your input structure file (last.??.pdb) and look for RNA and protein chains.  It will extract the chain ID's, residue numbers, insertion codes, and residue types from there.  It will also match the internal coordinates to the Cartesian coordinates it finds there, as usual.  You will then be able to issue commands that involve residues in those chains, as before.  Please note that you CANNOT use this command with long chain IDs, it just won't work.  If you have long chain IDs, just instantiate the chains explicitly.

In addition to removing the need for you to specify these chains manually, this command also has the advantage of handling insertion codes and gaps in the numbering.  You will be able to append an insertion code to the right of the residue number in *any* command, e.g. `constrainToGround A 32B` (where B is an insertion code).

The residue numbers and insertion codes do need to be increasing from the top to the bottom of the input structure file, though.  Before using this command, you should clean up the input structure file, removing anything that is not RNA or protein – including DNA, water, ions, or other molecules.

Whether you use `loadSequencesFromPdb` or specify the sequences manually, it is possible to use the '+' operator to increment or decrement a residue ID by some number of residues. For instance,

```
constrainToGround A 32B+2
constrainToGround A 32B+-3
```

will constrain residues two residues to the C-terminus and three residues to the N-terminus of 32B. Do not insert any spaces between the the '+' operator and either of its arguments. You can use the '+' operator with *any* command that takes residue numbers as an argument.

## 2.2 monoAtoms

The `monoAtoms` command specifies single atoms (e.g. monatomic ions) The syntax follows:

```
monoAtoms <chain ID> <first residue #> <# of atoms> <name of atom>
```

Currently only the following atom names are supported:

```
Mg+2, Cl-, Na+, K+, Li+, Ca+2, Cs+, Rb+
```

The single atoms created with this command support the `atomSpring`, `atomTether`, `springToGround`, `constrainToGround,` and `constraint` commands, just like the biopolymers. They do not support the `mobilizer` command.

monoAtoms are automatically added to the physics zone.

You can read the positions of monoAtoms from your input PDB file. You have to be careful with atom and residue names in that file though. For example, for a magnesium ion, the atom name should be "Mg+2", while the residue name should be "MG". Note that this is different from the PDB convention, in which both atom and residue name are "MG".

# 3   A word about units

I am making a special, very short chapter on units. In MMB 2.10 and earlier, some forces such as atomSpring, springToGround, atomTether, etc. took Å as the unit for dead lengths and ground locations. For consistency, we are going back to nm for the length unit. This is because internally all the math is done in nm, kJ/mol, ps, and daltons (g/mol).  This implies that spring constants are in kJ/mol/nm$^2$. For example, if you want to make a spring which in Amber99 units (Å, kcal/mol, ps) would be 310 kcal/mol/Å$^2$, the equivalent spring in our choice of units would be 129790.8 kJ/mol/nm$^2$.

Please note that if you have any dead lengths or ground locations  in your MMB 2.10 or earlier script which you are using with MMB 2.11, you will need to manually change them from Å to nm.

# 4   Forces

In this Appendix, we describe options for using the `baseInteraction,` `aromatic` two-residue forces, the `atomSpring,` `atomTether,` and `springToGround forces`, and the `contact` steric forces.  Note that since forces are additive, there is no hard limit on how many forces can exist in the system or even acting on a single residue, base, or atom.

## 4.1  baseInteraction

The syntax for this command is:

```
baseInteraction  <chain identifier for first residue>
          <residue number for first residue>
          <interacting edge for first residue>
          <chain identifier for second residue>
          <residue number for second residue>
          <interacting edge for second residue>
          <glycosidic bond orientation>
```

The following combinations of first base pairing edge, second base pairing edge, and glycosidic bond orientation are permitted:

```
WatsonCrick WatsonCrick Cis
WatsonCrick WatsonCrick Trans

WatsonCrick Hoogsteen Cis
WatsonCrick Hoogsteen Trans

WatsonCrick SugarEdge Cis
WatsonCrick SugarEdge Trans

Hoogsteen Hoogsteen Cis
Hoogsteen Hoogsteen Trans

Hoogsteen SugarEdge Cis
Hoogsteen SugarEdge Trans

SugarEdge SugarEdge Cis
SugarEdge SugarEdge Trans

WatsonCrick Bifurcated Cis
Stacking3 Stacking5 Cis
Stacking5 Stacking5 Trans
```

```
Stacking3 Stacking3 Trans
HelicalStackingA3 HelicalStackingA5 Cis
Superimpose Superimpose Cis
```

You might notice that some of these are actually not in the Leontis and Westhof classification. These are explained below:

- Stacking* simply specifies a stacking interaction between consecutive residues on a chain. The numbers indicate which face is interacting on each base. For example:

  `baseInteraction A 120 Stacking3 A 121 Stacking5 Cis`

  Means that the face of base 120 which would ordinarily point towards the 3' end of the strand in a helix, will be stacked on the face of base 121 which would ordinarily point to the 5' end of the helix.

- HelicalStacking* works the same as Stacking, but adds the offset appropriate for consecutive bases in a helix. HelicalStackingA3/HelicalStackingA5 is automatically applied to all consecutive bases in helices, unless you specify `setHelicalStacking FALSE`. MMB assumes an A-form helix exists whenever it finds three consecutively numbered RNA residues on a single strand Watson-Crick base paired with three consecutively numbered residues on the same or another single RNA strand. If you want to generate a helix where this is not the case, you should manually apply HelicalStackingA3 / HelicalStackingA5  interactions.

## 4.2  nucleicAcidDuplex

This command generates WatsonCrick/WatsonCrick/Cis interactions between two specified segments on the same or different RNA chains. It is a shortcut for manually specifying each such interaction for every pair of canonically interacting residues in the duplex. The syntax is:

```
nucleicAcidDuplex      <chain identifier A>
                       <first residue on A>
                       <last residue on A>
                       <chain identifier B>
                       <first residue on B>
                       <last residue on B>
```

Recalling that the duplex is antiparallel, we require that:

```
(first residue on A) < (last residue on A)
and
(first residue on B) > (last residue on B)
```

For example:

```
nucleicAcidDuplex A 1 3 A 10 8
```

Makes the segments between residues 1 and 3 (inclusive) and between 10 and 8 (inclusive) into two halves of a duplex, by applying a base pairing interaction between 1 and 10, 2 and 9, and 3 and 8.

## 4.3  Units

Before we describe the various variants of user-applied springs, let's clarify the units. MMB and molmodel use nm, kJ/mol, ps, and daltons (a.k.a. atomic mass units = u, or g/mol).  The Amber99 force field, on the other hand, uses nm, kcal/mol, and ps, and u. The user

## 4.4  atomSpring

The `atomSpring` command creates a linear spring connecting two atoms.   Two optional parameters  (square braces []) specify the dead length and spring force constant.

```
 atomSpring   <first chain ID>
              <first residue number>
              <first atom name>
              <second chain ID>
              <second residue number>
              <second atom name>
              [<dead length>
              [<spring constant>]]
```

## 4.5  atomTether

The `atomTether` command, as the name implies, applies no force if the distance between atoms is less than a certain  `<dead length>`,  and applies an attractive force with

Hookean `<spring constant>` when the distance exceeds the former. Default values for the last two parameters are 0.0 and 3.0, respectively, as they are for `atomSpring`. Make `<spring constant>` large for a strict "dog leash" or small for a permissive restraint.

```
atomTether   <first chain ID>
             <first residue number>
             <first atom name>
             <second chain ID>
             <second residue number>
             <second atom name>
             [<dead length>
             [<spring constant>]]
```

## 4.6 springToGround

The `springToGround` command creates a linear spring connecting a specified atom and a specified location in Ground. Two optional parameters (square braces []) specify the dead length and spring force constant.

```
springToGround   <atom chain ID>
                 <atom residue number>
                 <atom name>
                 <X location in Ground>
                 <Y location in Ground>
                 <Z location in Ground>
                 [<dead length>]
```

## 4.7 threading

The `threading` command applies `atomSpring`'s between pairs of atoms with the same name, on corresponding residues. The result is that the atoms of a given stretch of residues in a given chain 1 are aligned to the like-named atoms of a corresponding stretch in a second chain 2. For release 2.6.2 and higher, this command works with any biopolymer; its predecessor only worked for protein. The optional parameter (square braces []) specifies the spring force constant.

```
threading    <chain 1 ID>
                 <first residue number of chain 1>
```

```
                        <last  residue number of chain 1>
                        <chain 2 ID>
                        <first residue number of chain 2>
                        <last  residue number of chain 2>
                        [<spring constant (default = 3.0)>]
```

If you are trying to align just the backbone of a protein, you can use the `proteinBackboneThreading` command, which has the same syntax as above, but only applies springs between corresponding N, CA, and C atoms.

## 4.8 contact

You can also apply space-filling Contact spheres to a range of residues using the `contact` command. (The idea is similar to that of the parameters `addSelectedAtoms` and `addAllHeavyAtomSterics`)

```
contact    <contact type>
           <chain identifier>
           <residue number for first residue>
           <residue number for last residue>
```

The first residue should be lower numbered than the second, and both residues should be on the same chain. You can also issue:

```
contact    <contact type>
           <chain identifier>
```

And the contact spheres will be applied to every residue on the specified chain.

There are two kinds of permitted values of `contact type`. In the fixed type, the atom identities are hard-coded and can't be modified by the user, but the contact sphere radii and stiffness (both of which are the same for all atoms regardless of atom name) correspond to the `excludedVolumeRadius` and `excludedVolumeStiffness` parameters which are set in the MMB input file (e.g. commands.dat). These include:

`AllAtomSterics`                  : Puts one sphere on each atom of the chain, except for the end caps on proteins (when used).

`AllHeavyAtomSterics` : Puts one sphere on each atom of the chain EXCEPT hydroges, and again except for the end caps on proteins.

`RNABackboneSterics` : Puts one sphere on each of the following atoms: P, O5*, C5*, C4*, C3*, and O3*. An error will result from attempting to apply this to proteins, as anytime when you attempt to put sterics on an atom which doesn't exist on a given residue.

The second type of sterics are user configurable, in the parameter file (e.g. parameters.csv). Here the user can choose on which atoms to put the spheres, with a maximum of four atoms. The radii and stiffness can be controlled separately for each atom name. A different choice of zero to four atom names can be chosen for each residue type (4 residue types for RNA, 20 for protein). The user can add as many steric schemes to the parameter file as he/she wishes; as supplied the `parameters.csv` file has two: `SelectedAtoms` and `ProteinBackboneSterics`. For the first one, the parameters look like:

| RECORD | A | SelectedAtoms | SelectedAtoms | X | P | C4* | N9 |
|--------|---|---------------|---------------|---|---|-----|----|
| RECORD | C | SelectedAtoms | SelectedAtoms | X | P | C4* | N1 |
| RECORD | G | SelectedAtoms | SelectedAtoms | X | P | C4* | N9 |
| RECORD | U | SelectedAtoms | SelectedAtoms | X | P | C4* | N1 |

The second column is the residue type, and columns 7,8, and 9 are the atom names. Note that the glycosidic nitrogen is named differently for purines vs. pyrimidines. Subsequent columns give the sphere radii, stiffnesses, and information to identify these as `contact parameter` entries. Parameters become available for use immediately upon being entered in the parameter file, much as for MD force field parameter files.

It is also possible to apply a specified steric scheme to all residues within a certain distance of a specified residue. The distance is measured by between representative atoms – Cα for proteins, C4* for RNA and DNA. The syntax is:

`applyContactsWithin <radius (nm)> <contact scheme> <chain> <residue>`

## 4.9 Restraining to ground

Much as residues can be constrained to each other (see next chapter), any residue of any chain can also be restrained to ground, meaning that a force can be applied to pull all six

translational-rotational degrees of freedom to an equilibrium position and orientation in Ground:

```
restrainToGround <chain ID> <residue number>
```

Keep in mind that unlike a constraint, a restraint acts as a spring and thus allows some displacement with respect to ground. Any displacement at the end of a stage is carried over to the next stage, potentially leading to a "creeping" effect. Two parameters which are relevant to this command are `restrainingForceConstant` and `restrainingTorqueConstant`. These set the translational and angular restitution force constants.

## 4.10 Density based force field

As explained in the tutorial, MMB's density based force field is formulated following Klaus Schulten's MDFF as follows:

Where $i$ is the atom index, $m_i$ is the mass of atom $i$,          is the electronic density at the nuclear position of atom $i$, A is a user-adjusted scaling factor, and    is the gradient operator. Accordingly,    is the density-derived force vector applied to atom $i$. This is computed for and applied to every atom $i$ in the system.

To turn on the density based force field on or off, you just need to specify which chains you want to be subjected to such forces. For instance:

```
fitToDensity
```

Specifes that all chains in the system should be fitted to the map. If you only want certain chains to be fitted, with the remaining chains not subjected to these forces, just specify each chain to be fitted like this:

```
fitToDensity <chain ID>
```

Lastly, if you only want certain stretches of residues to be fitted, you can issue:

```
fitToDensity <chain ID> <start residue> <end residue>
```

.. and only the residues starting at `<start residue>` and ending at `<end residue>` of chain `<chain ID>` will be fitted.

Your density map must be in XPLOR format. To specify the location of the density map, file, use:

```
densityFileName <density file name>
```

The scaling factor (A in the equation above) defaults to unity, but you can set it to any floating point number (including negative numbers) as follows:

```
 densityForceConstant  <scale factor>
```

## 4.11 Physics where you want it

*Physics where you want it*, introduced in release 2.4, allows you to turn on the all-atoms force field only for certain regions of your system, referred to as the *physics zone*.

To specify a range of residues to be added to the physics zone, use:

```
includeResidues <chain ID> <first residue in range> <last residue in range>
```

Sometimes it will be convenient to include all residues within a certain radius of a specified residue. For this you would use:

```
includeAllResiduesWithin <distance> <chain ID> <residue number>
```

Note that `includeResiduesWithin` is an alias for `includeAllResiduesWithin`. The `distance` (in Å) is measured between key atoms, CA for protein and C4* for RNA and DNA.

You can also simply set `physicsRadius` to a value > 0. If this is set, all residues within `physicsRadius` of the "flexible"atoms will be included in the *physics zone*. "Flexible" atoms are defined as atoms belonging to a mobilized body of mass < 40. This is technically a parameter rather than a command, so is listed in that section separately. The syntax is just:

```
physicsRadius <radius>
```

Default behavior is for all atoms to be subjected to the non-bonded force field terms. If that is what you want, just don't specify either of the above commands.

Lastly, we have found that small chemical groups such as methyl or alcohol can spin out of control in the absence of viscous forces, leading to small time steps and excessive computational expense. To deal with this, you can scale the inertia of such small groups with:

```
smallGroupInertiaMultiplier <inertia scale factor>
```

Any nonnegative floating point number can be used here; we suggest 11.0.

## 4.12  Potential rescaling with the "Scrubber"

In Flores and Altman (RNA 2010) we found that kinetic trapping occurs often in computational RNA folding, as it does experimentally. To get out of these traps we created the *scrubber*. Potential rescaling refers to cyclically varying forces. In MMB, we use a rectangular waveform. For a fraction of the time (1 - `dutyCycle`) all forces (including baseInteraction's, sterics, Amber99 force field, springToGround's, etc.) will be turned off. Then for the remainder of the period (`dutyCycle`) these forces will be turned back on. The length of the period is set with the `scrubberPeriod` parameter (in ps, as always).

```
dutyCycle <"on" fraction>
scrubberPeriod <potential rescaling period, in ps>
```

This is used in some of the MMB tutorial examples.

## 4.13  addRingClosingBond

This add a ring closing bond. It only creates bonds between atoms in the same chain. Use it like this:

```
addRingClosingBond      <chainID>      <residueID1>      <atomName1>
<bondCenterName1>  <residueID2>  <atomName2> <bondCenterName2>
```

bondCenter's are named as e.g. `bond1, bond2, .. bondN`. For example, in a disulphide bridge (bonding atoms `SG`), you need to specify `bond1,` since `bond1` is bonding to `CB`.

# 5    Mobilizers and constraints

In this Appendix we describe `mobilizer` commands, which define or modify the internal coordinate topology of the molecule as well as `constraint` commands, which  add constraint equations that reduce the degrees of freedom of the system.

It is important to keep in mind the crucial difference between these two in Internal Coordinate Mechanics.  A `mobilizer` command can reduce or increase the number of bodies that exist in a system; in the former case you will always save computer time.  On the other hand a `constraint` command adds constraint equations which must then be solved; while the net effect depends on masses and forces, computational cost typically increases. Mobilizers control bond mobilities, which here can be `Free`, `Torsion`, or `Rigid`. `Free` means that the bond can change its length, angle, and dihedral. `Torsion` means it can change only its dihedral angle. `Rigid` means it has no degrees of freedom.

One must also avoid overconstraining the system.  For example, if two rigid molecules are already `Weld`'ed (see below) to each other, do not put additional constraints on this pair of molecules, even if they are nominally applied to different residues.  While this is easy to keep track of for two bodies, watch out for more insidious ways of overconstraining.  For example, if A is `Weld`'ed to  B, and B is `Weld`'ed to C, do not then `Weld` C to A.

## 5.1  mobilizer

The `mobilizer`  keyword is used for specifying the bond mobilities for a stretch of residues. This command is overloaded.  The first variant has the following syntax:

```
mobilizer  <bond mobility>
           <chain identifier>
           <first residue number>
           <last residue number>
```

The first residue should be lower numbered than the second, and both residues should be on the same chain. Bond mobility can be set to `Free, Torsion, Rigid, or Default.` The

"`Default`" bond mobility is special, as we'll explain in a moment. Don't forget you can use the keywords `FirstResidue`  or `LastResidue`, or do arithmetic on the residue numbers using the "+" operator, as described earlier.

You can also simply say:

```
mobilizer   <bond mobility>
            <chain identifier>
```

... and this will set ALL residues in chain `<chain identifier>` to `<bond mobility>`.

Lastly, you can say:

```
mobilizer   <bond mobility>
```

... and this will set all residues in ALL chains to `<bond mobility>`.

## 5.2  applyMobilizersWithin

The `applyMobilizersWithin`  command is used to specifying the bond mobilities for all biopolymer residues within a certain radius of a specified residue.  It has the following syntax:

```
applyMobilizersWithin <bond mobility>
                      <radius>
                      <chain identifier>
                      <residue ID>
```

The `radius` is measured between representative atoms (Cα for protein, C4* for nucleic acids) and (like always) in Å.  The acceptable values of `<bond mobility>`  are as listed above.

## 5.3  mobilizeInterfaces

The `mobilizeInterfaces` command is used to specifying the bond mobilities for all biopolymer residues within a certain distance of all interfaces of a given biopolymer chain or chains. The syntax is:

```
mobilizeInterfaces    <interface depth>
                      <bond mobility>
                      <chain 1> [<chain 2> [<chain 3> [...etc]]]
```

The `<interface depth>` is measured as the minimum distance between atoms on different chains across an interface. Note that this counts over *all* atoms, not just the Cα or C3*. We are now using OpenMM's neighborlisting for this, which is pretty economical even if you aren't set up to use the GPU. `<bond mobility>` is that desired at the interface – `Rigid, Torsion, Free,` or `Default. <chain 1,2, etc>` is the list of chains forming a complex, whose interfaces with the *rest* of the system you are interested in.  For example, say you have a complex of chains A, B, and E.  If you issue:

```
mobilizeInterfaces 0.6 Default A B
```

Then all residues at the interface between the complex AB, and chain E, to a depth of 0.6 nm, will get bond mobility `Default`.  Note that this will do nothing in particular to the interface between A and B! In this case you could just as easily have issued:

```
mobilizeInterfaces 0.6 Default E
```

## 5.4  singleBondMobility

The `singleBondMobility` command is used for specifying the bond mobility for a single bond:

```
singleBondMobility    <chain identifier for first atom>
           <residue number for first atom>
           <atom name for first atom>
           <bond mobility>
```

```
         <chain identifier for second atom >
         <residue number for second atom >
         <atom name for second atom>
```

The two atoms should be covalently bonded to each other, of course.

## 5.5  psiPhiMobility

`psiPhiMobility` is used for specifying the bond mobility for the bonds connecting the N to CA, and the CA to C on the protein backbone, along a given stretch of residues. It is equivalent to issuing the `singleBondMobility` command for the two mentioned bonds, for each residue in the range.

```
psiPhiMobility   <chain ID>
                 <residue number for first residue in range>
                 <residue number for last residue in range>
                 <bond mobility (Free, Torsion, or Rigid)>
```

You can also skip the residue numbers:

```
psiPhiMobility   <chain ID>
                 <bond mobility (Free, Torsion, or Rigid)>
```

.. and this will apply mobilizers to whole chain, from first to last residue. Lastly, you can skip the chain ID:
```
psiPhiMobility   <bond mobility (Free, Torsion, or Rigid)>
```

.. and this will apply mobilizers to whole chain, for every protein chain in the system.

`psiPhiMobility` is simply a shortcut for a bunch of `singleBondMobility` commands, and so works in exactly the same way as the latter. This means that it is applied late – see the section "**Order of application of mobilizers**."

## 5.6  Default mobilizers

It is important to understand what is the default setting for the mobilizers in your system.

The default bond mobility leaves most bonds set to `Torsion,` but there are also some `Rigid` bonds, depending on the residue type and atoms it connects. For instance, the bond mobilities for an RNA residue look like this:
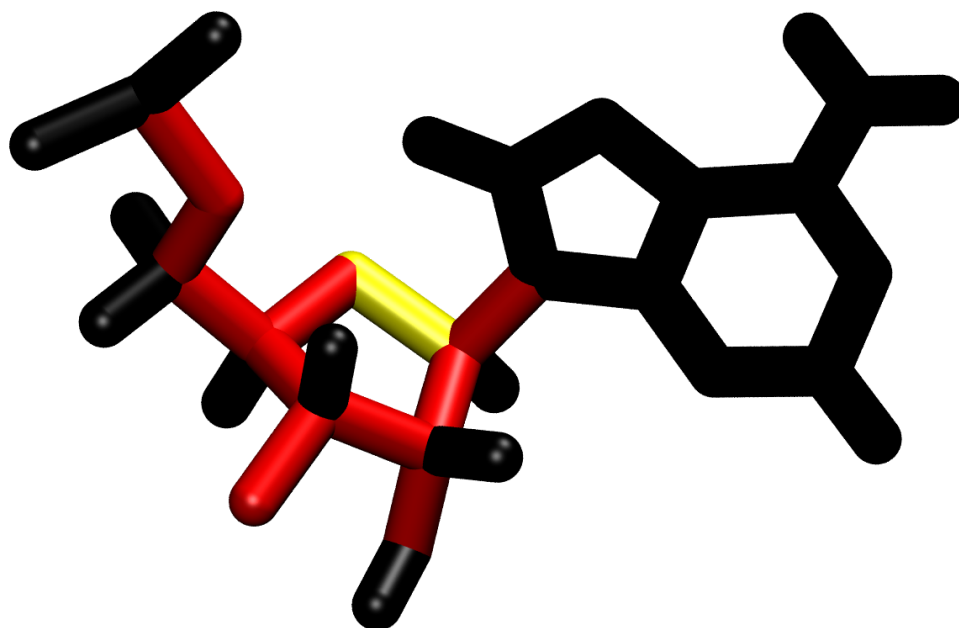
**Figure 1 : Default bond mobilities for an RNA residue**

**Black:** `Rigid`; **Red:** `Torsion`; **Yellow:** `Free.`

Similarly for a protein residue, most bonds are also `Torsion`. There are certain bonds and groups that are set to `Rigid`:

All peptide bonds (C-N)
All covalent bonds between hydrogens and heavy atoms
Proline N-Cα (This may be changed to `Torsion`)
Guanidinium group (Arginine CZ-NH1, CZ-NH1, CZ-NE)
Amide groups (Asparagine Cγ-Nδ2, Glutamine Cδ-NE2)
Cyclic groups in Tryptophan, Histidine, Phenylalanine, Tyrosine. EXCEPT that ring closing bonds are special (nonexistent topologically, subject to bonded MD force field terms):
      Tryptophan Cδ2-Cγ, CZ3-CH2
      Histidine, Tyrosine and Phenylalanine Cδ2-Cγ
      Proline Cδ-N

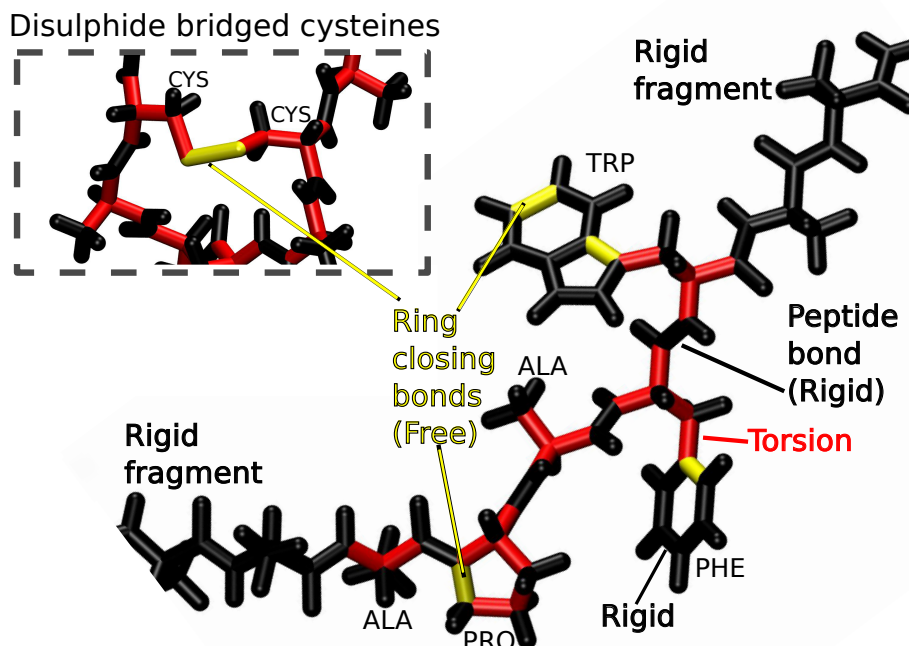**Figure 2 : Bond mobilities for proteins. By default, most bonds have Torsion bond mobility (red). Peptide bonds are Rigid, as are guanidinium and amide groups. Hydrogens and double-bonded oxygens are also connected with Rigid bonds. The user can create a ring closing bond which will have Free bond mobility. Ring closing bonds are special in that they do not topologically connect atoms, only apply bonded forces (bond stretch, angle bend, dihedral) – recall that closed cycles are not desirable in internal coordinates. Cyclic groups of Histidine, Tryptophan, Phenylalanine, and Tyrosine (but not Proline) are connected with Rigid bonds and so form a single body; the ring closing bonds do not change this. Peptide bonds are Rigid. All of these default bond mobilities can be overridden by the user. For example, the segments on either end of the chain have here been made Rigid. Lastly, the user can also create a disulphide bridge between cysteines using a ring closing bond (inset).**

## 5.7 Order of application of mobilizers

In order to get the desired result out of MMB, you should understand the order in which these commands are applied. They go like this:

1. `mobilizer (for Rigid, Free, and Torsion)`
2. `applyMobilizersWithin (for Rigid, Free, and Torsion)`
3. `mobilizeInterfaces(for Rigid, Free, and Torsion)`
4. `mobilizer, applyMobilizersWithin, and mobilizeInterfaces(for Default)`
5. `singleBondMobility`

 A common mistake is to forget that before any commands are applied, all chains have a default bond mobility, as described above.  Note also that the "`Default`" bond mobility isn't actively applied to residues – instead when you specify this, all *other* modifications to the residue bond mobility are removed, so it is simply never changed from its original bond mobility.

Here is a simple example:

```
protein A 1 AAAAAA
mobilizer Rigid A 1 6
mobilizer Default A 3 4
```

Results in two `Rigid` stretches (1 to 2 and 5 to 6) – the output looks something like this:

```
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:44
Mobilizer stretch 0 BondMobility = Rigid
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:45 chain=
A from residue 1 to 2
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:44
Mobilizer stretch 1 BondMobility = Rigid
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:45 chain=
A from residue 5 to 6
```

## 5.8 constraint

The `constraint`  command is used for specifying constraints to weld two residues together:

```
constraint      <chain identifier for first residue>
                <residue number for first residue>
                Weld
                <chain identifier for second residue>
                <residue number for second residue>
```

The two welded residues can be on different chains; in fact either or both residues can be in RNA or protein chains.  The weld is applied on C3* atoms of RNA residues and on C atom s of protein residues.  There is no preference for residue number ordering.

You can also specify which atoms you want welded, as follows:

```
constraint      <first atom chain identifier>
                <first atom residue>
                <first atom name>
                Weld
                <second atom chain identifier>
                <second atom residue number>
                <second atom name>
```

Lastly, you can weld to ground, either specifying the atom to be welded or using the default:

```
constraint       <chain identifier>
                 <residue number >
                 <atom name>
                 Weld Ground
```

or:

```
constraint       <chain identifier>
                 <residue number >
                 Weld Ground
```

## 5.9  Constraining to ground

Just as residues can be constrained to each other, any residue of any chain can also be constrained (rigidly attached) to ground:

```
constrainToGround <chain ID> <residue number>
```

See *Appendix: Parameters* for an explanation of the `constraintTolerance` parameter, relevant to this command.

Much more efficient are a couple of variants of this command.  For example:

```
constrainToGround
```

(with no parameters) attaches each chain to ground using a Weld rather than a Free mobilizer.  Thus rather than granting 6 DOFs and then removing them with constrain equations, the DOFs never exist to begin with.

Similiarly, you can choose the mobilizer type (`Free` vs. `Weld`) for all chains by issuing:

```
rootMobilizer <"Free" | "Weld">
```

Or, you can choose the mobilizer type for a specific chain by issuing:

```
rootMobilizer <Chain> <"Free" | "Weld">
```

## 5.10  Constraining rigid segments to each other or to ground

In the antibody design example (see Tutorial), we have a protein which has two rigid segments and one flexible segment.  To prevent the two rigid segments from moving with respect to ground, we welded them to ground. Alternatively, maybe we could have welded the rigid segments to each other, so the protein as a whole could move with respect to its binding partner (or ground, for that matter). Sometimes you may want a chain to have many rigid segments, all welded either to ground or to a specified residue.  MMB has a convenient command for this.

If you want to weld all rigid segments of all chains to ground, just issue:

```
constrainChainRigidSegments
```

If you want to weld the rigid segments of a specified chain to ground, issue:

```
constrainChainRigidSegments <chain ID> Ground
```

where `<chain ID>` refers to the chain in question.

Lastly, if you want to weld the rigid segments of a specified chain to a specified residue (on the same chain), issue:

```
constrainChainRigidSegments <chain ID> <residue ID>
```

In this latter case, all the rigid segments in chain `<chain ID>` will be welded to the same residue `<residue ID>`.  This means that the rigid segments will move together, allowing rigid body motions of the entire chain.  If you want several chains to move together, just add a `constraint` command.

# 6    Direct structure editing

In this chapter we talk about editing structure *directly*, meaning modifying internal or Cartesian coordinates of structures, adding or removing atoms, etc. For now this chapter is quite short, other features will be documented soon.

## 6.1  Initial displacement

This command simply displaces the designated chain by a given Cartesian vector, with respect to its position in the input structure file:

```
initialDisplacement <chain> <X> <Y> <Z>
```

## 6.2  Imposing protein secondary structure

Have you ever simply wanted to impose a certain secondary structure in a certain region of your model?  Say, make a helix-turn-helix into a single continuous helix?  Well, with the new `setPhiPsiAngles` command you can do just that. It sets the phi, psi, and peptide dihedral angles to the defaults for `Alpha`, `ParallelBeta` and `AntiParallelBeta` secondary structures, overriding whatever values these dihedrals may have taken from the input structure file.  The syntax is:

```
setPhiPsiAngles <chain ID> <start residue> <end residue> < Alpha |
ParallelBeta | AntiParallelBeta>
```

These are applied *after* structure matching. If the command is issued multiple times, the dihedral angles are set  in the order the commands were issued.

## 6.3  Introducing a substitution mutation

A common structure editing operation is to substitute one residue type for another. The syntax is:

```
substituteResidue <chain> <residue ID> <new residue type>
```

The residue at position `<residue ID>` will simply be replaced by one of type `<new residue type>`. If you provided an input structure file, MMB will match all the internal coordinates it can based on identical atom names, and use default values for the remaining internal coordinates. For example if your mutate alanine to valine, it will match the Cα and Cβ positions, but will choose a position for Cγ1 and Cγ2 based on default bond lengths, angles, and dihedrals. Note that substitutions in general will need to be equilibrated to ensure reasonable interatomic contacts.

# 7 Global parameters

This appendix, describes global parameters available to users. It does not cover *commands* such as `baseInteraction`, `aromatic`, `contact`, `mobilizer`, and `constraint`. The simplest difference between a *parameter* and a *command* is the following. A *command* can be issued an unbounded number of times, subject only to memory and computer time limitations. The major caveat is that in the case of `constraint` commands, one must not overconstrain the system. In contrast a *parameter* can only be set once (at least for a given stage); if a parameter is set multiple times for a given stage, only the last value of that parameter will be used. A listing of all user-configurable global parameters and their current values is printed at the beginning of every stage of an MMB run. Some additional parameters are available but rarely used or not recommended; contact the author with questions on these.

This chapter does not describe *staged* parameters. These are parameters for which not only the *value,* but also the *stage* at which they first take effect is specified, for example `temperature` and `dutyCycle`.

| | | | |
|---|---|---|---|
| addAllAtomSterics | Bool | FALSE | Add steric contact spheres to all atoms. This is more expensive and more prone to kinetic trapping than addSelectedAtoms. |
| addAllHeavyAtomSterics | Bool | FALSE | Add steric contact spheres to all atoms EXCEPT hydrogens. |
| checkSatisfied | Bool | FALSE | At each reporting interval, list all the baseInteraction's and determine which were satisfied. |
| constraintTolerance | float | 0.05 | This determines the tolerance of the Weld constraint. If Weld'ed pieces are moving relative to each other, reduce this number. |
| cutoffRadius | float | 0.1 | This is the range of the MMB potential. See our Multiple-template homology modeling paper. |
| densityFileName | String | | Name of file for fitting based on electron density, in .xplor format. If you need to convert from some other format, we recommend using mapman (e.g. rave_osx for mac). Instructions are here: http://xray.bmc.uu.se/usf/mapman_man.html#S10 |
| densityForceConstant | Float | 1 | Scale factor for the density based forces |
| firstStage | int | 1 | Stage at which simulation should begin. |
| globalAmberImproperTorsionScaleFactor | float | 0 | |

| globalBondBendScaleFactor | float | 1.0 | These eight parameters set scaling factors for terms in the Amber99 potential.  Most default to 0 for economy. |
|---|---|---|---|
| globalBondStretchScaleFactor | float | 1.0 | |
| globalBondTorsionScaleFactor | float | 0 | |
| globalCoulombScaleFactor | float | 0 | |
| globalGbsaScaleFactor | float | 0 | |
| globalVdwScaleFactor | float | 0 | |
| initialSeparation | float | 20.0 | Sets the separation between chains at stage 1, or whenever readPreviousFrameFile = false. |
| integratorAccuracy | int | 0.001 | Integrator tolerance, applies for variable step size time integrators. |
| integratorStepSize | int | 0.001 | Step size in ps, for fixed step size integrators. |
| integratorType | string | Verlet | Choose between Verlet, RungeKuttaMerson |
| integratorUseFixedStepSize | Bool | FALSE | self explanatory |
| lastStage | int | 1 | Stage at which simulation will end |
| leontisWesthofInFileName | string | ./parameters.csv | MMB parameter file |
| loadTinkerParameterFile | Bool | FALSE | If FALSE, uses hard-wired Tinker parameters.  If 1, reads parameters from tinkerParameterFileName |
| numReportingIntervals *alias* maxReportingIntervals | int | 100 | Number of reporting intervals per stage. |
| nastGlobalBondTorsionScaleFactor | int | 10 | Scale factor for NAST torsional potential |
| physicsRadius | Float | 0 | If this is set to a value > 0, all residues within physicsRadius of any "flexible" atoms will be added to the physics zone. "flexible" atoms are defined as those belonging to a mobilized body of mass < 40. |
| randomizeInitialVelocities | Bool | FALSE | Adds a random velocity to each body at the beginning of the simulation stage.  Note that if you are have any non-interacting bodies (e.g. free ions with charges turned off) you may wish to apply initial velocities, otherwise the Nose-Hoover thermostat will leave them in their zero kinetic energy state. |
| reportingInterval | float | 1.0 | Duration of reporting intervals, in ps. |
| removeRigidBodyMomentum | Bool | FALSE | When True, periodically sets overall translational and rotational momentum to zero. |
| rigidifyFormedHelices | Bool | FALSE | |
| scrubberPeriod | float | 4 | Duration of one cycle of potential rescaling (ON time + OFF time) in ps. |
| safeParameters | Bool | TRUE | When TRUE, checks for syntax errors as well as some potentially dangerous parameter values. |
| setForceAndStericScrubber | Bool | FALSE | No longer user configurable.  When dutyCycle < 1.0, this is automatically set to TRUE.  It turns ALL forces (including baseInteraction's, sterics, Amber99 force field, springToGround's, etc.) off for (dutyCycle -1) fraction of each scrubberPeriod. |
| setHelicalStacking | Bool | TRUE | if TRUE, identifies three consecutive WatsonCrick/WatsonCrick/Cis base pairs as a helix and applies HelicalStackingA3/HelicalStackingA5/Cis baseInteraction's between the consecutive residues on each strand. |
| setTemperature | Bool | TRUE | Turns on thermostat. |
| thermostatType | string | | Choices are NoseHoover and VelocityRescaling |

| | | | |
|---|---|---|---|
| tinkerParameterFileName | string | | Name of the tinker-formatted parameter file.  Only needed if the tinker force field is turned on. |
| baseInteractionForceMultiplier *alias* twoTransformForceMultiplier *alias* forceMultiplier | float | 100 | Scale factor applied to all baseInteraction and aromatic forces.  100 or 1000 is recommended to speed up modeling. |
| useFixedStepSize | Bool | FALSE | Specifies fixed-step-size time integration. |

# 8 Macros

This appendix describes macros available to users. These macros set parameters on the user's behalf. These are provided in cases where the corresponding commands might be confusing to the user, or simply not under user control.

| matchFast | This sets matchExact TRUE, matchIdealized FALSE, matchOptimize FALSE, and guessCoordinates FALSE. It is very economical. It is the default behavior, so usually there is no need to call this. |
|---|---|
| matchGapped | This sets matchExact TRUE, matchIdealized TRUE, matchOptimize TRUE, and guessCoordinates TRUE. It guesses positions for any missing atoms. matchFast does this just fine for side chains, but matchGapped can handle missing backbone atoms. There may be unphysical bond geometries at the boundaries between known and missing backbone atoms, but there are ways to heal this. |
| setDefaultMDParameters | Equivalent to issuing:<br>globalBondTorsionScaleFactor     1.0<br>globalAmberImproperTorsionScaleFactor  1.0<br>globalBondBendScaleFactor        1.0<br>globalBondStretchScaleFactor     1.0<br>globalBondTorsionScaleFactor     1.0<br>globalCoulombScaleFactor         1.0<br>globalVdwScaleFactor             1.0<br>globalAmberImproperTorsionScaleFactor  1.0 |

# 9    User defined variables, parameter arithmetic, and conditional blocks

In this Appendix, we describe how to define numerical variables, and various ways to specify sections of the input file which are to be read or ignored at certain stages.

## 9.1  Comment marker

The comment marker is `#`, e.g.:

```
# Don't read this, it's just a comment
```

## 9.2  User defined variables

User variables are defined with the following syntax:

```
@<variable-name> <float or integer value>
```

The variable `@<variable-name>` can then be used wherever a literal integer or float is expected.  If a float is assigned to the variable, and the variable is later used where an integer is expected, MMB will return an error. The definition of the variable should precede its first use in the input file.  For example:

```
#declare @myStage variable and set to 3
@myIntervals 3
```

```
# now use it where a number (in this case an integer) is expected:
numReportingIntervals @myIntervals
```

Don't use any punctuation or whitespace in `<variable-name>`.
Don't try to set `firstStage` or `lastStage` with a user variable.

## 9.3 Parameter arithmetic

User variables are pretty handy, and start to make the command file more like a programming language. In the same vein, MMB allows the '+' and '–' operators. This means that any integer or floating-point (double-precision) parameter value can be set using a combination of literals, user variables, and the above operators. There is no limit to the number of operators and operands. Here are a couple of examples:

```
@DUMMY 40
numReportingIntervals  @DUMMY+10-@DUMMY
@MYFLOAT 0.35
reportingInterval  4+@MYFLOAT-0
```

This is equivalent, of course, to:

```
numReportingIntervals  10
reportingInterval  4.35
```

Don't use any whitespace or additional punctuation (such as parentheses, commas, etc.) in an arithmetic expression. Note also that residue ID's are special (they're not integers), and their '+' operator follows different rules (see Chapter 2).

## 9.4 Conditional blocks

In many cases we will want to issue different commands and make different choices of parameter values at different stages of a job. For this purpose we can enclose a block of the input file in a conditional block, which is opened as follows:

| | |
|---|---|
| `readFromStage <stage-number>` | Read only if the current stage is equal to |
| or GREATER than `<stage-number>`. | |
| `readToStage` | Read only if the current stage is equal to |
| or LESS than `<stage-number>`. | |
| `readAtStage` | Read only if the current stage is EQUAL |
| to `<stage-number>`. | |
| `readExceptAtStage` | Read only if the current stage is NOT |
| EQUAL to `<stage-number>`. | |

The commands and parameters to be conditionally read follow, and the end of the block is indicated with a `readBlockEnd` statement, e.g.:

```
# start conditional block:
readAtStage 3
# read the following lines only at stage 3:
sequence C CCUAAGGCAAACGCUAUGG
firstResidueNumber C 146
baseInteraction A 2658 WatsonCrick A 2663 WatsonCrick Cis
contact C 146 SelectedAtoms C 164
# end conditional block:
readBlockEnd
# continue with the rest of the input file
```