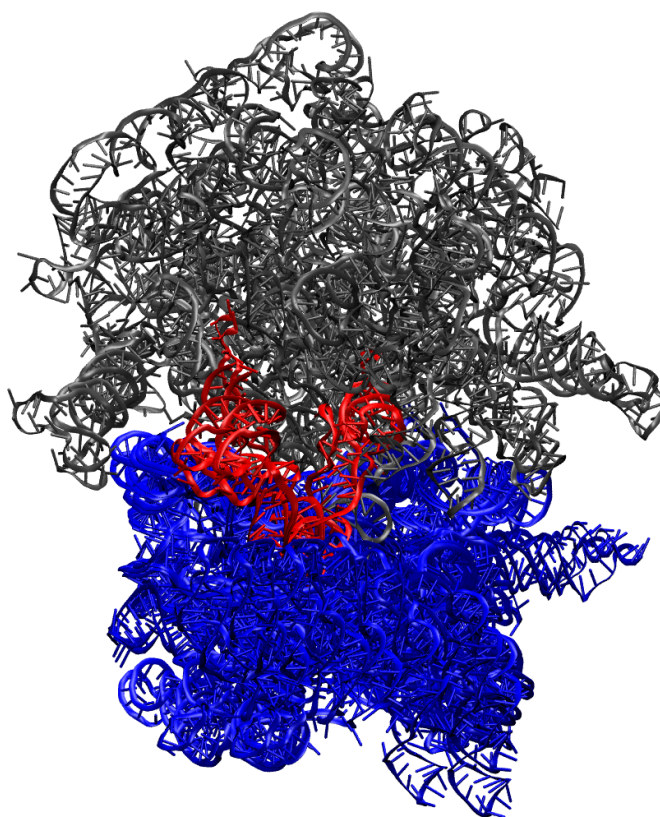




UPPSALA  
UNIVERSITET

# MMB 2.10



## Reference guide

August 14, 2012

---



## **Copyright and Permission Notice**

Copyright (c) 2011 Samuel Flores  
Contributors: Joy P. Ku

Permission is hereby granted, free of charge, to any person obtaining a copy of this document (the "Document"), to use and copy the Document without modification for academic teaching and research purposes, subject to the following conditions:

This copyright and permission notice shall be included in all copies or substantial portions of the Document.

THE DOCUMENT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS, CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENT OR THE USE OR OTHER DEALINGS IN THE DOCUMENT.







# Table of Contents

<b>1</b>	<b>WHAT'S NEW?</b>	<b>7</b>
<b>2</b>	<b>BIOPOLYMERS AND MONOATOMS</b>	<b>13</b>
2.1	Biopolymer sequences and first residue numbers	13
2.2	monoAtoms	14
<b>3</b>	<b>FORCES</b>	<b>16</b>
3.1	baseInteraction	16
3.2	nucleicAcidDuplex	17
3.3	atomSpring	18
3.4	atomTether	18
3.5	springToGround	19
3.6	threading	19
3.7	contact	20
3.8	Restraining to ground	21
3.9	Density based force field	22
3.10	Physics where you want it	23
<b>4</b>	<b>MOBILIZERS AND CONSTRAINTS</b>	<b>24</b>
4.1	mobilizer	25
4.2	applyMobilizersWithin	25
4.3	mobilizeInterfaces	26
4.4	singleBondMobility	26
4.5	Default mobilizers	27
4.6	Order of application of mobilizers	28
4.7	constraint	28
4.8	Constraining to ground	29
<b>5</b>	<b>GLOBAL PARAMETERS</b>	<b>29</b>
<b>6</b>	<b>MACROS</b>	<b>32</b>
<b>7</b>	<b>USER DEFINED VARIABLES, PARAMETER ARITHMETIC, AND CONDITIONAL BLOCKS</b>	<b>34</b>

7.1	Comment marker .....	34
7.2	User defined variables .....	34
7.3	Parameter arithmetic.....	35
7.4	Conditional blocks .....	35



# 1 What's new?

In release 2.10 I addressed a longstanding issue – the time requirement of reading in a PDB structure and matching an internal coordinate model to its Cartesian nuclear coordinates. Whereas previously there were three macros (`matchFast`, `matchGapped`, `matchGappedNoHeal`) to set the matching parameters, now there is only one behavior (`matchFast`) and thus no need to specify anything at all. Whereas previously `matchFast` required a special double-precision coordinate file to accurately match long chains, now an ordinary PDB can be read perfectly. The only downside is that now you can't automatically guess coordinates for loops bridging gaps that appear in your input structure file (but this can be done in other, non-automatic ways). Note that this doesn't mean you can't deal with gaps at all – in fact if you use `loadSequencesFromPdb` you can jump over such gaps, though an unphysically long bond may appear connecting the biopolymer atoms across the gap. I have found the gap-spanning feature useful in the past, so I hope to resuscitate it before long. In any event those of you who work on the ribosome will find the faster structure loading particularly appealing.

A couple of bugs were found that may have affected 2.9.2 users. First, the scrubber didn't always behave correctly. I fixed this bug and also removed the `setForceAndStericScrubber` parameter – now if you set `dutyCycle` to anything less than unity MMB assumes you want to turn the scrubber on. Second, the `applyContactsWithin` command wasn't working properly. This command wasn't in the Reference guide then (an omission I now correct), so it's unlikely most users would have known about it.

I also reorganized things a bit internally (made the `contact`, `mobilizer`, and `basePair` containers all children of a new class, `ResidueContainer`), to make the program more extensible, though this doesn't affect the user experience.

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

Turning limited experimental information into 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*.









## 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.

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.

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 '+' 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.

## 3 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.

### 3.1 `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.

## 3.2 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.

### 3.3 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>]]
```

### 3.4 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>]]
```

### 3.5 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>]
```

### 3.6 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.

### 3.7 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 hydrogens, 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 – C $\alpha$  for proteins, C4\* for RNA and DNA. The syntax is:

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

### 3.8 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.

### 3.9 Density based force field

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

$$\vec{f}_i = A \cdot m_i \cdot \vec{\nabla} D(x_i, y_i, z_i)$$

Where  $i$  is the atom index,  $m_i$  is the mass of atom  $i$ ,  $D(x, y, z)$  is the electronic density at the nuclear position of atom  $i$ ,  $A$  is a user-adjusted scaling factor, and  $\vec{\nabla}$  is the gradient operator. Accordingly,  $\vec{f}_i$  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, set:

```
densityMapActivate    <True | False>
```

By default, if `densityMapActivate` is `True`, all chains in the system will be subject to density based forces. If you only want a limited number of chains to be fitted, with the remaining chains not subjected to these forces, just specify each chain to be fitted with a command like this:

```
fitToDensity <chain ID>
```

... one for each chain, of course.



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>
```

### 3.10 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 turn this feature on or off, use:

```
physicsWhereYouWantIt <True | False>
```

This parameter defaults to `False`, meaning the force field terms are applied to all atoms. Set to `True` to restrict to a user specified set.

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

```
includeAllNonBondAtomsInResidues <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>
```

The distance (in Å) is measured between key atoms, CA for protein and C4\* for RNA and DNA.

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 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 welded (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 welded to B, and B is welded to C, do not then weld C to A.

## 4.1 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`. 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>`.

## 4.2 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 ( $C\alpha$  for protein,  $C4^*$  for nucleic acids) and (like always) in Å. The acceptable values of `<bond mobility>` are as listed above.

### 4.3 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. The syntax is:

```
mobilizeInterfaces   <interface depth>
                     <bond mobility>
                     <chain identifier>
```

The `<interface depth>` is measured as the minimum distance between representative atoms ( $C\alpha$  for protein,  $C4^*$  for nucleic acids) on different chains across an interface. `<chain identifier>` is the chain whose interfaces you are interested in.

As an example, if you have a trimer of chains A,B, and C, and issue :

```
mobilizeInterfaces 6.0 Torsion C
```

Then all residues at all interfaces between chain C and A, and between C and B (but not between A and B), to a depth of 6.0Å, will get bond mobility `Torsion`.

### 4.4 singleBondMobility

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

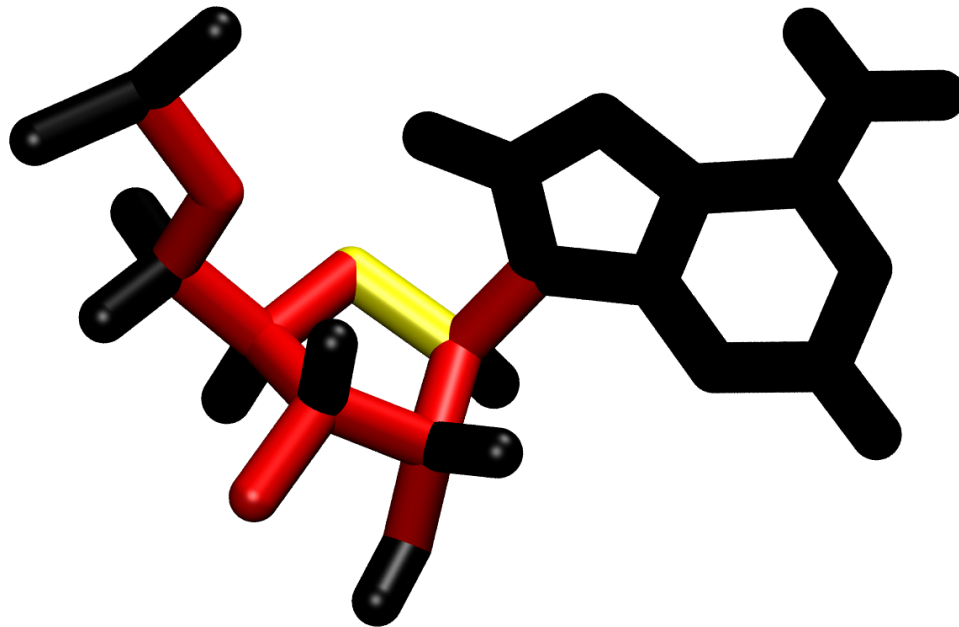
```
singleBondMobility    <chain identifier for first atom>
                      <residue number for first atom>
                      <atom name for first atom>
                      <bond mobility>
                      <chain identifier for second atom >
                      <residue number for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

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

## 4.5 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`, but the peptide bond is `Rigid`, as are some cyclic side chain groups.

## 4.6 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`
2. `applyMobilizersWithin`
3. `mobilizeInterfaces`
4. `singleBondMobility`

A common mistake is to forget that before any commands are applied, all chains have a default bond mobility, as described above.

## 4.7 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
```

## 4.8 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.

# 5 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`.

<code>addAllAtomSterics</code>	Bool	FALSE	Add steric contact spheres to all atoms. This is more expensive and more prone to kinetic trapping than <code>addSelectedAtoms</code> .
<code>addAllHeavyAtomSterics</code>	Bool	FALSE	Add steric contact spheres to all atoms EXCEPT hydrogens.
<code>checkSatisfied</code>	Bool	FALSE	At each reporting interval, list all the <code>baseInteraction</code> 's and determine which were satisfied.
<code>constrainRigidSegments</code>	Bool	FALSE	When TRUE, adds a <code>constrainToGround</code> command for each rigid fragment in each chain in the system. Useful for equilibrating small regions in multiple points throughout a complex without affecting parts of the complex distant to those small regions.
<code>constraintTolerance</code>	float	0.05	This determines the tolerance of the Weld constraint. If Weld'ed pieces are moving relative to each other, reduce this number.
<code>cutoffRadius</code>	float	0.1	This is the range of the MMB potential. See our Multiple-template homology modeling paper.
<code>densityFileName</code>	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. <code>rave_osx</code> for mac). Instructions are here: <a href="http://xray.bmc.uu.se/usf/mapman_man.html#S10">http://xray.bmc.uu.se/usf/mapman_man.html#S10</a>
<code>densityForceConstant</code>	Float	1	Scale factor for the density based forces
<code>densityMapActivate</code>	bool	False	When True, turns on density based forces
<code>firstStage</code>	int	1	Stage at which simulation should begin.
<code>globalAmberImproperTorsionScaleFactor</code>	float	0	



globalBondBendScaleFactor	float	1.0	These eight parameters set scaling factors for terms in the Amber 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	int	100	Number of reporting intervals per stage.
<i>alias</i> maxReportingIntervals			
nastGlobalBondTorsionScaleFactor	int	10	Scale factor for NAST torsional potential
randomizeInitialVelocities	Bool	FALSE	Adds a random velocity to each body at the beginning of the simulation stage. Note that if you 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	0.2	Duration of reporting intervals, in ps.
rigidifyFormedHelices	int	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, Amber 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	float	100	Scale factor applied to all baseInteraction and aromatic forces. 100 or 1000 is recommended to speed up modeling.
<i>alias</i> twoTransformForceMultiplier			
<i>alias</i> forceMultiplier			
useFixedStepSize	Bool	FALSE	Specifies fixed-step-size time integration.

## **6    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	See the chapter, "Matching to the input structure file." This sets matchExact TRUE, matchIdealized FALSE, and matchOptimize FALSE. It is very economical. It is now the ONLY matching parameter setting macro, and only reinstates the defaults, so is effectively vestigial
setDefaultMDParameters	Equivalent to issuing: globalBondTorsionScaleFactor 1.0 globalAmberImproperTorsionScaleFactor 1.0 globalBondBendScaleFactor 1.0 globalBondStretchScaleFactor 1.0 globalBondTorsionScaleFactor 1.0 globalCoulombScaleFactor 1.0 globalVdwScaleFactor 1.0 globalAmberImproperTorsionScaleFactor 1.0

# 7 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.

## 7.1 Comment marker

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

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

## 7.2 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.

### 7.3 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).

### 7.4 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>.
```

<code>readToStage</code>	Read only if the current stage is equal to
<code>or LESS than &lt;stage-number&gt;.</code>	
<code>readAtStage</code>	Read only if the current stage is EQUAL
<code>to &lt;stage-number&gt;.</code>	
<code>readExceptAtStage</code>	Read only if the current stage is NOT
<code>EQUAL to &lt;stage-number&gt;.</code>	

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
```