

The AMBER/INTERFACE Front End

(Version 2.0)

INTERFACE, the AMBER/INTERFACE front end
and all accompanying documentation are

(c) 1991,1992,1993,1997 David A. Pearlman

All Rights Reserved

NOTICE!

INTERFACE, the AMBER/INTERFACE front end
and all accompanying documentation are

Copyright (c) 1991,1992,1993,1997
David A. Pearlman
All Rights Reserved.

Except as permitted under the United States Copyright Act of 1976, or by the licensing agreement under which this manual and related software were provided, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database retrieval system, without the prior written permission of the publisher.

The software described in this document is provided pursuant to a license agreement, and may only be used, copied, or otherwise disclosed in accordance with the terms of the license.

This document contains proprietary information made available only under contract to specified parties. It may not be copied, disclosed, or transferred to others without the express written permission of the author.

Table of Contents

Copyright Notice.....	1
Table of Contents	2
1. Using the INTERFACE/AMBER Front End.....	4
I) What is INTERFACE?.....	4
II) What is the AMBER/INTERFACE front end?.....	4
III) What are the advantages of the AMBER/INTERFACE front end?	4
IV) How do I invoke INTERFACE?.....	5
V) How Can I Generate a Manual for the AMBER/INTERFACE Front End?..	6
VI) How is the AMBER/INTERFACE front end installed?.....	7
VII) How do I write a new IPS or customize an existing one?.....	11
VIII) Where can I learn more about commands and constructs allowed in the command input file?.....	12
IX) Commands recognized in the command input file	12
2. Intrinsic Features of the Input Language.....	13
Introduction	13
Intrinsic Commands and Control Constructs	14
ASSIGN	14
DO	15
DO WHILE	16
EXIT.....	17
GETTEMPLATE	18
GOTO.....	19
HELP	20
IF	21
MANUAL	23
REDIRECT	24
STOP	26
Other Elements of the Interface Language.....	27
In-Line Variable Substitution (<>).....	27
Continuation lines (\).....	29
Comment lines (!).....	30
Intrinsic Functions Provided in Interface	31
3. Guide to the AMBER/INTERFACE Front End.....	33
Index to Command Specificity.....	34
Command Definitions	35
archive	35
belly.....	36
cap	37
change.....	38
clearall	40
cnst	41
com	44
constraints.....	45
decoupled	47
delete	48
dielectric	49

direct.....	50
DMW.....	51
dna.....	53
echo.....	54
entropy.....	55
Ewald.....	56
EXTCOM.....	58
external.....	59
fast_water.....	60
force_field.....	61
free_energy_components.....	62
generate.....	64
getarchive.....	65
gibbs.....	66
group.....	67
imaging.....	69
info.....	70
intragroup.....	71
logcommands.....	72
logenergy.....	73
md.....	74
minimize.....	75
minmd.....	76
mixing.....	77
nameset.....	78
nmr.....	79
pairlist.....	83
parallel.....	85
PBC.....	86
pdbgen.....	88
peacs.....	89
polarization.....	90
position_restraints.....	91
putarchive.....	92
random_seed.....	93
read.....	94
report.....	95
restart.....	96
restraints.....	97
run.....	101
sander.....	104
scratchname.....	105
sequential.....	106
set.....	107
shake.....	108
slow_growth.....	109
steps.....	110
temp.....	111

time_average	114
time_limit	116
timestep	117
title.....	118
triplet	119
windows	120
Appendices	121
I) Sample Script To Perform Minimization	121
II) Script to perform MD with a restraint.....	123
III) Sample Script for MD/NOE Refinement.....	125
IV) Sample Script Performing Torsional Driving.....	127
V) Sample Script to Run Gibbs.....	132
VI) Creating Customized Redirect Files	134
VII) Extracting PDB files from a coordinate archive file	136
VIII) Modifying the Standard Amber/Interface	138

Chapter 1

Using the INTERFACE/AMBER Front End

I) What is INTERFACE?

INTERFACE is a program which allows one to design a command language. A program is written in the IPS (Interface Programming Script) language, which defines the elements of the command language. In common usage, this command language is defined so that mnemonic commands with user-friendly syntaxes are translated into the confusing input required for a pre-existing program (such as “numbers in columns”). Typically, most end users never look at the IPS script used to generate the command language. They merely reap the fruits of a much improved interface to their favorite programs. The IPS language contains numerous sophisticated elements which make the generation of this robust and effective command language straightforward.

II) What is the AMBER/INTERFACE front end?

IPS scripts have been written which replace the old “numbers in columns” input for the MINMD, SANDER, and GIBBS modules of AMBER with easy-to-use command oriented languages. The commands in the new language are parsed, and the required old-style AMBER input is generated from them. But the end user never has to see or know about this old-style input.

III) What are the advantages of the AMBER/INTERFACE front end?

In addition to the obvious advantage of mnemonic language-oriented input over confusing numbers-in-columns, INTERFACE offers several more important advantages:

- 1) The commands defined in the IPS are supplemented by various intrinsic functions of the Interface language. These include DO-loops, IF and GOTO constructs, variable assignment statements, numerous intrinsic functions (e.g. SIN, COS, EXP, etc.), and in-line variable substitution. These can be interspersed among the other commands as desired. Thus, one can run numerous AMBER jobs from one script, changing only desired elements of the simulation for each run. For example, one might define a torsional

Using the INTERFACE/AMBER Front End

restraint whose target value changed on each iteration of a DO-loop. This makes it easy to perform “torsional driving”.

- 2) Commands can, with few exceptions¹, be specified in any desired order. This allows the user to arrange the commands in a fashion which makes most sense. It also makes errors less likely.

For a specific command, options can also be specified in any order.

- 3) Comments can be freely interspersed with the commands in the command-input file. This makes documentation of the input much more informative.
- 4) The input for the various programs has been rationalized. This means that the input for the three programs (MINMD, SANDER, GIBBS) is identical for those elements which have the same effect in each program. This reduces considerably the time required to learn all three programs. Commands which are unique to a particular program are read but ignored if specified as input to the other programs.
- 5) Along with commands which generate the input required for AMBER, the IPS scripts also define commands which submit the programs with the required arguments. The actual computer-dependent commands required to submit the program on the host machine are contained in the IPS script, need only be set up once (when installing the IPS), and are hidden from the end user. This means that the identical command script can be used to both set up and run the programs on ANY computer, regardless of the actual operating system used on the host computer. And since INTERFACE is written in standard FORTRAN, it can be implemented on nearly any computer with very little effort.
- 6) The command language defined with the IPS can easily be modified or extended by the user. For example, the user can easily define aliases for any command names. And additional command definitions can be added to the IPS definition with minimal effort.
- 7) INTERFACE performs considerable syntactical error-checking, and additional error checking is performed in the IPS script itself. This helps avoid simple mistakes in the command input which would result in erroneous behavior. Any command which is not recognized will cause an informative message to be reported to the user.

IV) How do I invoke INTERFACE?

¹The notable exceptions are: 1) The commands MINMD, SANDER, and GIBBS, which indicate which program will be run. This affects the internal processing of subsequent commands. One of these commands should precede other commands defining the input; 2) The command GENERATE, which causes the intermediate “numbers-in-columns” file to be created, and which should follow all other commands defining the input; and 3) GROUP commands which define group input, and should closely follow the command for which the group input is being defined.

Using the INTERFACE/AMBER Front End

Once Interface and the IPS template files have been installed (see below), you run interface as you would any program. You will immediately obtain the prompt:

```
INTERFACE (?=HELP) >
```

to which you should reply

```
IN = command_file # TEMPLATE = template_file
```

where `command_file` is the file you have created which contains the commands related to running AMBER, and `template_file` contains the command definitions. The `template_file` is derived from the IPS scripts provided with this distribution, and defines all the commands needed for MINMD, SANDER, and GIBBS. The end-user is not responsible for creating the IPS scripts. They are included in the AMBER/INTERFACE distribution.

Once you enter your reply, your `command_file` will be parsed and executed.

On Unix and VAX/VMS machines, you can optionally specify the IN and TEMPLATE files on the command line. The allowed syntax is

```
interface -i(nput) command_file -t(emplate) template_file
```

Each flag can be abbreviated to the letter not enclosed in parentheses. If no flags are given on the command line, you will obtain the INTERFACE (?=HELP) > prompt, just as on other types of computers.

V) How Can I Generate a Manual for the AMBER/INTERFACE Front End?

In subsequent sections of this manual, the commands recognized in the AMBER/INTERFACE front end (defined in the supplied IPS files) are described. If, however, these scripts are modified at your site, or if you would like an additional copy of the manual (formatted for a line printer), you can easily generate one using INTERFACE. This is made possible by the help utility commands (HELP, MANUAL) recognized by INTERFACE.

The INTERFACE program allows one to query for help on various defined commands. When you define a command in the IPS, you can also specify text which describes the command, and which will be reported to the user if HELP is requested on that particular command. For every command of the AMBER/INTERFACE front end, help text has been defined.

To generate a manual for the interface, initiate the INTERFACE program as described in the previous section, but instead of specifying `IN = command_file`, specify `IN = STDIN`. This will cause the command input to be read from the standard input device (usually the terminal). Next, issue the command

Using the INTERFACE/AMBER Front End

```
MANUAL / OUTPUT = filename
```

where filename is the name of the file to contain the desired manual. You can subsequently print the specified manual.

Alternatively, once you are running INTERFACE, you can request on-line HELP on any desired topic. For example, if instead of issuing the above MANUAL command you specified the command:

```
HELP TEMP/*
```

help on the “TEMP” command, and all associated options, would be echoed to the terminal.

To exit the AMBER/INTERFACE front end, you can issue the command “STOP”. In general, you can also cleanly exit interface by issuing the “end of file” command appropriate for the computer you are using. For example, on a Unix computer, you would issue ^d (control-d). On a VMS VAX, you would issued ^z (control-z). Of course, you can also issue whatever command is used to interrupt program execution.

A copy of the manual appears later in this document.

VI) How is the AMBER/INTERFACE front end installed?

The instructions in this section need only be carried out to install the AMBER/INTERFACE front end. If you are an end-user ready to run a pre-installed version of the front-end, you can skip to the next section (VII). If you are installing the program package, and have been supplied a pre-compiled version of INTERFACE, you should skip the first subsection below (“Installing the INTERFACE program”) and carry out the instructions in the second and third subsections (starting with “Installing the auxiliary programs”).

Installation of the AMBER/INTERFACE front end is a four stage process. First, you must create the interface directory tree by copying the files supplied on tape to the computer which serve as the host. Second, the INTERFACE program itself must be customized (if necessary) for the host computer, and then compiled. Third, the auxiliary programs used with the interface must be compiled. Finally, the supplied IPS scripts must be customized for the host computer. These are then run through INTERFACE to set up the template files which act as the instruction set read by INTERFACE to implement the AMBER/INTERFACE front end.

In the standard distribution Unix distribution, two interactive scripts are supplied in the top level Interface directory. These scripts, install_int and install_ambint (or install_int.com and install_ambint.com under VMS), query the user for all required information and then do all the required compilations.

Creating the required AMBER-INTERFACE directory tree

Using the INTERFACE/AMBER Front End

The AMBER/INTERFACE distribution consists of a top level directory (e.g. “interface”) and a number of subdirectories. You should transfer the supplied tape to the target host computer, making sure you supply the appropriate commands to retain the directory structure on the tape during the transfer. On a Unix computer, a command such as tar xvf will typically work.

Installing the INTERFACE program:using the supplied script

Move to the top level directory of the tree in which you placed the AMBER/INTERFACE files (see above). If you are installing the standard Unix distribution, the script “install_int” has been supplied in this directory. This file will query the user for the appropriate information and install the interface program. Issue the command

```
source install_int
```

You will be queried for a few pieces of information. The appropriate system dependent changes will be performed. Subsequently, you simply need issue the command “make” to do the actual compilation.

Note: You should read points 4-6 of the following section before executing the script.

Installing the INTERFACE program on computers for which no script is available

(Points 1-3 of this section are provided to help in installing INTERFACE on computers for which no standard compilation script has been provided. If you have installed the program as described in the previous paragraph, you can skip to points 4-6

With two necessary exceptions, INTERFACE is written entirely in ANSI standard FORTRAN. This means that porting INTERFACE to any computer with a FORTRAN compiler should be quite simple. Basically, all you need to do is the following:

- 1) Copy the files which comprise the INTERFACE program into a chosen directory.
- 2) Generate an appropriate version of extrnl.f. This file contains the machine-dependent commands used to implement the “EXTERNAL” command in the IPS language. The purpose of the EXTERNAL(string) command is to pass “string” to the host operating system for execution. The standard distribution of INTERFACE includes three versions of extrnl.f: extrnl_vms.f, extrnl_unx.f, and extrnl_ucs.f. These are the versions of extrnl.f appropriate for VMS/VAX, UNIX, and Cray/Unicos computers, respectively. Simply copy the appropriate version to extrnl.f. If you are trying to install INTERFACE on a non-Unix, non-VMS computer, you will need to edit one of the pre-existing extrnl.f files and add the appropriate calls for the host computer, as described in any of these files.

If the host computer does not support the types of calls required to effect the EXTERNAL command, you can provide a “dummy” extrnl.f file which does nothing but return (or return with a warning message). In this case, INTERFACE will still run, but the EXTERNAL command will have no effect.

Using the INTERFACE/AMBER Front End

- 3) Use the appropriate version of unxfn.f. On computers running Unix (and Unicos), this file contains the machine-dependent commands required to allow specification of input files on the same line with the invocation of INTERFACE. On these computers, copy unxfn_unx.f to unxfn.f. On all other types of machines, copy unxfn_dmy.f. to unxfn.f This is a dummy routine. (The motivated user may wish to create a version of unxfn appropriate for their own favorite non-Unix computer. See unxfn.f for details of what is expected from this subroutine).
- 4) You may wish to increase or decrease the default memory allocations. See the file “storage.inc”, which contains the dimension declarations for most of the storage arrays used by INTERFACE. These are clearly marked in this file. There should be no need to change the memory allocations for running INTERFACE with the AMBER front-end IPS files.
- 5) If you are compiling on a Unix (or Unicos) machine, you need only modify the supplied Makefile to provide the appropriate compilation flags, then issue the command “Make”..If you are compiling on a VAX/VMS machine, you can simply run the command file compile.com: @compile.com. If you are attempting to compile on any other type of computer, you may have to edit Makefile or compile.com to create a script of the appropriate type. Or you may wish to concatenate all the source code files into one large file before compilation. If so, be sure to make interface.f the first file in the concatenation.

Some computers (such as the IBM risc-station series) require you to issue special compilation flags if you are going to use character strings longer than a pre-set limit. As shipped, INTERFACE uses fairly long character strings and you may need to use such a compilation flag on these machines. The compiler will usually issue a warning message if any action needs to be taken in this regard.

- 6) While INTERFACE is written in ANSI FORTRAN, and should compile correctly on any machine with a FORTRAN compiler, the quirks and bugs inherent in some compilers may require small modifications to the code. If you encounter such a case, please forward your experiences to the address at the top of this document.

Creating the appropriate “template” files with the supplied script

Once the INTERFACE program has been compiled, you need to generate the required template file from the supplied IPS scripts. INTERFACE reads the IPS script and from it produces a second file (the “template” file) which is reformatted and indexed to speed program performance. It is this template file, and not the IPS itself, which is used by INTERFACE when interpreting commands in the input file. Once the template file has been created, the user need only specify the this template file when starting INTERFACE. The IPS itself is not needed again unless/until it is changed.

Move to the top level directory of the tree in which you placed the AMBER/INTERFACE files (see above). If you are installing the standard Unix distribution, the script “install_ambint” has been supplied in this directory. This file will query the user for the

Using the INTERFACE/AMBER Front End

appropriate information and install the AMBER/INTERFACE template file. Issue the command

```
source install_ambint
```

You will be queried for a few pieces of information. The appropriate system dependent changes will be performed, and the AMBER/INTERFACE script will be installed. In addition, several auxiliary programs which accompany the AMBER/INTERFACE front end (see the following section) will be compiled and installed.

Once the install_ambint script has completed, you are done installing the AMBER/INTERFACE front end.

Creating the appropriate "template" files when no appropriate script is available

(If you have used the install_ambint script to create the "template" files and install the auxiliary files, you are done with the installation. You may now skip to section VII).

There are two components of the supplied IPS which need to be customized for the host computer. The first change involves changing pointers to the actual location (directory) of files on the host computer. The second change involves the scripts for commands which invoke the EXTERNAL command to allow commands to be executed by the host computer. Obviously, the commands passed must be appropriate the particular system.

1) Edit the file amber.def. Near the top of this file, you will find the following commands

```
REDIRECT = "/usr/amberint/def/" /DEFAULT

ASSIGN PROGRAM_USE_MINMD      = "/usr/amber4/exe/minmd"
ASSIGN PROGRAM_USE_SANDER     = "/usr/amber4/exe/sander"
ASSIGN PROGRAM_USE_GIBBS      = "/usr/amber4/exe/gibbs4"
ASSIGN PROGRAM_USE_PDBGEN     = "/usr/amber4/exe/pdbgen"

ASSIGN UTILITY_DIR = "/usr/amberint/exe/"
```

In the first of these statements, the string between the quotes must be replaced by the full directory specification for the directory in which you have placed all the IPS scripts supplied with this distribution. Note that this specification must include any required trailing punctuation (e.g. the final slash required to separate the directory from the filename in a Unix specification, or the colon required to separate the directory from the filename on a VAX/VMS computer).

In the following four ASSIGN statements, you should replace the strings between quotes with the full specifications (directory plus filename) of the executable files corresponding to MINMD, SANDER, GIBBS, and PDBGEN.

Finally, in the ASSIGN UTILITY_DIR statement, the string between quotes should be replaced by the full directory specification of the directory in which you will place all the

Using the INTERFACE/AMBER Front End

auxiliary programs supplied with the AMBER/INTERFACE distribution (see the next section). As with the REDIRECT statement, you should include any trailing punctuation required to separate the directory name from the filename.

Once you have modified these assignment statements, you can exit `amber.def`.

- 2) You need to modify the file `misc.def` so that the scripts it contains--scripts which run the supplied auxiliary programs when requested--are correctly executed on your host computer.

If you are installing the script on either a Unix or VAX/VMS computer, you can use pre-customized scripts, supplied with this distribution. For Unix,

```
cp      misc_unx.def  misc.def
```

For VAX/VMS,

```
copy   misc_vms.def  misc.def
```

On other types of machines, you will need to examine `misc_msc.def`, customize it for your host computer, and copy the result to `misc.def`.

- 3) You need to modify the file `run.def`. This IPS contains the EXTERNAL command used to run MINMD, SANDER or GIBBS on the host computer when RUN is specified in the user-input. As with `misc.def`, this distribution comes with pre-customized scripts appropriate for Unix and VAX/VMS computers. For Unix,

```
cp      run_unx.def  run.def
```

For VAX/VMS

```
copy   run_vms.def  run.def
```

On other types of machines, you should customize `run_msc.def` for your host computer, then copy the result to `run.def`.

- 4) Once you have customized the IPS scripts you can then create the template file from the IPS scripts. Invoke INTERFACE, and then, in response to the INTERFACE (?=HELP) > prompt reply

```
IN = STDIN # DEF = amber.def # TEMPLATE = amber.dat # OVERWRITE
```

You can precede either `amber.def` or `amber.dat` by a directory specification, if desired. When the informative message

```
AMBER/INTERFACE TEMPLATE READ
```

Using the INTERFACE/AMBER Front End

appears the template file has been created. You can now safely specify the appropriate end-of-file character (control-d on Unix, control-z on VMS) to terminate INTERFACE execution.

After the template file has been created, you can subsequently invoke the INTERFACE/AMBER front-end by specifying the name and path of the template file you just created when starting INTERFACE (see How Do I Invoke Interface, Section IV above).

Installing the auxiliary programs

(If you have used the install_ambint script to create the “template” files and install the auxiliary files, you are done with the installation. You may skip to section VII).

In addition to INTERFACE itself, the AMBER/INTERFACE front end distribution includes three auxiliary programs:

Program	Purpose
direct.f	Copies the sequential access coordinate archive file created during an AMBER MD simulation to a direct access file.
archivgetd.f	Retrieves the specified coordinate set from the direct access coordinate archive file provided.
archivputd.f	Stores the set of coordinates in the supplied AMBER restart file at the specified location in the named direct access coordinate archive file.

These programs should be compiled and placed where desired. It is expected that the executables will be named direct, archivgetd, and archivputd on a Unix machine, and direct.exe, archivgetd.exe and archivputd.exe on a VAX/VMS computer. The location of the directory which contains the executable programs is specified in the top of the amber.def file, as described in the previous section.

VII) How do I write a new IPS or customize an existing one?

There is a detailed manual which describes the IPS language in full. This manual accompanies the AMBER/INTERFACE distribution. You should refer to this manual for specific details on how the IPS works, how you can customize and add to an existing script, and other features.

VIII) Where can I learn more about commands and constructs allowed in the command input file?

Chapter 2 outlines most of the intrinsic commands and constructs recognized in the command input file. For more information about general issues related to the Interface program and how it functions, you may refer to the accompanying INTERFACE manual, which describes in detail the features available in both the user command (UC) input file and the IPS. For most basic Amber/Interface applications, this manual will be sufficient. But if you intend to do more with the interface than simple customization of existing scripts, it is suggested that you read at least the first three chapters of the Interface manual.

IX) Commands recognized in the command input file

Two types of commands are recognized in this file: 1) Intrinsic commands; and 2) user-defined commands.

Intrinsic commands are those which are provided automatically by INTERFACE. These include control constructs (e.g. IF, DO, etc.), assign statements, and intrinsic functions (e.g. SIN, COS, EXP). These are listed below, and are described in the following chapter (and in detail in the INTERFACE manual which accompanies this distribution).

User-defined commands are the commands which are defined in the IPS files provided. These commands are detailed in chapter 3..

Intrinsic commands for the command input file:

```
ASSIGN (CASSIGN, GASSIGN)
DO ... END DO
DO WHILE ... END DO
EXIT (IF, DO)
GETTEMPLATE
GOTO label, label:
HELP
IF (...) THEN, ELSE IF (...) THEN, ELSE, END IF
IF (...) COMMAND
MANUAL
STOP
! (comment lines)
<> (in line variable substitution)
```

Each of these constructs is described in the following chapter and in the accompanying INTERFACE manual. The commands defined in the IPS files are described in chapter 3.

Intrinsic Features of the Input Language

This chapter lists commands and features which are native to the input file language in all Interface applications. There are three sections. The first lists control constructs, intrinsic commands, and control constructs recognized in the Interface language, plus a few other functionalities (EXIT, GETTEMPLATE, HELP, MANUAL, STOP). The second section lists other elements of the language (comment lines and variable substitution constructs) which can be used. The third section provides a list of intrinsic functions recognized by Interface.

In some cases, the descriptions here are truncated from those in the Interface manual, emphasizing aspects which are relevant to the Amber/Interface.

Intrinsic Commands and Control Constructs

ASSIGN

Syntax:

```
ASSIGN variable_name = expression
```

Description: ASSIGN associates the given variable_name with the expression on the right of the equals sign. "expression" can be a numerical or character constant, variable, or variable expression. variable_name must adhere to the rules for variable_name specification (must start with a letter; must contain only alphanumeric or "_" characters). If "expression" is a character expression, it must be enclosed in character delimiters (usually double quotes ("")).

If variable variable_name has already been assigned, the previous assignment is replaced after the expression on the right of the equals sign is evaluated. Since variable names are not declared with an explicit type, a variable name associated with one data type can be reassigned to a different data type without error. The current data type of any variable is maintained internally.

Assignments made with the ASSIGN command are local to the level at which they are made. That is, assignments made in the IPS file do not affect variables in the UC file, and vice-versa. To make an assignment to a variable_name which can be accessed by both the IPS and UC files, use the GASSIGN command. To make an assignment to a variable_name which only affects the value of the variable at the level complementary to that at which the assignment is made, use the CASSIGN command.

An ASSIGN performed in the UC file is by default non-volatile to memory clearance (same as NVASSIGN). An ASSIGN in the IPS file is by default volatile to memory clearance (same as VASSIGN).

Examples:

Statement	assignment made
ASSIGN A_VAL = 3*(1+6)	A_VAL = 27
ASSIGN A_VAL = A_VAL + 3.	A_VAL = 30.0
ASSIGN CHRNAM = "char data"	CHRNAM = "char data"
ASSIGN R = MOD(2.1,2.0)	R = 0.1
ASSIGN TR = 2.GT.1	--error-- only character and numerical data types can be assigned in an ASSIGN statement

DO

Syntax:

```
DO variable_name = M1, M2, M3
    (...any series of valid commands...)
END DO
```

Description: Do Control loop. The commands between initial opening DO statement and the END DO statement are executed as many times as indicated by the DO loop control parameters, M1, M2 and M3. M1, M2, and M3 are all integer (or real) constants, variables, or expressions. On each iteration, the given variable_name is set to the current control value. The first control value is M1, and on each iteration of the loop, M3 is added to variable_name until variable_name > M2 (if M3 > 0) or variable_name < M2 (if M3 < 0).

By default, M3 = 1. M1 and M2 must be specified. The total number of iterations of the DO loop is given by

$$\text{MAX}(\text{INT}((\text{M2}-\text{M1}+\text{M3})/\text{M3}), 0).$$

Note that if M1 > M2 and M3 > 0, or M1 < M2 and M3 < 0, then the commands within the DO loop will not be executed at all.

When the DO loop finishes executing, variable_name will equal the control value on the last iteration where the contents of the DO loop were executed. (E.g. variable_name=M2, if M3 is omitted). Note that this differs from the FORTRAN 77 standard, where variable_name = the control value on the last iteration + M3 (E.g. variable_name = M2+1, if M3 is omitted).

DO loops can be nested within other DO loops, or within other control structures.

The space between END and DO is required.

Examples:

```
ASSIGN IBEG = 1
ASSIGN IEND = 3
ASSIGN INC = 2

DO I = IBEG, IEND, INC
    ECHO "I =" <I>
END DO
ECHO "I at end of loop = " I
```

would produce:

```
I = 1
I = 3
I at end of loop = 3
```

DO WHILE

Syntax:

```
DO WHILE (logical expression)
    (...any series of valid commands...)
END DO
```

Description: The DO WHILE construct will repeatedly execute the commands within the DO WHILE ... END DO construct until the provided logical expression is false. The parentheses surrounding the logical expression are required. Any of the logical operators described above (section XI) can appear in a logical expression.

The DO WHILE construct can appear nested within other DO WHILE constructs, or within other control structures.

The space between END and DO is required.

Example:

```
ASSIGN I = 1

DO WHILE (I.LT.3)
    ECHO "I =" <I>
    ASSIGN I = I + 1
END DO
```

would produce

```
I = 1
I = 2
```

EXIT

Syntax:

```
EXIT DO
EXIT IF
```

Description: The EXIT command effects an exit from the type of command block specified, by skipping commands until the appropriate END (DO or IF) command is found. For example, EXIT DO will cause an exit of a DO loop, transferring control to the first line outside the appropriate END DO command.

Example:

```
DO I = 1,3
    ECHO "I =" <I>
    IF (I.EQ.2) EXIT DO
END DO
```

would produce

```
I = 1
I = 2
```

GETTEMPLATE

Syntax:

```
GETTEMPLATE TEMPLATE = template_file
                        {# DEF = command_definition_file}
                        {#OVERWRITE}
```

Description: GETTEMPLATE causes the reading of an IPS command definition file, or of a template file derived from the IPS. Any commands read in this IPS/template file are added to any commands read in the IPS/template file specified when INTERFACE was started. Any commands in the IPS which do not lie within a COMMAND definition or DEFER block will be executed immediately when the template file is read.

The `command_definition_file` is the file containing the unmodified IPS. The `template_file` is a modified and processed version of the IPS, created either on a previous run of INTERFACE, or when GETTEMPLATE is specified. It is this file which is actually used when INTERFACE is running. (See section I of the “Interface” manual (Features and Considerations) for more details). If the `template_file` has already been created, do not specify the `command_definition_file` (IPS script). If the `template_file` has not already been created, you must specify both the `command_definition_file`, and the name of the `template_file` which will be created from it. The `template_file` will remain when INTEFACE exits.

Files can be specified with any path name valid for the host computer.

The `TEMPLATE=` and `DEF=` options are separated by a mandatory pound sign (#). We use this non-standard separator for this command (only), because the default slash (/) separator is commonly used in file names.

OVERWRITE: If the IPS file (`command_definition_file`) is modified after creation of a corresponding `template_file`, a new version of the template file should be generated. Use the `OVERWRITE` flag to allow a new version with the same name to overwrite the old version.

Examples:

```
GETTEMPLATE DEF = program.ips # TEMPLATE =
program.tmp
```

would read the IPS in file `program.ips`, and create a template file named `program.tmp`. The commands in `program.ips` would be added to those defined in the `TEMPLATE` file specified when INTERFACE was started.

On a subsequent run of INTERFACE, one could specify

```
GETTEMPLATE TEMPLATE = program.tmp
```

since the template file corresponding to program.ips would already exist.

GOTO

Syntax:

```
GOTO label
    (...any valid series of commands...)
label: command
```

Description: The GOTO command effects a transfer to the line which begins with the specified label. A label may contain any string of alphanumeric characters, plus the underscore character "_". Any other characters or blanks in the label will cause an error.

The label itself must be immediately followed by a colon (:), and must be the first non-blank element on the line. The referenced label may either precede or follow the GOTO statement. If the GOTO statement references a nonexistent label, an error will occur.

A label may optionally be followed by a valid command on the same line.

If file redirection has been requested (see REDIRECT), the GOTO label must be contained in the current file.

A GOTO must not refer to a label which results in a transfer into a DO-loop or IF block. A transfer into an IF or DO block which was defined before the GOTO label was encountered will result in an "unmatched DO...END DO" or "unmatched IF...END IF" type message.

The optional command following a label cannot be an "END" or "ELSE" command.

Examples:

```

        ASSIGN I = 0
START:  ASSIGN I = I+1
        ECHO "I =" <I>
        IF (I.LT.3) GOTO START
        IF (I.EQ.4) GOTO 1_END
        ECHO "Not 4 yet"
        GOTO START
1_END:
        ECHO "I = 4 now"
```

would produce

```

I = 1
I = 2
I = 3
Not 4 yet
I = 4
```

I = 4 now

HELP

Syntax:

```
HELP {command_name} {/option_name}
```

Description: The HELP command will produce a list of commands or options for a command which have been defined in the IPS file. In addition to the listing, any informative HELP message text which has been supplied in the IPS file (using DEFHELP commands) will be echoed to the user.

When HELP is specified with a `command_name`, but no `option_name(s)`, HELP will produce a description of the given command and a listing of all options which are defined for the specified `command_name`.

HELP can be specified with a chosen `option_name`. This will generate a descriptive message about the option. The special character "*" may be specified in the `option_name` field. In this case, help on all available options for the specified command will be reported.

Note that alias names (defined using the ALIAS command) will not appear in the help command listings, but may be specified as explicit arguments to HELP to retrieve the appropriate descriptive messages.

If `command_name` or `option_name` is specified, but does not correspond to a valid name, a warning will be reported, and the program will continue.

command_name: Name of a command defined in the template file.

option_name: Name of an option defined for the corresponding `command_name`

Example:

```
HELP command1/option2
```

would retrieve all information about `option2` with parent `command1`, as defined by DEFHELP commands in the template file.

```
HELP
```

would produce a listing of all commands which were defined in the template file.

IF

Syntax:

```
(I)   IF (logical expression) THEN
        (...)
      {ELSE IF (logical expression) THEN}
        (...)
      {ELSE}
        (...)
      END IF

(II)  IF (logical expression) command
```

Description:

Type (I): The IF () THEN ... ELSE IF () THEN ... ELSE ... END IF construct allows sections of code to be conditionally executed, depending on the values of specified logical expressions. The parentheses surrounding the logical expressions are required.

First, the logical expression in the initial IF () THEN statement is evaluated. If this expression is true, the following block of commands is executed until the another element of the IF block (ELSE IF, ELSE) is encountered, at which point all subsequent commands will be skipped until the END IF statement is found. If no ELSE IF or ELSE statements were specified, all commands between the IF () THEN and the END IF will be executed.

If the logical expression in the initial IF () THEN statement is false, subsequent commands are skipped until either A) an ELSE IF () THEN statement with a logical expression that evaluates as true is found, at which point behavior is as described above for the initial logical expression evaluating to true; B) an ELSE statement is found, at which point all commands will be executed until the terminating END IF command is encountered; C) the IF block terminating END IF statement is found.

"ELSE IF () THEN" and "ELSE" statements are optional parts of the IF block construct. The "END IF" statement is required. As many "ELSE IF () THEN" statements may be used as are necessary, but only one "ELSE" statement be used, and must follow any "ELSE IF () THEN" elements.

Type (II): The IF (logical expression) command construct will conditionally execute the specified command if the provided logical expression is true. If the logical expression is false, no further action is taken. "command" can be any valid command (and associated arguments/options), but should not typically be part of a multi-line construct (e.g an END DO command).

The parentheses surrounding the logical expression are required. The space between END and IF is required.

Examples:

Type (I):

```
DO I = 1,3
  IF (I.EQ.1) THEN
    ECHO "First pass"
  ELSE IF (I.EQ.2) THEN
    ECHO "Second pass"
  ELSE
    ECHO "Third pass"
  END IF
END DO
```

would produce

```
First Pass
Second Pass
Third Pass
```

Type (II):

```
DO I = 1,3
  ECHO "I =" <I>
  IF (I.EQ.2) ECHO "I equals 2"
END DO
```

would produce

```
I = 1
I = 2
I equals 2
I = 3
```

MANUAL

Syntax:

```
MANUAL  
    {/OUTPUT = file_name}  
    {/NO_ALPHABET}
```

Description: The MANUAL command allows the user to request a "manual" of all commands, options, and the associated help messages .

Note that the manual will not list the names of commands which are simply aliases for other definitions.

OUTPUT: If specified, the "manual" will be written to the named file. If the file does not already exist, it will be created. If it already exists, the manual will overwrite contents of the file. If OUTPUT is not specified, the "manual" will be written to STDOUT.

NO_ALPHABET: By default, the commands list reported when the MANUAL command is given is alphabetized. If /NO_ALPHABET is specified, then the commands list will be reported in the order in which the commands are defined in the IPS files.

Example:

```
MANUAL /OUTPUT = manual.dat
```

would write all the available help information and definitions to file manual.dat. The command listing would be alphabetized.

REDIRECT

Syntax:

```
REDIRECT = filename
          {/BEG = M}
          {/END = M}
          {/NOTENOUGH = stop | continue (D)}
          {/DEFAULT}
```

Description: REDIRECT allows redirection of input for either the input file. The specified number of lines are read from the indicated file. Control then reverts to the input file in which the REDIRECT was issued.

REDIRECT commands can appear nested in files being read by a REDIRECT command. A maximum of 10 levels of such redirection are allowed.

Redirection can only be made to formatted sequential access files.

If the filename contains slashes (/), the entire name must be surrounded by double quotes.

filename: The name of the file to be used in the redirection. Any name valid on the host computer can be used, but the name must correspond to a formatted, sequential access file (default type of file). This is the actual file name (not an alias).

The filename specification may contain the special construct

DEFDIR:filename

“DEFDIR:” is a literal string, and must precede the filename. If this construct is provided, DEFDIR: is replaced by a definition previously provided by the user for DEFDIR. Thus, for example, one could specify all redirection in the file as DEFDIR:filename. Then, if later it became desirable to move the location of all the REDIRECT files, only one change would have to be made to the input file: a modification of the directory pointer contained in DEFDIR.

The value of DEFDIR is defined by a REDIRECT/DEFAULT command (see /DEFAULT below).

The special filename "&STDIN" can be used to indicate that redirected input is to come from the standard input source (usually unit 5 == terminal input).

BEG = M: Specifies the number of the line at which reading should begin in the the given file. The value may optionally be specified as "\$+M" or "\$-M", which will be translated into a relative displacement from the end of the file (e.g. \$-1 is the last line in the file).

By default, reading begins at the first line of the file.

END = M: Specified the number of the line at which reading should end in the given file. The value may optionally be specified as "\$+M" or "\$-M" (see above).

By default, reading ends at the end of the file.

NOTENOUGH = stop | continue (D): Specifies the behavior if the /END = M option is given and the specified line exceeds the last line in the actual file. By default, the program will close the file, and control will revert to the file where the REDIRECT command was encountered. If /NOTENOUGH = stop is specified, the program will stop.

DEFAULT: When DEFAULT is specified, the "filename" string provided is defined as DEFDIR for all subsequent REDIRECT commands. When DEFAULT is specified, no actual redirection occurs with this REDIRECT command, and any other qualifiers, if specified, are ignored. Once DEFDIR has been defined by a REDIRECT/DEFAULT command, it can be reassigned by a subsequent REDIRECT/DEFAULT command.

DEFDIR can also be defined using the DEFINE command.

Example:

```
REDIRECT = "/usr/misc/utilities/" /DEFAULT
          (...any series of commands...)
REDIRECT = DEFDIR:COOLING.INTERF
```

The first REDIRECT command would set a value for DEFDIR:. It would be set to a directory specification. This definition is maintained in non-volatile memory and can subsequently be used at any time. Later, we specify redirection to a file given as DEFDIR:IPS1.DEF. The actual redirection will be to file

```
/usr/misc/utilities/COOLING.INTERF.
```

Note that we must enclose the directory specification in double quotes because it contains slash characters which would otherwise be interpreted as option delimiters.

STOP

Syntax:

```
STOP {"string"}  
     {/OUTFILE = file_alias_name}
```

Description: The STOP command will stop program execution, reporting an optional text string, if specified.

"string": Optional text string to be output before the program is stopped. If the string has any embedded slash (/) characters, it must be surrounded by quote characters ("). Text is output starting in column 1, ending in column 80, and is broken into multiple lines at appropriate break points if too long to fit on one line.

Example:

```
STOP "Program stopped"
```

would cause program execution to stop, and the message "Program stopped" to be echoed to standard output.

Other Elements of the Interface Language

In-Line Variable Substitution (<>)

Any command given in either the IPS or at the UC level can contain strings enclosed by “diamond” brackets <>². Any such string is evaluated by the function evaluator before the string is parsed, and <expression> is replaced by this evaluated value. The length of the replacement string is the actual length of the string if <expression> is either an integer or character. If <expression> is a real value, it is by default replaced by a real string in the format G16.5. (The format used for a real string can be changed by setting FORMEC using the DEFINE command in the IPS/template file). "expression" can be any numerical or character constant, variable, intrinsic arithmetic function (e.g. COS, EXP, ABS, etc.), or compound function.

The <expression> construct can appear anywhere in a command line, with two exceptions: 1) The <expression> construct cannot appear in the statement label, if any; and 2) The <expression> construct cannot appear as part of the command-name itself (the first non-blank field on the line, or the first non-blank field following the label, if a label is specified).

If you wish to use < and/or > as a literal character (not part of an in-line variable substitution construct), precede it with a single backslash. E.g.

```
ECHO "This is what brackets look like: \< \>"
```

would return

```
This is what brackets look like: < >
```

Examples:

I)

```
ECHO COS45 = <COS(45.*3.14/180.)>
ASSIGN RNUM<2*3> = 18.
```

would result in parsing of the lines

```
ECHO COS45 = 0.707
ASSIGN RNUM6 = 18. .
```

²There are two exceptions: <>'s cannot be used in statement labels (see GOTO) or a REDIRECT command in the IPS file (but they can be used in a REDIRECT command in the User Command input).

II)

While INTERFACE does not directly support named arrays, the functionality of an array can be implemented using <>'s, e.g.

```
DO I = 1,10
  ASSIGN JUNK<I> = FLOAT(I)
END DO
```

would assign the series of reals 1.,2.,3,...10. to the variables named JUNK1, JUNK2, JUNK3...JUNK10. These could be subsequently accessed in a similar manner, such as

```
DO J = 1,10
  ECHO "JUNK(<J>) = " <JUNK<J>>
END DO
```

which would sequentially echo a series of lines to the user:

```
JUNK(1) = 1.000
JUNK(2) = 2.000
      .
      .
JUNK(10) = 10.000 .
```

(note use of nested <<>>'s in this example).

III)

With some commands you will have doubly protect a “<” or “>” character if you wish to use it literally. For example, with the EXTERNAL command from the UC file, you should double-protect literal angle brackets because the string will twice be searched for them: first when translating the string as an argument; and then again when the passed string is sent to the host operating system. E.g.

```
EXTERNAL "time \> time.dat"
```

would be the appropriate command from the command input level to place the time in file time.dat (where the command actually executed by the Unix operating system will be “time > time.dat”).

Continuation lines (\)

By default, it is assumed that each command uses only one physical line in the input file. If the backslash continuation character “\” appears as the last non-blank character on a line, then the next line will be interpreted as a continuation of the current line. As many continuation lines may be used as desired. The command will be terminated with the first line that does not end with a “\” character.

Example:

```
IF ( AVARIABLE.EQ.1 .OR. AVARIABLE.EQ.2 .OR. \  
    (AVARIABLE.EQ.3 .AND. BVARIABLE.NE.3)) THEN
```

In this example, the IF statement spans two lines, which will be concatenated into a single line (removing the “\” character) before being processed.

Comment lines (!)

Any line (except continuation lines) which begins with the exclamation point comment character “!” in the first column is treated as a comment line, and is not processed. Blank lines are also treated as comment lines.

Example:

```
assign i = 12
assign j = 14

! here we multiply i by j and output the result

assign k = i*j
echo "<k>"
```

Intrinsic Commands Provided in Interface

The *Intrinsic Functions Provided in Interface* INTERFACE

program supports
a variety of

intrinsic functions as part of an algebraic/character expression. These include most of the important functions typically provided in programming languages, plus some additional functions of use in parsing (e.g. BLSTR, UCASE). These functions are listed below. In the descriptions, N, N1, N2, etc. represent numerical arguments (either integer or real). C, C1, C2, etc. represent character arguments. Function names are case independent.

Function Syntax	Description
INT(N)	Integer portion of argument N.
FLOAT(N)	Convert N to floating representation.
NINT(N)	Integer nearest to argument N.
ABS(N)	Absolute value of argument N.
MOD(N1, N2)	Remainder after N1 is divided by N2
SIGN(N1, N2)	Return the magnitude of N1 with the sign of N2.
MAX(N1, N2, N3 . . .)	Return the maximum of N1,N2,N3...Ni (0 < i < 26)
MIN(N1, N2, N3 . . .)	Return the minimum of N1,N2,N3...Ni (0 < i < 26)
SQRT(N)	Return the square-root of N.
EXP(N)	Return "e" (2.718...) raised to the power N.
LOG(N)	Return the natural logarithm of N1 (log base e of N).
LOG10(N)	Return the logarithm, base 10, of N.
SIN(N)	Return the SIN of N. (N in radians).
COS(N)	Return the COS of N. (N in radians).
TAN(N)	Return the TAN of N. (N in radians).
ASIN(N)	Return the ArcSIN of N (in radians).
ACOS(N)	Return the ArcCOS of N (in radians).
ATAN(N)	Return the ArcTAN of N (in radians).
ATAN2(N1, N2)	Returns the ArcTAN of (N1/N2) (in radians).
SINH(N)	Returns the hyperbolic Sine of N.
COSH(N)	Returns the hyperbolic Cosine of N.
TANH(N)	Returns the hyperbolic Tangent of N.
RAN(N1, N2)	Returns a random number on the inclusive interval (N1,N2).The seed for the random number generator can be changed using the DEFINE command.
INDEX(C1, C2)	Returns the position of the character substring C2 in the character string C1. The returned value is the position of the beginning of the substring. If the substring is not found, a value of 0 is returned.
LEN(C1)	Returns the length of the character string C1.
BLSTR(C1)	Returns the string C1 with all leading and trailing blanks stripped off.
UCASE(C1)	Returns the upper-case equivalent of string C1 (only alphabetic characters are changed).

Intrinsic Commands Provided in Interface

UPCASE (C1)	Alternative recognized name for the UCASE function.
IASVAL (C1)	If the variable named C1 has been assigned a value, IASVAL will return 1,2, or 3, depending on whether C1 refers to an integer, real, or character value. If the variable has not been assigned, IASVAL will return 0. IVS variable descriptors may be used. Be sure to surround a variable name you wish to pass as the argument by double quotes.
C1 (N1 : N2)	Returns the substring from position N1 to position N2 of character string C1. NOTE: For this construct, C1 must be a character string or name of a character variable. A substring specifier cannot follow a right parenthesis ")".

Chapter 3

Guide to the AMBER/INTERFACE Front-End

On the following pages, the commands recognized by the AMBER/INTERFACE front end to MINMD, SANDER and GIBBS is described. In addition to the commands described on the following pages, the commands and constructs described in the previous chapter are recognized in the UC file, and can be freely interspersed with the following commands. For some examples, see the appendices.

While most of the following command definitions are self-explanatory, a few of the more specialized commands (such as CHANGE and RESTRAINTS) assume the user has access to the standard AMBER manual for cross reference. In general, if questions arise in interpreting the purposes of the following commands, the user is advised to refer to the AMBER manual.

Note that as of Amber version 4.1, MINMD no longer exists as a separate module. All of the functionalities of MINMD are now available in SANDER. Reference to MINMD is maintained in the following descriptions for compatibility of Interface with older versions of Amber.

Index to Command Specificity

Commands Which Affect MINMD, SANDER and GIBBS:

archive	belly	cap	clearall
com	delete	dielectric	direct
dna	echo	ewald	extcom
external	fast_water	force_field	generate
getarchive	group	info	logcommands
logenergy	nameset	pairlist	parallel
PBC	pdbgen	polarization	position_restraints
putarchive	random_seed	read	report
restart	run	scratchname	sequential
set	shake	steps	temp
time_limit	timestep	title	triplet

Commands specific to MINMD:

md	minimize	minmd
----	----------	-------

Commands specific to SANDER:

change	md	minimization	nmr
peacs	restraints	sander	

Commands specific to GIBBS:

constraints	cnst	decouple	DMW
entropy	free_energy_components		gibbs
imaging	intragroup	mixing	slow_growth
windows			

The above commands are described in alphabetical order, beginning on the next page.

archive

Allows requests for archiving of coordinates/velocities/energies.

Default: No coordinates/velocities/energies are archived.

Options Defined:

coord	velocities	energies	binary
only	soluteonly		

coord

COORDINATES = (M1{,M2}).

Coordinates will be saved every M1 steps. After M2 steps, no additional sets of coordinates will be saved. If both M1 and M2 are omitted, no coordinates will be saved. If M2 is omitted, coordinates will be output every M1 steps for the entire run.

velocities

VELOCITIES = (M1{,M2}).

Velocities will be saved every M1 steps. After M2 steps, no additional sets of velocities will be saved. If both M1 and M2 are omitted, no velocities will be saved. If M2 is omitted, velocities will be output every M1 steps for the entire run.

energies

ENERGIES = (M1{,M2}).

Energies will be saved every M1 steps. After M2 steps, no additional sets of energies will be saved. If both M1 and M2 are omitted, no energies will be saved. If M2 is omitted, energies will be output every M1 steps for the entire run.

binary

Specifies that all archive files will be binary. By default, all such files are formatted. (Also affects the format of the component energy log files with SANDER).

only

ONLY = ({ntwpr0,} ntwprt)

If specified, the coordinates/velocities only for atoms from NTPR0->NTPRT will be written to the requested archive file(s). If NTPRT<0, then the coordinates/velocities only for atoms from NTPR0 to the lastatom of the solute will be written. If a single value is specified with the ONLY qualifier, it is assumed that this represents NTPRT, and an effective value of NTPR0=1 will be used (atoms 1->NTPRT will be written).

soluteonly

If specified, the coordinates/velocities only for atoms of the solute will be written the requested archive file(s). This option has the same effect as specifying a single negative NTWPRT argument with the /only option.

belly

Use belly. Should be followed by GROUP instructions which define the appropriate group. Only the atoms defined by the group instructions will move in a belly run.

Default: No belly is defined or used.

Options Defined:

verbatim

verbatim

Copy the following AMBER-format GROUP input lines verbatim to the appropriate portion of the input file.

If this option is not specified (default), GROUP commands are used to define the group. This is recommended.

cap

Specifies that if a CAP was defined in EDIT, it will be used here.

Default: A cap will not be used, even if defined in EDIT.

Options Defined:

pointer force

pointer

POINTER = natcap.

If specified, natcap is the cap atom pointer, and overrides any cap atom pointer passed from EDIT/PARM.

Default: Use the pointer passed from EDIT/PARM.

force

FORCE = fcap.

Specifies the force constant to be used in CAP restraints.

change

This command can be used to change the values/weights of various parameters for a SANDER simulation. See the documentation of SECTION ONE for SANDER for more details.

```
CHANGE type /FROM=value1 /TO=value2
  /STEP1=istep1/STEP2=istep2
  /EVERY=inc
  /ARITHMETIC(D) /GEOMETRIC
```

Valid options for type (e.g. TEMP0, VDW, etc.) are defined in the SANDER manual.

FROM and TO define the beginning and ending values for the weight change.

STEP1 and STEP2 define the range of steps over which the weight change occurs.

EVERY defines how often the target type value is updated.

ARITHMETIC/GEOMETRIC defines whether the change will occur as an arithmetic or geometric series (see IMULT in manual).

Valid values for "type" include:

BOND	ANGLE	TORSION	IMPROP	VDW	HB	ELEC	NB
ATTRACT	REPULSE	RSTAR	SOFTTR	INTERN	ALL	REST	RESTS
RESTL	NOESY	SHIFTS	SHORT	TEMP0	TAUTP	CUTOFF	NSTEP0
STPMLT	DISAVE	ANGAVE	TORAVE	DISAVI	ANGAVI	TORAVI	

The various options are defined in detail in the manual. For a few special "type" options, value1, value2, etc. have different meanings. See the SANDER manual.

Options Defined:

every arithmetic geometric

from

FROM = value1

Specifies value1, corresponding to istep1.

Default = 0.0

to

TO = value2

Specifies value2, corresponding to istep2.

Default = 0.0

step1

STEP1 = istep1

Specifies istep1, first step for change to be active.
Default = 0.

step2

STEP2 = istep2

Specifies istep2, last step for change to be active.
Default = 0 (active from step1 to end or run).

every

EVERY = iinc

Defines IINC, the number of steps between each change of the target value.

Default = 0

arithmetic

Specifies that change shall be carried out with a arithmetic progression (sets IMULT = 0). This is the default.

geometric

Specifies that change shall be carried out with a geometric progression (sets IMULT = 1).

ARITHMETIC and GEOMETRIC are mutually exclusive. Specify at most one.

clearall

CLEARALL clears all stored memory. In scripts where you will be running multiple AMBER jobs (multiple invocations of the “run” or “generate” commands), you should separate the complete sets of commands which describe the runs from one another by CLEARALL commands. This will ensure that the parameters set in one simulation will be cleared before the next simulation is prepared.

For example, one frequently runs multiple simulations by placing the commands which set up the runs within a DO-loop construct. In this case, you would typically place a CLEARALL command before the closing END DO statement for the loop. Then on each start of the loop you would essentially effect a fresh start for the input.

Options Defined:

(none)

cnst

The CNST command is used to define restraints or constraints to be applied during a GIBBS run. Added constraints can be used to drive a PMF calculation if the CONSTRAINTS command is also specified.

This command is ignored in MINMD and SANDER runs.

Syntax:

```
CNST /AT1=iat1 /AT2=iat2 /AT3=iat3 /AT4=iat4
     /LAMBDA1=rlambda1 /LAMBDA2=rlambda2
     /K1=rkeq1 /K2=rkeq2 /REQ1=req1 /REQ2=req2
     /REPLACE /UMBRELLA
     /TORSION=harmonic | sinusoidal
     /CONSTRAINT
     /IPER=iper /IPER2=iper2
```

The values are as described in the GIBBS manual (section 19 input when INTR > 0).

The REPLACE flag sets IZE=1.

The UMBRELLA flag sets IUMB=1

TORSION=harmonic sets ITOR = 0 (default)

TORSION=sinusoidal sets ITOR = 1.

CONSTRAINT sets ITOR=2.

Options Defined:

at1	at2	at3	at4
lambda1	lambda2	k1	k2
req1	req2	replace	umbrella
torsion	constraint	iper	iper2

at1

AT1 = iat1

Defines the number of the first atom of the restraint/constraint.

Default: None (must be specified).

at2

AT2 = iat2

Defines the number of the second atom of the restraint/constraint.

Default: None (must be specified).

at3

AT3 = iat3

Defines the number of the third atom of the restraint/constraint. If not specified (only AT1 and AT2 are specified), a distance restraint is defined.

Default: 0

at4

AT4 = iat4

Defines the number of the fourth atom of the restraint/constraint. If not specified (only AT1, AT2, and AT3 are specified), a valence angle restraint is defined. If this all four atoms are defined, a torsional restraint is defined.

Default: 0

lambda1

LAMBDA1 = rlambda1

rlambda1 -> rlambda2) defines the range of lambda values over which the restraint/constraint will be applied.

Default: 0.0

lambda2

LAMBDA2 = rlambda2

rlambda1 -> rlambda2) defines the range of lambda values over which the restraint/constraint will be applied.

Default: 0.0

k1

K1 = rkeq1

The force constant to be used in the restraint term added to the force field when a restraint is defined. K1 is the force constant at lambda=LAMBDA1. Ignored if /CONSTRAINT is also specified.

Default: 0.0

k2

K2 = rkeq2

The force constant to be used in the restraint term added to the force field when a restraint is defined. K2 is the force constant at lambda=LAMBDA2. Ignored if /CONSTRAINT is also specified.

Default: 0.0

req1

REQ1 = req1

The target value for the added restraint/constraint at $\lambda = \text{LAMBDA1}$.
Default: 0.0

req2

REQ2 = req2

The target value for the added restraint/constraint at $\lambda = \text{LAMBDA2}$.
Default: 0.0

replace

Specifies that any terms in the standard energy function which apply to the same internal coordinate as this restraint will be "zeroed" out. (Sets IZE=1; see manual). Ignored if /CONSTRAINT is also specified.

Default: Overlapping energy terms will not be zeroed out (IZE=0).

umbrella

Specifies that the added restraint term will be considered an "umbrella" weighting term in the simulation.

Default: the added restraint will be added into the force field, and standard free energy statistics will be determined (Sets IUMB=1; see the AMBER/GIBBS manual).

Ignored if /CONSTRAINT is also specified.

torsion

TORSION = harmonic (D) | sinusoidal

Defines the form of the potential term used when a torsional restraint has been defined. HARMONIC specifies that a harmonic restraint term will be used. SINUSOIDAL specified that a sinusoidal (COS) term will be used. The TORSION option is ignored if an bond or angle restraint is being defined, or if /CONSTRAINT is also specified.

constraint

If specified, the atom sequence defined with this CNST command defines a *constraint* to be applied (rather than a restraint). (Sets ITOR=2; see the AMBER/GIBBS manual).

Default: A restraint is applied.

iper

IPER = iper

Defines the value of variable IPER, which is used to request a "secondary" set of restraints/constraints. See the AMBER/GIBBS manual for more details.

iper2

IPER2 = iper2

Defines the value of variable IPER2, which is used to request a "secondary" set of restraints/constraints. See the AMBER/GIBBS manual for more details.

com

Specifies values related to Center of Mass motion removal.

Options Defined:

remove steps

remove

Remove COM motion on first step.

Default: Center of Mass motion is not removed on the first step.

steps

STEPS = nscm.

Remove COM motion every nscm steps.

Default: Never remove motion after first step.

employed.

PF: Use Potential Forces. This is the default method.

CF: Use Constraint forces. If any constraints whose target value changes with lambda are within a closed ring, you should specify CF.

PF_OVERRIDE: If any constraints applied to the perturbed group are within a closed ring, but none of the constraints within closed rings change with lambda, you may force the use of the PF method by specifying the PF_OVERRIDE flag.

This option is new for Amber/Gibbs version 4.1. TI with constraint energies is not allowed in earlier versions.

decoupled

DECOUPLED = electrostatic | vdw.

Allows electrostatic decoupling to be requested. The argument tells which leg of the decoupling is to be carried out. (Electrostatic corresponds to IELPER = +1; VDW corresponds to IELPER=-1. See the manual for more details).

Default: No decoupling is performed.

This option is only available in GIBBS runs. For MINMD and SANDER it will be ignored.

Options Defined:

(none)

delete

DELETE file1 {,file2, file3, ...}.

Deletes the given set of files. Up to 20 filenames, separated by commas, can be specified. Enclose any filenames which contain forward slashes "/" by double quotes, or enclose the entire list by left-right parentheses ().

Options Defined:
(none)

dielectric

DIELECTRIC = N{R}.

Specifies what dielectric constant is to be used. N is a real or integer number (or numerical expression). R is optional, but if present indicates a distance-dependent dielectric constant is to be used.

Options Defined:
(none)

direct

This command converts an output coordinate archive file (mdcrd) from sequential access to direct access and outputs the direct access file.

Options Defined:

input output restart

input

INPUT = filename.

Specifies the name of the coordinate archive file (mdcrd) from md.

output

OUTPUT = filename.

Specifies the name of the direct access file to be created.

restart

RESTART = filename

Specifies the name of the restart file corresponding the simulation that produced the coordinate archive file. Read to determine the number of atoms in the system. Assumed to be formatted.

DMW

Specifies that a window simulation is to be performed using the Dynamically Modified Windows methodology. See the AMBER/GIBBS manual for more discussion of Dynamically Modified Windows.

Default: A window simulation is performed using fixed width windows.

Note: this command is only available for GIBBS runs. For MINMD and SANDER it will be ignored.

Options Defined:

points	correlation_c	target	initial_dlambda
min_dlambda	max_dlambda	reset	last_lambda

points

POINTS = (iavslp{,iavslm}).

IAVSLP specifies how many points will be used in calculating the bestfit line used to predict slope. IAVSLM, if given, specifies the minimum number of points required to calculate a slope, when fewer than IAVSLP points are available (in the early stages of a simulation). By default, IAVSLM=2. Set IAVSLM = -1, if no slope should be calculated until IAVSLP points are available.

correlation c

CORRELATION_C = corrs1.

Defines the minimum acceptable correlation coefficient for best-fit lines. If the calculated correlation coefficient is < corrs1, the number of points used for the line will be halved, and the bestfit line re-calculated. A minimum of two points will be used.

target

TARGET = amxmov.

Specifies the target free energy change per window.

initial dlambda

INITIAL_DLAMBDA = almdl0.

Specifies value of d_lambda to use until enough points to calculate a slope (see /POINTS) have been accumulated.

min dlambda

MIN_DLAMBDA = dlmin.

Specifies the minimum allowable value of lambda over any interval.

max dlambda

MAX_DLAMBDA = dlmax.

Specifies the maximum allowable value of lambda over any interval.

reset

RESET = amxrst.

Specifies the maximum allowable free energy change over any interval. If the change is greater than this, d_lambda will be adjusted, and the free energy for the window recalculated. The default value is 10* TARGET (10*amxmov).

last lambda

LAST_LAMBDA = almstp(i)

Specifies the last lambda value for which the values of TARGET, MIN_LAMBDA, MAX_LAMBDA and RESET defined with this DMW command will apply.

If this is the first DMW command specified:

The values will apply to the range (X, LAST_LAMBDA), where X is START_LAMBDA (from the WINDOWS or SLOW_GROWTH command).

If this is not the first DMW card specified:

The values specified for TARGET, MIN_LAMBDA, etc. will apply to the range (LAST_LAMBDA(i-1), LAST_LAMBDA), where LAST_LAMBDA(i-1) is the value of LAST_LAMBDA specified with the *previous* DMW command.

Set to <0.0 or >1.0 to indicate all the values of TARGET, MIN_LAMBDA, MAX_LAMBDA, and RESET defined here apply to all remaining values of lambda from X to the end of the run (where X is as defined above). This is the default if /LAST_LAMBDA = is not specified.

If only one DMW command is given, LAST_LAMBDA should not be specified (or else given as <0.0 or >1.0). This will result in the set of TARGET, MIN_LAMBDA, MAX_LAMBDA and RESET values specified with the command being used for the entire simulation.

NOTE: If multiple DMW instructions are given to define different values of TARGET, MIN_LAMDBA, MAX_LAMBDA and RESET over different ranges of lambda, the values associated with POINTS, CORRELATION_C, and INITIAL_LAMDBA will be taken from the *last* DMW instruction provided.

dna

DNA

Modify charges at end hydroxyls for DNA. Prevents artifactual interactions between terminal hydroxyl groups and nearby moities. Has no effect for non-DNA molecules.

Default: No charge modification for DNA is performed.

Options Defined:
(none)

echo

Echo string. Allows the user to echo the given string to the standard output (STDOUT). Strings containing slashes (/) or commas (,) MUST be surrounded by double quotes.

Options Defined:
(none)

entropy

Requests that entropy information be calculated and reported

Default: No entropy information is calculated or reported.

This command only has an effect in GIBBS.

Options Defined:

d_temp

d temp

D_TEMP = dtuse.

Specifies the delta(temperature) to be used in approximating the differentials required to calculate entropy differences. Default = 0.5. dtuse should not typically be much larger than this. This value is only used when calculating entropy with FEP. When using TI, D_TEMP is ignored.

Ewald

Specifies that an Ewald summation be performed to calculate the electrostatic contributions to the total free energy. As currently implemented in AMBER, this command will also force the non-bonded Lennard-Jones interactions to be calculated using an Ewald-type summation. By default, Ewald summations are performed using the Particle Mesh Ewald (PME) method. The user can optionally specify that the summation be performed by the exact method. The exact method can be significantly slower for large systems.

Options Defined:

unit_cell	nfft	spline_order	leave_charge
verbose	exact_ewald	direct_sum_tol	

unit_cell

UNIT_CELL = (a , b, c, alpha, beta, gamma).

Defines the unit cell parameters. A,B,C are in Angstroms; ALPHA, BETA, and GAMMA are in degrees. This information must be specified with the EWALD command, and overrides any corresponding information in the PARM file. However, if one is reading box information from the restart file, the box information from that file overrides both the PARM info and the info specified here. You should specify ALPHA=BETA=GAMMA=90.0 for a rectangular periodic box.

nfft

NFFT = (nfftx , nffty , nfftz)

Specifies the size of the charge grid in each dimension (x,y,z). Higher values lead to greater accuracy, but at correspondingly greater simulation cost. It is found that setting NFFT in each dimension to approximately the nearest integer of the box length in that direction (i.e. a grid spacing of roughly 1 Angstrom in each dimension) allows reasonable results. The Fast Fourier Transform routine used by the PME method will be significantly more efficient if each of the NFFT values is a product of powers of 2, 3, and 5.

spline_order

SPLINE_ORDER = order.

Defines the order of the B-spline interpolation. Higher order leads to better accuracy. A minimum order of 3 should be used, and an order of 4 is recommended to generally provide good results. The computational cost of the PME increases with roughly the third power of the spline order.

leave_charge

By default, when a PME calculation is being performed, the net charge on the unit cell is modified to be zero. If LEAVE_CHARGE is specified, the net charge on the unit cell is not modified.

verbose

If specified, VERBOSE will increase the amount of information output during a PME run.

exact ewald

By default, specifying the command EWALD will turn on the Particle Mesh Ewald (PME) method. If the EXACT_EWALD option is chosen, an exact Ewald summation is performed. For systems of > 500 atoms, the exact Ewald summation will be *significantly* slower than PME.

direct sum tol

DIRECT_SUM_TOL = dsum_tol

Specifies the value of the direct sum at the Lennard-Jones cutoff value be less than dsum_tol. This, in turns, modifies the effective value of the cutoff CUT for the direct sum. Typical values are 1.0D-6 to 1.0D-5. This value must be specified with the EWALD command.

EXTCOM

Defines a series of commands to be run as a subprocess. The command EXTCOM is followed by the series of commands and ended by a line END_EXTCOM.

The following rules must be followed in defining commands:

- 1) If the command is to contain a back squiggly bracket ,“{“, enclose that bracket in a pair of double quotes.
- 2) If the command contains any forward or backward diamond brackets (< or >), the bracket must be preceded by a backslash (e.g. \>).
- 3) If you wish to pass a command containing double quotes, you should specify each quote as a series of four double quotes in a row. E.g. the line

```
echo "  hello"
```

would become

```
echo """"  hello"""".
```

This command can be useful for running a program that requires interactive input without writing a specific interface definition for that command. For example, on a Unix computer you might want to run the program “myprogram” which requires four values to be specified:

```
extcom/noforout
  myprogram \<\< idone
            specified_infile
            specified_outfile
            option1_reply
            option2_reply
  idone
end_extcom
```

Options Defined:

```
scratch_file      noforout          wait          nowait
```

scratch file

Defines the name of the scratch file to be used in executing this command. Default is scrtxtcom1. The string given with with the SCRATCHNAME command (if any) will always be appended to this filename.

noforout

Suppress any output from this series of commands that would normally be sent to the standard output device.

wait

Wait until the commands finish before continuing (default).

nowait

Submit commands to host operating system, then continue immediately.

fast_water

By default, Sander and Gibbs automatically search for the TIP3P water molecules in the system, and set up pointers to make calculations involving these molecules run much faster. Two types of speedup are involved: 1) An analytic SHAKE routine applicable to 3-point molecules is used; 2) a special procedure is used to calculate the non-bonded interactions between pairs of TIP3P water pairs.

In most circumstances, these default speedups will be desirable. On rare occasion, one may wish to inhibit these speedups, or change the definition used by the program to define TIP3P molecules. This command allows this to be done.

Options Defined:

off water_definition notip

off

OFF = (non-bond,shake)

By specifying this option with one or both of the allowable arguments, one can turn off the use of the fast water procedures. OFF=(non-bond) turns off the fast calculation of TIP3P-TIP3P non-bonds. OFF=(shake) or OFF=(non-bond,shake) turns off BOTH the fast analytical shake and the fast calculation of non-bonds.

water_definition

WATER_DEFINITION = (residue, oxygen, hydrogen1, hydrogen2)

Amber determines which residues are 3-point TIP3P waters and the fast water routines are applied to these. These waters must have the appropriate residue name, and appropriate atom names. The default residue and atom names expected by the programs are those which are typically assigned in EDIT for TIP3P water. If you wish to change these defaults, this can be done by specifying them with the /water_definition option. All four names should be specified if this option is used.

For standard operation with TIP3P water, you should never need to specify this option.

notip

By default, Gibbs assumes that if you are running a periodic boundary conditions (PBC) simulation with solvent, the solvent is TIP3P water. A special characteristic of this solvent model is that there are no h-bond (10-12) interactions between any pair of solvent molecules. A calculational speedup is thus obtained by skipping all such h-bond interactions.

With Gibbs, you must specify the /NOTIP option if you choose to use PBC with a solvent model where there should be h-bond (10-12) interactions calculated between

pairs of solvent molecules. Note that in any case, all 10-12 interactions between solvent and solute molecules will still be determined normally.

This option only has an effect in Gibbs. It is ignored for Sander/Minmd. This option is also ignored if PBC is not specified.

force field

This command allows the user to modify certain characteristics of the force field.

Options Defined:

elect14_scale nb14_scale

elect14 scale

ELECT14_SCALE = scee

Defines the electrostatic scaling factor for 1-4 interactions. The default value is 2.0.

NB14 scale

NB14_SCALE = scnb

Defines the non-bonded scaling factor for 1-4 interactions. The default value is 2.0.

free energy components

Request that either A) components of the total *free energy* be logged to the file PATNRG (assignable at run-time); or B) that the derivatives of the free energy with respect to charge, epsilon, r^* , and any applied constraints be logged to PATNRG. Option (A) is the default when FREE_ENERGY_COMPONENTS is specified. Option (B) is invoked if /DERIVATIVES is specified.

FREE_ENERGY_COMPONENTS only has an effect for GIBBS runs. For MINMD and SANDER it will be ignored

One can specify component analysis (option A above) in two ways. First, you can specify that each atom, residue, or molecule be a "group". In this case, all energies are attributed to atoms which are either in the perturbed group, or whose position depends on a constraint (when constraint free energy contributions are being accumulated). A legend to exactly which atoms are assigned to which groups will appear at the top of the PATNRG file.

One can also specify component analysis (option A) using the /GROUP qualifier. In this case, the user provides group input to specify which atoms should be considered part of which group. When /GROUP is chosen, any atom specified will be included in the group, whether or not it is part of the "perturbed group" (specified in PARM). So, for example, you could use FREE_ENERGY_COMPONENTS/GROUP to determine the contribution from solvent atoms and (separately) from solute atoms.

Free energy derivatives are requested by specifying the /derivatives qualifier. /GROUP input *must* be specified when free energy derivatives are requested.

Notes:

- 1) The various options for FREE_ENERGY_COMPONENTS determine the form of the output log, but do not affect the free energy calculation.
- 2) Free energy component analysis is exact when slow growth or thermodynamic integration is being carried out. But because $\exp(a+b) \neq \exp(a) + \exp(b)$, component analysis will only be approximate when any kind of FEP window simulation is being carried out.
- 3) While the total free energy is state function, the components are not, and so are best interpreted qualitatively.

GROUP input, when required, is specified by GROUP commands following this command.

Options Defined:

derivatives	atom	residue	molecule
group	breakdown	steps	equilibrate

derivatives

Log the derivatives of the free energy with respect to charge and non-bond parameters epsilon and r*. The derivatives with respect to constraints will also be calculated if constraints/energy has also been specified.

atom

Log the free energies per atom.

residue

Log the free energies per residue.

molecule

Log the free energies per molecule.

group

Log the free energies per user-defined group. When /group is specified, the FREE_ENERGY_COMPONENTS command must be followed immediately by the group definition. The group is defined using GROUP commands.

For free energy components:

Free energy components will be logged as defined by the GROUP definition, subject to the condition that only those atoms which are part of the perturbed group (defined in PARM), or which move with a constraint (if CONSTRAINT energy contributions are being calculated) will ultimately be included. All atoms not explicitly included in a group will be put in a final single group.

For free energy derivatives (/DERIVATIVES specified):

Free energy derivatives will be logged for only those atoms included in a group definition. Any atom of the system may be designated as part of any group (but each atom will be a member of at most one group).

breakdown

By default, only total free energies per component are logged in the PATNRG file. If /BREAKDOWN is specified, a breakdown of each component energy into electrostatic, non-bond, and internal components will also be provided. /BREAKDOWN has no affect when /DERIVATIVES is specified.

steps

STEPS = ntatdp

The component free energies (or derivatives) will be reported every ntatdp steps. Note that if free energy components are being logged, a free energy report will occur at a particular multiple of ntatdp steps only if the free energy accumulators have been updated since the last report. For free energy derivatives, energies will be reported every ntatdp steps in all cases.

equilibrate

EQUILIBRATE = ncmpdr

If specified, gives the number of steps of equilibration to be used when calculating free energy derivatives. After the first ncmpdr steps, the accumulators for the free energy derivatives calculation will be cleared and reset. Free energy derivative values reported from that point forward will only reflect averaging since the accumulators were cleared.

/EQUILIBRATE only affects calculations of free energy derivatives. This flag is ignored if /DERIVATIVES is not also specified.

generate

Requests that a file appropriate for use as input to MINMD, SANDER or GIBBS be generated. For correct operation, this command must be the last command of the input definition.

Default: No file will be generated.

Options Defined:

output

output

OUTPUT = filename

Specifies the name of the file to be used as MINMD/SANDER/GIBBS input.

namelist

By default, the GENERATE command creates a file in the formatted “numbers in columns” format. By specifying the NAMELIST option, a namelist-style output file will be created. A namelist-style file is *required* for Sander version 5.0 and above.

If you are generating a namelist-style output file, and would like to include additional variables in the namelist output that are not defined by standard Amber/Interface commands, use the NAMESET command.

getarchive

GETARCHIVE gets the specified number archived set of coordinates from the specified direct-access archive file. Such a direct access archive file can be created in one of two manners:

- 1) You can use the DIRECT command to convert the standard sequential access coordinate archive file created by AMBER into a direct access file.
- 2) You can store individual coordinate sets to a direct access archive-format file using the PUTARCHIVE command.

The appropriate direct access archive-format file must have been created before issuing the GETARCHIVE command.

Options Defined:

archive output number periodic

archive

ARCHIVE = filename.

Specifies the direct-access archive file to be read.

output

OUTPUT = filename.

Specifies the name of the file to contain the specified coordinate set retrieved from the archive file.

number

NUMBER = isetnum

Specifies the sequential number of the archived set of coordinates to be retrieved.

periodic

Specifies that the system whose coordinates are being read was run with periodic boundary conditions (PBC) on. The default is no periodic boundary conditions.

gibbs

Specifies that the GIBBS program will be used. This command should be the first command issued when a GIBBS simulation is being set up.(although any commands intrinsic to the INTERFACE command language may safely precede it). This command will supersede a previously-issued MINMD or SANDER command, if issued.

Options Defined:

(none)

group

```
Syntax: GROUP
  {/ATOMS = (IAT11,IAT12,  IAT21,IAT22, ...)}
  {/RESIDUES = (IRES11,IRES12,  IRES21,IRES22, ...)}
  {/ONLY = (atomname1:atomtype1:treename1:residuenam1,
           atomname2:atomtype2:treename2:residuenam2,
  ...)}
  {/WEIGHT = weight}
  {/TITLE = title}
```

A series of GROUP definition commands should follow commands which require group input (e.g. BELLY). See the AMBER manual for further discussion of how GROUP input is interpreted (e.g. when a negative residue number is specified, and how the ONLY/FIND command operates).

Once a GROUP command is issued, group definition continues until the first non-GROUP command is issued. At that point, the group definition is assumed to be complete.

Atom and residue ranges are specified in pairs (as many pairs as desired) with the ATOMS and RESIDUES qualifiers.

FIND and FILTER are valid aliases for ONLY. Filter specifications with the ONLY command are given by four fields separated by colons (:) as shown. As many filter specifications may be given as are desired. A "*" in any field is a wildcard match.

Only one value may be given with the /WEIGHT command, and this value is ignored unless position restraints are being defined.

Options Defined:

atoms	residues	only	title
weight			

atoms

```
ATOMS = (IAT11,IAT12, IAT21,IAT22, ...)
```

Specifies ranges of atoms to be included in the group definition (up to 40 are allowed). All atoms in each range pair (e.g. IAT11->IAT12) will be included.

residues

```
RESIDUES = (IRES11,IRES12, IRES21,IRES22, ...)
```

Specifies ranges of residues to be included in the group definition (up to 40 are allowed). All residues in each range pair (e.g. IAT11->IAT12) will be included.

only

```
ONLY = (atomname1:atomtype1:treename1:residuenam1,
```

```
atomname2:atomtype2:treename2:residuenam2, ...)}
```

Defines filter specifications for a group definition. Of the atoms defined by the ATOMS and RESIDUES qualifiers, only those which also meet the criteria set by the ONLY command will be used. ONLY has the same effect as FIND, as defined in the AMBER manual. Each filter specification has four parts as shown above. Up to 40 filter specifications can be provided. A wildcard * character can be provided for any field. This will guarantee a match for that field.

title

TITLE = title

Defines a title for the group.

weight

WEIGHT = weight

Defines the force constant to be used when this group definition is being used for positional restraints. Ignored in all other cases.

imaging

IMAGING = {atom | residue(D) | residue2}

Define the type of non-bonded imaging to be performed with PBC

This option is only available in GIBBS. In MINMD or SANDER it will be ignored.

ATOM gives old-style Amber atom-based imaging (not recommended).

RESIDUE gives residue-based imaging. Each atom of any solute or solvent residue sees the same image of any interacting residue. This is the default and is recommended for most simulations.

RESIDUE2 is like RESIDUE, except that for each atom of the solute different whole-residue images of interacting residues may be used. This option will result in significantly slower performance than RESIDUE when using PBC on a vector machine. It may be useful when using a non-water solvent which is fairly long in one or more dimensions.

Options Defined:

(none)

intragroup

Specifies that certain intra-perturbed group contributions to the free energy will be used. Specify as many options as are appropriate.

Default: No intra-group (but all other contributions) will be included in the reported free energies.

This command is only available in GIBBS runs. For MINMD and SANDER it will be ignored.

Options Defined:

none non_bond 1_4 internals

none

Include no intra-group contributions. This is the default.

non_bond

Include non-bonded contributions (except 1-4's).

1_4

Include 1-4 non-bonded contributions. If you specify this option you must specify /NON_BOND too.

internals

Include contributions of intra-perturbed group internals.

logcommands

Requests that the commands in the Amber/Interface script used to generate the Amber input file be either written to this input file, logged to another specified file, or written to the standard output device (STDOUT).

If logcommands is specified with no options, the commands will be written to the end of the input file created when the “generate” command is given.

Version \geq 4.1 Amber programs have provisions to copy these logged commands to the formatted user output file. Thus, by logging commands here, you will obtain a record of these commands in your Amber output.

Default: These commands are not logged anywhere.

Options Defined:

output

output

OUTPUT = filename

Log the commands to the given filename, rather than to the end of the Amber input file created with the “generate” command. If you wish to have the commands echoed to the standard output device (typically the terminal), set filename to “stdout”.

logenergy

This command searches the specified minimization output file for the final energy, and logs the energy, plus a supplied sequence number, to a specified file.

Options Defined:

`infile` `logfile` `sequence`

infile

INFILE = filename.

Specifies the name of the formatted output file from minimization which is to be searched for the final energy.

logfile

LOGFILE = filename.

Specifies the name of the file to which the final energy and sequence number are to be logged.

sequence

SEQUENCE = rlog.

Specifies a "log number" to be written to the log file, along with the energy. Written as a real number.

md

Select a MD run. MINIMIZE should not be specified with MD.

This selection is available in MINMD and SANDER. It is ignored when GIBBS has been specified.

Default: There is no default. You must specify MD if you wish to run molecular dynamics.

Options Defined:
(none)

minimize

Select a minimization run. MD should not be specified with MINIMIZE.

This selection is available in MINMD and SANDER. It is ignored when GIBBS has been specified.

Default: There is no default. You must specify MINIMIZE if you wish to run minimization.

Options Defined:

conjugate_gradients		steepest_descents	steps
initial_step	maximum_step	rms_energy	delta_energy

conjugate_gradients

Full conjugate gradients will be used. The first 10 steps are steepest descents at the start of the run and after every nonbonded pair list update.

steepest_descents

STEEPEST_DESCENDS [=ncyc]

Steepest descents minimization will be carried out for NCYC, steps, followed by conjugate gradients. If the =NCYC portion of the option is omitted, only steepest descents will be used.

steps

STEPS = maxcyc.

Specifies the maximum number of cycles of minimization.

initial_step

INITIAL_STEP = dx0.

Specifies the initial step length for min. The default value is usually acceptable.

maximum_step

MAXIMUM_STEP = dxm.

Specifies the maximum step length for min. The default value is usually acceptable.

rms_energy

RMS_ENERGY = drms.

Specifies the rms energy gradient convergence criterion. If the rms energy gradient \leq drms before the maximum allowable number of steps, minimization will stop.

delta_energy

DELTA_ENERGY = drms.

Specifies the delta energy gradient convergence criterion. If the difference in energy for successive steps is \leq delta_energy, minimization will stop.

minmd

Specifies that the MINMD program will be used. This command should be the first command issued when a MINMD simulation is being set up.(although any commands intrinsic to the INTERFACE command language may safely precede it). This command will supersede a previously-issued SANDER or GIBBS command, if issued.

The MINMD program is obsolete as of Version 5.0.

Options Defined:

(none)

mixing

Allows specification of the type of non-bonded mixing to be carried out. Also allows specification of the van der waals radius to be used for interactions between pairs where one atom is shrinking.

This command is only available in GIBBS runs. For MINMD and SANDER it will be ignored.

Options Defined:

vanishr old new

vanishr

VANISHR = dsx0

Defines the distance dsx0 (in Angstroms) to which interatomic vdw interactions will shrink for pairs where one/both atoms are vanishing. The default is the radius of the non-vanishing interacting particle (if any). A smaller value is often desirable.

Default: Radius of the non-vanishing particle.

old

Specifies old-type GIBBS non-bonded mixing. Provided for backwards compatibility, but not recommended. (Sets IOLEPS=1; see the AMBER/GIBBS manual for more details).

new

Specifies new-type GIBBS non-bonded mixing. This is the default.

nameset

NAMESET variable_name = variable_value

Defines a variable name and associated value to be output to the &cntrl namelist when namelist output is being generated using the GENERATE/NAMELIST command (see the GENERATE command). This command is useful for adding variables to the namelist that are not by-default generated by the AMBER/Interface.

NOTE: If the variable_value is a character string, it must be surrounded by triple quotes, e.g. atomname = “““C1””” would assign the character string C1 to the variable atomname.

nmr

Can be used to define options related to "nmr" restraints. These include:

- 1) Request that the restraint violations be printed at the beginning or end of the simulation, and specify the file(s) which is to contain the report(s).
- 2) Specify the name of a file which contains restraint definitions in the AMBER namelist format.
- 3) Request that various types of specialized refinement (e.g. NOESY volumes) be carried out, and specify the names of files from which parameters for these refinements will be read.
- 4) Specify additional control parameters required when carrying out NOESY volume restraint refinement.
- 5) Request that soft-repulsion non-bonds replace the standard 6-12 function.
- 6) Request that a verbatim transfer of lines following the NMR command be transferred to the generated input file. This can sometimes be valuable when using a portion of a previously-generated input deck.

This command only has an effect with SANDER.

Options Defined:

soft	cutoff	listin	listout
disang	volumes	shifts	dumpave
penalty_function	update	scale_factors	mass_scale
maximum_submolecules		verbatim	

soft

Soft=force

Use soft repulsion non-bonds with the given force constant. force (kcal/mole)

Default: Non-bonded interactions are calculated normally.

cutoff

CUTOFF = pencut.

Any restraints whose contribution to the energy is less than the given cutoff pencut (kcal/mole) will not be reported individually in the end-of-run summary.

Default: cutoff is 0.0.

listin

LISTIN = filename

Specifies the name of the file to contain an informational listing of the restraints defined, and for each the deviation from the target value, at the beginning of the simulation. Specify LISTIN = POUT to cause this information to be written to the normal output file.

Default: This information will not be written.

listout

LISTOUT = filename

Specifies the name of the file to contain an informational listing of the restraints defined, and for each the deviation from the target value, at the END of the simulation. Specify LISTOUT = POUT to cause this information to be written to the normal output file.

Default: This information will not be written.

disang

DISANG = filename

Specifies the name of a file from which distance and angle restraint information will be read. *The format of restraints defined in this file is described in the SANDER manual.*

If, in addition to specifying DISANG=, you specify any restraints using RESTRAINT commands, the restraints will be written in the appropriate AMBER format to the DISANG file. In this case, it will be assumed that the DISANG file does not previously exist.

If you specify DISANG= without specifying any RESTRAINT commands, it is assumed that the DISANG file already exists.

If you are specifying a large number of restraints, it is recommended that you either specify them in the namelist format appropriate for the DISANG file, or that you specify a DISANG file along with the RESTRAINT commands. In the latter case, each restraint will be converted to the appropriate namelist format by the Interface program, and placed in the DISANG file (so that subsequent runs can use the DISANG file directly). The reason for this recommendation is that processing of large numbers of RESTRAINT commands Interface can be time consuming.

Default: All restraints will be specified by RESTRAINT commands in the standard input file.

volumes

VOLUMES = filename

Specifies that NOESY volume restraint are to be imposed. The restraints will be read from the given filename in the namelist format given in the SANDER manual (see SECTION FOUR of the Sander documentation).

Default: No NOESY volume restraint information will be read, and no NOESY restraints will be imposed.

shifts

SHIFTS = filename

Specifies that chemical shift restraints are to be imposed. The restraints will be read from the given filename in the namelist format given in the SANDER manual (see SECTION FIVE of the Sander documentation).

Default: No chemical shift restraints will be read, and chemical shift restraints will be imposed.

dumpave

DUMPAVE = filename

Specifies the name of the file to which the time-averaged values of all restraints will be written if IDMPAV has been set to a non-zero value by a CHANGE instruction with type = DISAVI, ANGAVI or TORAVI. (See the SANDER manual for more information).

Default: Not such information will be written.

penalty function

PENALTY_FUNCTION = {absolute(D) | sum_of_squares | r6weight}

Defines the target function to be optimized when volume restraints are being used (/VOLUMES specified).

absolute:	Optimize the sum of the absolute values of the errors.
sum_of_squares:	Optimize the sum-of-squares of the errors.
r6weight:	For NOESY intensities, the penalty will be of the form $awt*[Ic^{**}(1/6)-I0^{**}(1/6)]^{**2}.$ Chemical shift penalties will be as for absolute.

Default: The “absolute” target function will be optimized.

Ignored when volume restraints are not being used.

update

UPDATE = noeskp

When volume restraints are being used, UPDATE can be used to specify that the NOESY volumes will be re-evaluated every noeskp steps. At all intermediate steps, the intensities and derivatives at the last evaluation step will be used.

Default: noeskp=1 (the NOESY volumes are re-evaluated at every step).

Ignored when volume restraints are not being used.

scale factors

SCALE_FACTORS = iscale

Defines the number of additional variables (beyond the 3N structural parameters) to be refined in a NMR refinement simulation. There are currently no supported options using a value of ISCALE.ne.0. This option is included here for future compatibility.

The default is 0.

Ignored when volume restraints are not being used.

mass scale

MASS_SCALE = scalm

When volume restraints are being refined, this option allows the "mass" for the additional scaling factors (see /SCALE_FACTORS) to be defined. All additional scale factors will use this same mass. The larger the "mass", the slower these extra variables will respond to their environment.

The default is scalm = 100amu.

Ignored when volume restraints are not being used.

maximum submolecules

MAXIMUM_SUBMOLECULES = mxsub

Defines the maximum number of submolecules that will be used

The default is mxsub = 1.

verbatim

The following lines will be written verbatim to the input file being generated, in the appropriate location for weight change restraint definition information when SANDER is being run. This option is provided to allow you to use input in the appropriate AMBER format from a pre-Interface run, *but is not recommended*.

Follow the series of lines to be copied verbatim with an ENDNMR command on a separate line.

The only other options to NMR that are valid with VERBATIM are SOFT and CUTOFF.

Default: All restraint and change information is specified using RESTRAINT/CHANGE commands (and from files specified with the NMR command).

pairlist

Define options related to pair-list generation.

Options Defined:

update	old	new	cutoff
cut2nd	cutpert	static	

update

UPDATE = nsnb.

The pairlist will be update every nsnb steps.

old

Pairlist to be generated by comparing all atoms of residue pairs, and including the residue pair if any pair of atoms between them is closer than the cutoff. This option is much slower than the default, where for periodic systems the pairlist is generated considering only the first atom of each solvent residue (e.g. "O" in H₂O). /OLD gives the pairlist behavior of AMBER versions 3.0 and earlier.

If one is using a solvent other than water, and the solvent is long on one or more dimensions, it may be advantageous to use the /OLD option. Otherwise, the default (/NEW) is recommended. (See the AMBER manual entry for NTID). NEW is the default.

new

The non-bonded pairlist for solvent atoms in periodic systems is generated considering only the first atom of each solvent molecule (e.g. "O" in H₂O). This is the default. (See manual entry for NTID).

cutoff

CUTOFF = cut.

Defines the primary non-bonded cutoff (in Angstroms) to be used.

cut2nd

CUT2ND = cut2nd

Defines an optional secondary cutoff (Anstroms). Non-bonded interactions between the primary cutoff (specified by /cutoff) and the secondary cutoff are only re-calculated at every pairlist update. This allows one to include these longer-range (and, presumably, more slowly changing) interactions in a calculation at a modest cost.

cutpert

CUTPERT = cutprt

CUTPRT = cutprt (alternative syntax)

Defines an optional alternative cutoff for interactions with atoms of the perturbed group in Gibbs. Other non-bonded interactions use the primary cutoff.

It is legal to specify both /cutpert and cut2nd. In this case, for the perturbed group those interactions between cutpert and cut2nd constitute the secondary range of interactions that are only calculated every non-bonded update.

CUTPERT has no effect for minmd/sander simulations.

static

STATIC = (read,write)

Allows the use of a static (read-from-disk) pairlist. If STATIC=read is specified, the initial pairlist will be read from file PRLIST at program startup. If STATIC=write is specified, the pairlist will be written to file PRLIST at every non-bonded update. The read and write options can be specified together, if desired. This option is typically most useful for debugging, not for standard simulations.

This option is only available in SANDER, not in MINMD or GIBBS.

parallel

Specifies that the program is to be run in parallel. This command is currently properly implemented only for code that was compiled with parallel directives on a Silicon Graphics (SGI) computer.

Options Defined:

processors

processors

PROCESSORS = numthread.

Defines the number of parallel processes to run.

PBC

Request the simulation be carried out using Periodic Boundary Conditions.

Default: System is not run using Periodic Boundary Conditions.

Options Defined:

volume	pressure	all_solute_nb	target_pressure
tau	uniform	submolecule	no_ss_image
box_dimensions	no_translate_solute		no_shift

volume

System run at constant volume. Either VOLUME or PRESSURE must be specified with PBC.

pressure

PRESSURE {= isotropic(D) | anisotropic}

System run at constant pressure. By default, isotropic scaling will be used. Specify = anisotropic to use anisotropic scaling.

all solute nb

Calculate all solute-solute n-b interactions, regardless of cutoffs. This can be useful for highly charged solute molecules.

Default: Cutoffs are applied to solute-solute interactions.

target pressure

TARGET_PRESSURE = pres0.

Sets the target pressure for constant pressure PBC.

tau

TAU = tau_{tp}.

Sets the pressure coupling constant for constant pressure PBC.

uniform

Selects uniform submolecule scaling for constant pressure PBC (this is the default).

submolecule

Selects submolecule Center of Mass Scaling for constant pressure PBC.

UNIFORM and SUBMOLECULE are mutually exclusive.

no ss image

Perform no solute-solvent imaging. (See manual entry for IMGSLT).

Default: Solute-solvent imaging is performed.

box dimensions

BOX_DIMENSIONS = (BOX(1), BOX(2), BOX(3))

Overwrites the box dimensions read from the topology or coordinate file. This option will only work for GIBBS.

Default: The box dimension passed from PARM are used.

no translate solute

By default, solute molecules which exit the primary image box will be translated back into that box. If NO_TRANSLATE_SOLUTE is specified, these molecules will not be translated back into the box. This may be desirable with large solutes. Solvent atoms which exit the primary image box are *always* translated back into the box.

NOTE: This option only affects GIBBS runs. In minmd and sander, solute molecules are always translated back into the primary image box.

NOTE2: As of version 4.1, if the /all_solute_nb option has been specified, no solute atoms are ever translated back into the primary image box, regardless of whether the /no_translate_solute flag has been specified.

no shift

By default, every 500 steps during molecular dynamics with PBC (in SANDER or GIBBS), the entire system will be shifted so that the center of geometry of the solute will be centered in the primary image box. By specifying this option with GIBBS, the shifting can be disabled.

NOTE: This option only affects GIBBS runs. In SANDER, shifting is always carried out. This option cannot be specified with the /no_translate_solute.

The “shifting” feature is new to Amber version 4.1.

pdbgen

This command creates a pdb-format file from an AMBER format restart file.

Options Defined:

verbatim position binary

input

INPUT = filename

Specifies the name of the coordinate restart file (restart) from min/md/sander/gibbs/GETARCHIVE,etc.

output

OUTPUT = filename

Specifies the name of the pdb file to be created.

param

PARAM = filename

Specifies the name of the topology file (output from PARM) for this system.

binary

BINARY = {input,param}

Indicates that the specified files are in binary format. By default, both the input and param files are assumed to be in formatted form.

peacs

Request that a PEACS (Potential Energy Annealed Conformational Search) be carried out. In a PEACS search, MD is carried out so that a constant potential energy contour is followed. The target potential energy contour which is followed is slowly annealed as the calculation continues. In a recent study, PEACS showed promise as a more efficient alternative to standard MD as a conformational sampling tool.

A PEACS search contrasts to standard MD, where the pathway is one of constant total energy. Only molecular dynamics (not minimization) can be carried out with PEACS.

For more on the PEACS technique, refer to Schaik *et al.*, J. Comp. Aided Mol. Design 6, 97 (1992).

This option is only available in SANDER. It will be ignored in MINMD and GIBBS.

Options Defined:

initial_energy tauv0 tauv

initial energy

INITIAL_ENERGY = vzero

VZERO = vzero (alternative syntax)

Defines the level of the initial potential energy contour to be followed. If none is specified, the initial energy of the system is used.

tauv0

TAUV0 = tauv0

Defines the rate constant for annealing of the target energy level over time. This option must be specified with the peacs command.

tauv

TAUV = tauv

Defines the coupling constant between the target constant-energy contour and the actual contour followed.

The default value of TAUV is 0.1 psec.

polarization

Run calculation including scf-polarization.

NOTES:

- 1) To run a polarization calculation, the appropriate information must have been supplied in the PARM stage.
- 2) For Gibbs and Sander, polarization is only implemented in versions ≥ 4.1 ; in earlier versions, this command will have no effect.

Options Defined:

3body ions

3body

Request that 3-body terms are to be included. This option only implemented in Gibbs (versions ≥ 4.1).

Triplets must be defined using the "TRIPLET" command.

ions

IONS = nion

Gives the number of ions in the system. This value *must* be supplied when a 3-body calculation has been requested. It is ignored otherwise.

position_restraints

Requests that position restraints be implemented. Should be followed by the appropriate GROUP command input. (See the AMBER manual entry for NTR).

Default: No position restraints are imposed.

Options Defined:

verbatim position binary

verbatim

Specifies that the following lines are the group input in AMBER format, and are to be copied verbatim to the appropriate section of the input file.

If this option is not specified (default), the group definition is provided using GROUP commands, which should follow this command. This is recommended.

position

Position restraints will be implemented. This is the default with the POSITION_RESTRAINTS command (and is currently the only possibility).

binary

Specifies that the coordinates used for the position restraints are in binary form. By default, they are formatted.

putarchive

Put the specified set of restart-format AMBER coordinates at the specified position in a chosen direct-access archive file. If the archive file already exists, the coordinates will be added at the specified location. If the archive file does not already exist, it will be created, and the coordinates added at the specified location.

All PUTARCHIVE stores to a particular direct-access archive file must store the same number of coordinates.

Options Defined:

archive	input	number	periodic
firstatom	lastatom		

archive

ARCHIVE = filename.

Specifies the name of the archive file to which the set of coordinates is to be written.

input

INPUT = filename.

Specifies the name of the restart-format set of coordinates to be written to the archive file.

number

NUMBER = isetnum.

Specifies the sequence number in the direct-access archive file which shall contain the coordinates.

periodic

Specifies that the system whose coordinates are being written was run with periodic boundary conditions (PBC) on. The default is no periodic boundary conditions.

firstatom

FIRSTATOM = ifirst

Defines the first atom of the input restart coordinate set to be written to the selected location of the specified archive file. The default atom is 1.

lastatom

LASTATOM = ilast

Defines the last atom of the input restart coordinate set to be written to the selected location of the specified archive file. The default atom is the last atom of the input restart coordinate set.

random_seed

RANDOM_SEED = ig.

Assign ig, the seed for the random number generator. The default is usually acceptable.

Default: 71277

Options Defined:
(none)

read

Specifies what values to read from input coordinate file.

Options Defined:

velocities box formatted unformatted

velocities

velocities {=reset}

Coordinate file also contains velocities. If the optional phrase =reset is specified then velocities will be read, but reassigned according to current forces.

Default: Only coordinates are read.

box

Coordinate file also contains box parameters.

Default: Box parameters are not read. BOX is only valid if VELOCITIES is also given.

formatted

Formatted coordinate input (default).

unformatted

Unformatted coordinate input.

report

Allows specification of how often energy information will be written to the normal output file.

Options Defined:

steps

steps

STEPS = ntp.

Every ntp steps, energy info will be written to the user (formatted output file). This option must be specified with REPORT.

restart

RESTART

Specifies that job is a restart.

Options Defined:

(none)

restraints

Defines an internal (bond, angle, or torsion) restraint. See the documentation of SECTION THREE for SANDER for more details.

This command only has an effect for SANDER (ignored with MINMD and GIBBS).

```
RESTRAINTS syntax:
RESTRAINTS
  /AT1=inum1:name1/AT2=inum2:name2/AT3=inum3:name3/AT4=inum4:name4
  /R1=(r1 {,r1a}) /R2=(r2 {,r2a}) /R3=(r3 {,r3a}) /R4=(r4 {,r4a})
  /K2=(rk2 {,rk2a}) /K3=(rk3 {,rk3a})
  /JCOUPLING = (A,B,C)
  /STEP1=nstep1 /STEP2=nstep2
  /GRPAT = (inum1:name1 {,inum2:name2 ,inum3:name3, ...})
  /GROUP = R6 | COM
  /EVERY = ninc
  /ARITHMETIC(D) /GEOMETRIC
  /RELATIVE
  /NOTIMEAVE
```

In general:

The values are as described in the SANDER manual. See that manual for more details.

Atom name specifications (for AT1, AT2, AT3, AT4, GRPAT) are given in the format
number : name

If both fields (separated by a colon) are given, number is the residue number and name is the name of an atom in that residue.

You may omit the colon and the following name. In this case, it is assumed you are specifying atoms by absolute atom number. You must specify all atoms in a RESTRAINT card by either residue_number : name or just atom_number. Do not mix specification formats for a single RESTRAINT.

ARITHMETIC and GEOMETRIC correspond to IMULT = 0 and 1, respectively.

GROUP = R6 means use $\langle r^{*-6} \rangle^{*-1/6}$ group-to-group averaging (IR6=1).

GROUP=COM means use center-of-mass to center-of-mass restraints when an atom group is defined (IR6 = 0).

RELATIVE means the restraint target values are relative to their current values (IRSTYP = 1).

NOTIMEAVE means exclude the restraint from time-averaging if time-averaged restraints have been requested (IFNTYP = 1). Time-averaged restraints are requested using a CHANGE instruction.

For any RESTRAINT instruction after the first one, the values of R1, R2, R3, R4, R1A, R2A, R3A, R4A, K2, K3, K2A, K3A, EVERY, ARITHMETIC/GEOMETRIC and GROUP,

if not reset, will default to the values set of the previous RESTRAINT card where they were specified.

Options Defined:

at1	at2	at3	at4
r1	r2	r3	r4
k2	k3	jcoupling	step1
step2	grpat1	grpat2	group
every	arithmetic	geometric	relative
notimeave			

at1

AT1 = inum1:name1.

Specifies the residue number and atom name reference for atom 1. If no colon is found in the value specified, it is assumed you have just specified inum1, and that this is the absolute atom number of atom 1.

at2

AT2 = inum2:name2.

Specifies the residue number and atom name reference for atom 2. If no colon is found in the value specified, it is assumed you have just specified inum2, and that this is the absolute atom number of atom 2.

at3

AT3 = inum3:name3.

Specifies the residue number and atom name reference for atom 3. If no colon is found in the value specified, it is assumed you have just specified inum3, and that this is the absolute atom number of atom 3.

at4

AT4 = inum4:name4.

Specifies the residue number and atom name reference for atom 4. If no colon is found in the value specified, it is assumed you have just specified inum4, and that this is the absolute atom number of atom 4.

r1

R1 = (r1 {,r1a}).

Defines the target values of r1 (corresponding to STEP1) and r1a (corresponding to STEP2). If r1a is omitted, it is set equal to r1.

r2

R2 = (r2 {,r2a}).

Defines the target values of r2 (corresponding to STEP1) and r2a (corresponding to STEP2). If r2a is omitted, it is set equal to r2.

r3

R3 = (r3 {,r3a}).

Defines the target values of r3 (corresponding to STEP1) and r3a (corresponding to STEP2). If r3a is omitted, it is set equal to r3.

r4

R4 = (r4 {,r4a}).

Defines the target values of r4 (corresponding to STEP1) and r4a (corresponding to STEP2). If r4a is omitted, it is set equal to r4.

k2

K2 = (rk2 {,rk2a}).

Defines the value of force constant rk2 (corresponding to STEP1) and rk2a (corresponding to STEP2). If rk2a is omitted, it is set equal to rk2.

k3

K3 = (rk3 {,rk3a}).

Defines the value of force constant rk3 (corresponding to STEP1) and rk3a (corresponding to STEP2). If rk3a is omitted, it is set equal to rk3.

jcoupling

JCOUPLING = (A,B,C)

Specifies that this restraint should be on the J value corresponding to the torsion defined by at1-at2-at3-at4. At each MD step, J is calculated from the underlying torsion tau using the Karplus relationship

$$J = A \cos^2(\tau) + B \cos(\tau) + C$$

where A, B, and C are given as arguments to the JCOUPLING option. This option has no effect if at1:at4 define a distance or valence angle.

step1

STEP1 = nstep1.

Defines the value of nstep1, the first step at which this restraint is to be applied. Default = 0.

step2

STEP2 = nstep2

Defines the value of nstep2, the last step at which this restraint is to be applied. Default = 0 (apply restraint from STEP1 to the end of the simulation).

grpat1

GRPAT1 = (inum1:name1 {,inum2:name2 , inum3:name3, ...})

Defines a series of atoms whose averaged position will be used in a distance restraint in lieu of the atom specified with AT1. Must only be used with distance restraints.

Each atom is specified as inum1:name1, just as for AT1, AT2, etc. If no colons are found in the values specified, it is assumed you have just specified inum1 for each atom, and that this is the absolute atom number of atom i. You must use the same

format (absolute atom numbers or residue numbers with atom names) for all atom specifications, both here, and with the AT1, AT2 specifiers.

Default: No group will replace atom 1.

grpatt2

GRPATT2 = (inum1:name1 {,inum2:name2 , inum3:name3, ...})

Defines a series of atoms whose averaged position will be used in a distance restraint in lieu of the atom specified with AT2. Must only be used with distance restraints.

Each atom is specified as inum1:name1, just as for AT1, AT2, etc. If no colons are found in the values specified, it is assumed you have just specified inum1 for each atom, and that this is the absolute atom number of atom i. You must use the same format (absolute atom numbers or residue numbers with atom names) for all atom specifications, both here, and with the AT1, AT2 specifiers.

Default: No group will replace atom 2.

group

GROUP = COM (D) | R6 (D).

Determines variable IR6, which chooses what type of averaging will be used when single atoms have been replaced by group definitions (see GRPAT1 & GRPAT2 here, and the IGR/GRNAM descriptions in the SANDER manual).

GROUP=COM chooses center-of-mass averaging. This is the default.

GROUP=R6 chooses $\langle r^{*-6} \rangle^{*-1/6}$ averaging.

every

EVERY = ninc.

Sets the value of NINC, which determines how frequently changes in the target values r1,r2,r3,r4,rk2 and rk3 (as they change to r1a,r2a,r3a,r4a,rk2a,rk3a) will be carried out. The default is EVERY=0, which effects the change at every step.

arithmetic

Specifies that change shall be carried out with a arithmetic progression (sets IMULT = 0). This is the default.

geometric

Specifies that change shall be carried out with a geometric progression (sets IMULT = 1).

relative

If chosen, specifies that the values r1,r2,r3,r4 (and r1a,r2a,r3a,r4a) are relative displacements from the current value of the restraint being defined. By default (if RELATIVE not specified), the values are absolute values.

notimeave

If chosen `_and_` if time-averaged restraints are being applied (TYPE = DISAVE, ANGAVE or TORAVE) in a weight CHANGE command, then this restraint will be excluded from time-averaging. By default, all restraints of a given type will be time-averaged, if time-averaging is chosen. This command sets IFNTYP =1.

run

Spawn a process to run MINMD/SANDER/GIBBS with the appropriate files assigned.

Options Defined:

program	input	param	coordinates
refcoord	output	restart	inf
window_stats	map_matrix	map_scratch	arc_coord
arc_veloc	arc_energy	atnrg	mincoord

program

PROGRAM = program_name

By default, when you specify RUN, the location of the required program (MINMD, SANDER or GIBBS) used will be that specified when the Amber/Interface template script was installed. If you wish to use a different version of the program, you can specify /PROGRAM =program_name here. Note that you should specify the complete directory path and program name.

Default: The programs defined when the AMBER/Interface IPS script was installed will be used.

input

INPUT = filename

Input file: control data for the run. This file must be specified.

Alias names "pin"/"mdin" can also be used to select this option.

param

PARAM = filename

Input file: Topology file (created by PARM). This file must be specified.

Alias names "pparm"/"prmtop" can also be used to select this option.

coordinates

COORDINATES = filename

Input file: Initial positions and (optionally) velocities. This file must be specified.

Alias names "pinprd"/"inprd" can also be used to select this option.

refcoord

REFCOORD = filename

Input file: Reference coordinates for optional position restraints. This file must be specified if POSITION_RESTRAINTS have been requested, otherwise it is not used.

Alias names "prefc"/"refc" can also be used to select this option.

output

OUTPUT = filename

Output file: Formatted results and diagnostics.

Alias names "pout"/"mdout" can also be used to select this option.

restart

RESTART = filename

Output file: Restart coordinates and velocities.

Alias names "prest"/"restrt" can also be used to select this option.

inf

INF = filename

Output file: Short file containing a summary of current energies. Continually updated while a run is executing.

Alias names "pinfo"/"mdinfo" can also be used to select this option.

window_stats

WINDOW_STATS = filename

Output file: A concise summary of important energy information for each window (or interval).

WINDOW_STATS is only used for GIBBS runs.

Alias name "micstat" can also be used to select this option.

map_matrix

MAP_MATRIX = filename

Output file: Contains data related to the matrix of free energy data generated. Only used when a 2-D free energy map is being created.

MAP_MATRIX is only used for GIBBS runs.

Alias name "constmat" can also be used to select this option.

map_scratch

MAP_SCRATCH = filename

Output file: A scratch file which contains data required when generating a matrix of free energies corresponding to two independent sets of constraints.

MAP_SCRATCH is only used for GIBBS runs.

Alias name "cnstscrt" can also be used to select this option.

arc_coord

ARC_COORD = filename

Output file: Archived coordinate sets.

Alias names "pcoord"/"mdcrd" can also be used to select this option.

arc_veloc

ARC_VELOC = filename

Output file: Archived velocity sets.

Alias names "pvel"/"mdvel" can also be used to select this option.

arc_energy

ARC_ENERGY = filename

Output file: Archived energy sets.

Alias names "pen"/"mden" can also be used to select this option.

atnrg

ATNRG = filename

Output file: File to contain log of free energy components if requested (see FREE_ENERGY_COMPONENTS).

If FREE_ENERGY_COMPONENTS/DERIVATIVES has been specified, then this file will contain the free energy derivatives.

Alias names "patnrg"/"atnrg" can also be used to select this option.

mincoord

MINCOORD = filename

Output file: File to contain multiple sets of minimized coordinates if requested for a MINMD run.

This option only available in MINMD (not SANDER or GIBBS).

Alias name "mincor" can also be used to select this option.

sander

Specifies that the SANDER program will be used. This command should be the first command issued when a SANDER simulation is being set up.(although any commands intrinsic to the INTERFACE command language may safely precede it). This command will supersede a previously-issued MINMD or GIBBS command, if issued.

Options Defined:

(none)

scratchname

SCRATCHNAME = appendix_string

The scratchname command can be used to specify a string to be appended to the end of the names of temporary files created while AMBER/Interface is running. These temporary files are created when executing commands such as DIRECT, PUTARCHIVE, etc., which are run as subprocess scripts. By specifying unique values for scratchname, you can run several Interface processes from the same directory without risk of two processes trying to access the same temporary file.

Options Defined:
(none)

set

SET variable_name = value.

Allows user to define any variable name, as given in the amber manual, to a specified value.

E.g.

```
SET SCNB = 1
```

Options Defined:

(none)

shake

Requests that SHAKE be applied to bonds.

Default: SHAKE is not used for any bonds.

Options Defined:

all tolerance retry

all

SHAKE is to be performed on all bonds. By default, SHAKE is only applied to bonds to hydrogen atoms outside of the perturbed group.

tolerance

TOLERANCE = tol.

The geometrical tolerance used in implementing SHAKE. In GIBBS, this value will override the distance tolerance set by the CONSTRAINT/TOLERANCE= command, if specified.

retry

RETRY = N.

Defines number of times program will allow SHAKE to fail and attempt to continue before stopping. Default = 0. Option only has effect for GIBBS.

slow_growth

Specifies that a slow growth simulation will be carried out. You must specify either SLOW_GROWTH or WINDOWS for a free energy simulation.

Note: This command only available in GIBBS runs. For MINMD and SANDER, this command will be ignored.

Options Defined:

time	start_lambda	nodoublewide	increase
decrease	ti		

time

TIME = ctimt.

Specifies the length of simulation time to be performed in spanning the [0,1] lambda interval.

start_lambda

START_LAMBDA = almda.

Specifies the starting value for lambda.

nodoublewide

Specifies that double-wide sampling will not be performed. By default, double-wide sampling will be performed.

increase

Specifies that lambda will increase with time (0->1)

decrease

Specifies that lambda will decrease with time (1->0)

ti

In the context of slow growth, this means that the differentials required will be calculated by multiplying the analytic derivatives by the lambda increment. Normally the differentials are calculated as a numerical difference of two energies.

steps

STEPS = nstlim.

Sets the number of steps per run for MD.

Options Defined:

runs

runs

RUNS = nrun.

Sets the number of runs of nstlim steps each for MD.

temp

Set temperature-related flags.

Options Defined:

constant	target	initial	scale
coupling	tau_solute	tau_solvent	dtemp
last_solute	limit_velocity	maxlimit	remove_freedom

constant

CONSTANT = energy | temperature.

Specifies whether constant energy or constant temperature MD is to be carried out.

target

TARGET = temp0.

Specifies the target temperature for constant temperature MD.

initial

INITIAL = tempi.

Specifies the temperature used to assign initial velocities if no velocities are read. If not specified, velocities are calculated from current forces.

scale

SCALE = heat

If specified, and /INITIAL has also been specified, and velocities are being assigned at run-time, all velocities assigned will be scaled by the value heat provided.

Default: No scaling is performed.

coupling

COUPLING = split | single (D) | random,nstep[,nset] | quick.

For constant temperature dynamics, specifies the type of scaling.

SPLIT: Separate scaling will be performed for the solute & solvent.

SINGLE: A single scaling factor will be used for the solute and solvent.

RANDOM,NSTEP[,NSEL]: Velocities will randomly reassigned in a Maxwellian distribution whenever the temperature deviates from the target temperature by more than DTEMP, and every NSTEP steps. By default, all velocities will be rescaled. If NSEL is also specified, only NSEL velocities will be rescaled each time. NSEL can

only be set for GIBBS. In SANDER, all atoms are rescaled every NSTEP steps when this option is chosen.

QUICK: A quick rescaling of velocities to bring the temperature back to the target value will occur each time the current temp. deviates by more than DTEMP from the target.

NOTE: The RANDOM and QUICK options are only available with SANDER and GIBBS (not with MINMD).

tau_solute

tau_solute = tautp. Specifies the temperature coupling constant to be used for solute atoms (or all atoms if COUPLING = SINGLE).

tau_solvent

tau_solvent = tauts. Specifies the temperature coupling constant to be used for solvent atoms if COUPLING = SPLIT.

Default: Same tau_solvent = tau_solute.

dtemp

dtemp = dtemp. Specifies the value of DTEMP. DTEMP is used for in some temperature scaling schemes. See the COUPLING = option for more details.

last_solute

LAST_SOLUTE = isolvp.

For SANDER and GIBBS, specifies the last *atom* to be considered part of the solute when carrying out “split” temperature coupling (/coupling=split).

For MINMD, specifies the last *residue* to be considered part of the solute when carrying out “split” temperature coupling.

Ignored when not carrying out “split” temperature coupling.

Default:

The last solute atom pointer passed from PARM is used. Note that this pointer will be = NATOM if a periodic system has not been defined. In this case, you must specify LAST_SOLUTE to effect separate solute/solvent scaling.

limit_velocity

LIMIT_VELOCITY = vlimit

VLIMIT = vlimit (alternative syntax)

Specifies the magnitude of the maximum allowable velocity component for any atom. The default (0.0) means no limits are set. Can be useful in cases where an occasional hot spot in the system is making your simulations unstable. For example, use of this

option in nmr refinement calculations can sometimes make those simulations much more stable. When used, a liberal tolerance (vlimit) is suggested (e.g. 5-10 at 300K).

This option is also available in Gibbs. However, if you are experiencing frequent hot-spots in Gibbs, this can be symptomatic of other problems in the system, and limiting the velocity in that case may lead to non-Boltzmann statistics. For this reason, it is strongly suggested that for Gibbs you set a modest maximum number of times the maximum velocity can be exceeded using the /MAXLIMIT= option.

This option is new to version 4.1 Amber. It will be ignored when used with older versions of the program.

maxlimit

MAXLIMIT = ivemax

When specified with the /LIMIT_VELOCITY= option, /MAXLIMIT allows the user to limit the total number of steps on which velocity limiting may be performed. If MAXLIMIT is exceeded, the program will stop with an error message. By default (MAXLIMIT=0), the velocities may be reset due to LIMIT_VELOCITY on an unlimited number of steps.

This option is only available with Gibbs (version 4.1). It is ignored by MINMD/SANDER.

remove freedom

REMOVE_FREEDOM = ndfmin

The number of degrees of freedom in the system is used to calculate the temperature. The number of degrees of freedom will be automatically calculated by the program. In all standard cases, the value determined automatically will be correct and you should not specify this option.

However, if your system contains linear triatomic moieties, the number of degrees of freedom automatically calculated will be incorrect by a factor of 1*(number of linear triatomic moieties). In such a case, you should specify this factor with the REMOVE_FREEDOM flag.

time_average

Requests that the restraints which have been specified be imposed as “time-averaged.” That is, the time-averaged value of the relevant internal coordinate, rather than the instantaneous value, is restrained to the target value.

TIME_AVERAGE Syntax:

```
TIME_AVERAGE = type
    /tau=value1
    /power=ivalue2
    /pseudo_force
    /last_tau=value
    /step1=istep1 /step2=istep2
    /dump=idmpave
    /initial = value /actual /target
```

type: distance, angle, or torsion.

Gives the type of time averaged restraint class being defined. You must specify a type with the time_average command. Each TIME_AVERAGE command affects only the class of restraints specified as “type.”

Options Defined:

tau	power	pseudo_force	last_tau
step1	step2	dump	initial
actual	target		

tau

TAU = value

The value of tau to be used in the exponential decay term. Default is 0.0, which will give no exponential decay (true average).

power

POWER = ivalue

The average used with be internal_coordinate**ivalue (specify integer).

pseudo_force

By default, the forces for each class of restraint are calculated exactly. If /pseudo_force is specified, the forces are calculated approximately as $(dE/dr_{ave}) * d(r(t)/dx)$, where “r” is the internal coordinate being restrained. This form can be useful in avoiding the potential instabilities due to the presence of a term dependent on the $(1+power)$ exponentiation term in the exact form. See the Amber manual for more details.

last tau

LAST_TAU = value

The value of tau to be used in the exponential decay term when reporting the final restraint violations (requested using the command `nmr/listout=filename`). This value of tau is not used for any other portion of the calculation, and does not affect the trajectory. The default is that no exponential decay (linear averaging) is used for this final list.

step1**step2**

STEP1 = istep1.

STEP2 = istep2.

By default, the value of tau will not change during the simulation. If STEP1 and or STEP2 are specified, the value of tau defined with this command will only apply to the range step1->step2. *Use of /step1 and /step2 is not generally recommended.*

dump

DUMP = ivalue

If specified > 0 and a dump file has been specified with a `NMR/DUMPAVE=` command, then the time-averaged values of all restraints will be dumped to the dumpave file every "dump" steps. If dump is specified with more than one time_average command, the value specified with the first time_average command will be used.

initial

INITIAL = value

The value of the integral that gives the time-averaged values is undefined on the first step. By default, the integral is assigned the current value of the internal on the first step. Specifying a value for initial allows you to change this default (also see `/actual` and `/target` below).

actual

The value specified with `/initial` will be an offset to the actual value of the particular internal:

$$r = r(\text{first-step}) + \text{"initial"}$$

This is the default.

target

The value specified with `/initial` will be an offset to the target value of the particular internal:

$$r = r(\text{target}) + \text{"initial"}$$

time_limit

TIME_LIMIT = timlim.

Set maximum allowed job time, in seconds.

Options Defined:
(none)

timestep

TIMESTEP = dt.

Defines the timestep (in psec) to be used.

Options Defined:
(none)

title

TITLE = title_name.

Allows user to specify a title for the job.

Options Defined:
(none)

triplet

Define a triplet when the user has requested 3-body terms be included in a polarization calculation (see polarization/3body). These commands will be ignored if polarization/3body has not been specified.

TRIPLET Syntax:

```
TRIPLET / ATOMS = (atname1 , atname2 ) \
  / ACON = (acon1 [, acon0 ] ) \
  / BETA = (beta3b1[, beta3b0] ) \
  / GAMMA = (gama3b1[, gama3b0] )
```

All options are required. The acon0, beta3b0 and gam3b0 are only required in Gibbs, and give the values of these parameters in the "lambda=0" state.

Options Defined:

atoms	acon	beta	gamma
-------	------	------	-------

atoms

ATOMS = (atname1, atname2)

Defines the names of two additional atoms making a triplet with each central ion.

acon

ACON = (acon1 [, acon0])

Defines the pre-exponential factor for the triplet. For Gibbs, you must specify both the pre-exponential factor in the lambda=1 state (acon1) and in the lambda=0 state (acon0). For Sander, only acon1 is used.

beta

BETA = (beta1 [, beta0])

Defines the beta value for the triplet. For Gibbs, you must specify both the beta value in the lambda=1 state (beta1) and in the lambda=0 state (beta0). For Sander, only beta1 is used.

gamma

GAMMA = (gamma1 [, gamma0])

Defines the gamma value for the triplet. For Gibbs, you must specify both the gamma value in the lambda=1 state (gamma1) and in the lambda=0 state (gamma0). For Sander, only gamma1 is used.

windows

Specifies that a windows simulation will be carried out. Specify either SLOW_GROWTH or WINDOWS for a free energy simulation.

Note: This command only available in GIBBS runs. For MINMD and SANDER, it will be ignored.

Options Defined:

start_lambda	width	equilibrate	collect
ti	nodoublewide	increase	decrease

start_lambda

START_LAMBDA = almda.

Specifies the starting value for lambda.

width

WIDTH = almdel.

Defines the fixed width of each window if Dynamically Modified Windows (DMW) is not specified. Ignored if Dynamically Modified Windows is used.

equilibrate

EQUILIBRATE = nstmeq.

Defines the number of steps of equilibration at each window.

collect

COLLECT = nstmul.

Defines the number of steps of data collection at each window.

ti

Specifies that thermodynamic integration, rather than free energy perturbation, will be used to calculate free energy differences.

Default: The free energy perturbation (FEP) approach to calculating the free energy differences will be used.

nodoublewide

Specifies that double-wide sampling will not be performed.

Default: double wide sampling is performed.

increase

Specifies that lambda will increase with time (0->1)

decrease

Specifies that lambda will decrease with time (1->0)

Appendix I:

Sample Script To Perform Minimization

The following script carries out minimization in the presence of restraints read from the file "min.noe". Simply remove the "change" and "nmr" commands to carry out standard unrestrained minimization.

(Note that the examples in this manual and several more can be found in the ../interf/examples directory in the standard Amber/Interface distribution. The Amber scripts are named *.example).

```
! This is a sample input file to carry out minimization with restraints.
!
! This particular script both sets up the input file and runs the
!   program. If all you want to do is set up the min.in file, remove
!   the run... command at the end of this file. Then the appropriate
!   input will be in min1.inp.
!
! set up the minimization run

      sander

! assign vr, which is the version of this run.

      assign vr = 1

      info_off
      minimize/steep=100/steps=2000/rms_energy=1.0
      title = "minimization sample"

      time_limit = 9999999.0
      read/formatted
      dielectric = 1r
      pairlist / update = 50 / cutoff = 8.0
      report/steps = 100

! Set the restraint weights during the minimization to 0.1

      change REST /from= 0.1/to= 0.1

! The restraints are in file "min.noe", in the &rst namelist format
! described in the Amber manual.

      nmr/disang = "min.noe"           \
        /listin = min<vr>.bdv         \
        /listout = min<vr>.edv

      generate /namelist /output=min<vr>.inp

! Now run the minimization. Note that any filename that contains a
! forward slash (/) must be enclosed in double quotes.

      run \
        /input = min<vr>.inp           \
        /param = "../parm.prm"         \
```

Appendix II:

Script To Perform MD With A Restraint

The following script provides a simple example of how to impose a restraint when running an MD simulation. A simple MD simulation without any added restraints could be run by commenting out (or deleting) the two restraint instructions.

```
! A simple sample input file for running MD with a single added
! distance restraint.
!
! If you just want this script to generate the appropriate input file
! and not actually run sander, remove the "run" command.
!

assign version = 1

    sander
    md
    info/off
    title = "sample input to run md with distance restraint"

! Set up the stanard run control parameters:

    time_limit = 99999999.0
    read /formatted
    steps = 10000
    temp / target=300.0 / initial=300.0 / constant=temp / coupling=single
\
    / tau_solute=0.20 /tau_solvent=0.20
    pbc / pressure / target=1.0 / tau=0.6

    timestep = 0.002
    pairlist / update=50 / cutoff=8.0
    dielectric = 1.0

    report / steps=100

! Shake all bonds.

    shake / all / tolerance = 0.00005 / retry=5

! Archive the coordinate every 500 steps:

    archive /coord = 500

! Set up the distance restraint between atoms C1 in residues 1 and 2.
! Change the distance from 7A to 4A over the first 5000 steps, then keep
! it fixed at 4A for the remainder of the simulation. Use a force constant
! of 100 kcal/mole throughout the run.

    restraint /at1=1:C1 /at2=2:C1 \
              /step1 = 0 /step2 = 5000 \
              /r1=(0.0,0.0)/r2=(7.0,4.0)/r3=(7.0,4.0)/r4=(20.0,20.0)\
              /k2=100.0 /k3=100.0

    restraint /at1=1:C1 /at2=2:C1 \
              /step1 = 5001 /step2 = 0 \
```

```
/r1=(0.0)/r2=(4.0)/r3=(4.0)/r4=(20.0)\  
/k2=100.0 /k3=100.0
```

```
! Generate the formatted file corresponding to the above commands, which  
! can be used as input to SANDER:
```

```
generate /namelist / output = sander-<version>.inp
```

```
! Now run the program;
```

```
run \  
  /input      = sander-<version>.inp           \  
  /param      = "../parm.prm"                 \  
  /coord      = "min.rst"                     \  
  /output     = sander-<version>.out           \  
  /restart    = sander-<version>.rst           \  
  /inf        = sander-<version>.inf           \  
                                              \  
clearall
```

Appendix III:

Sample Script for MD/NOE Refinement

The following script demonstrates the simplicity with which a series of MD/NOE refinement runs can be set up using the AMBER/INTERFACE. Standard MD simulations could be performed by removing the “nmr” and “change” commands.

```

! This is a sample input file to carry out md with restraints.
! we run a series of md simulations with different random number
! seeds (to generate an envelope of structural information).
!
! Run 30 simulations; change the random number for each. Each
! simulation is only 10000 steps (20ps).
!
!
! set up the md run

do iloop = 1,30
    sander
    md

! assign vr, which is the version of this run. Make it the same as iloop.
! all the files created for this iteration of the loop will be md<vr>.*

    assign vr = iloop

! assign the changing (on each loop) value of the random seed

    random_seed = 71200 + iloop

    info/off
    title = "md.sample"
    time_limit = 9999999.0
    read/formatted
    dielectric = 1r
    pairlist / update = 50 / cutoff = 8.0
    report/steps = 1000

! -----
! Set the parameters specific to MD:

    steps = 10000/runs=1

! Use shake; Use standard temp. coupling with a single scale factor

    shake/toler=0.0005
    temp/target=300./constant=temp/tau_solute=0.04/initial=300.0 \
        /coupling=single/dtemp=10.0

    timestep = 0.0020
    com/remove/steps = 1000

! -----
! Set the parameters that govern the refinement variables

! ramp the TEMP up from 300 to 900 over the first 2000 steps, then
! keep it at 900 for steps 2000-> 8,000. Then bring it back down to
! 300 over the final 2000 steps.

```

```

change TEMPO /from= 300.0/to= 900.0 /step1= 1/step2= 2000/every=50
change TEMPO /from= 900.0/to= 900.0 /step1= 2001/step2= 8000/every=50
change TEMPO /from= 900.0/to= 300.0 /step1= 8001/step2=10000/every=50

! Raise the restraint weights from 0.1 to 36.0 over the first 1000 steps
! and then keep them there.

change REST/from= 0.1/to= 36.0 /step1= 1/step2= 1000/every=50/geom
change REST/from= 36.0/to= 36.0 /step1= 1001/step2= 0

! The following, if uncommented, would force time-averaged refinement

! time_average = distance/tau=10.0/power=3/pseudo_force
! time_average = torsion /tau=10.0/power=-1

! The restraints are in file "min.noe", in the &rst namelist format
! described in the Amber manual. Alternatively, if they were in a file
! rest.noe in Amber/Interface format (i.e. "restraint" commands), you would
! uncomment the first line below, and the restraints would be converted to
! &rst format and placed in the file min.noe. On subsequent runs, you would
! remove the redirect command and read the restraints from min.noe
! directly.

! redirect = rest.noe

nmr/disang = "min.noe" \
  /listout = md<vr>.edv

generate /namelist /output=md<vr>.inp

! Now run the md. Note that any filename that contains a
! forward slash (/) must be enclosed in double quotes.

run \
  /input = md<vr>.inp \
  /param = "../parm.prm" \
  /coord = "min1.rst" \
  /output = md<vr>.out \
  /restart = md<vr>.rst \
  /inf = md<vr>.inf

! The options, etc. must be cleared after each loop iteration:

clearall
end do

```

Appendix IV:

Sample Script Performing Torsional Driving

The following script demonstrates the flexibility and power of the AMBER/INTERFACE front end. It first runs an MD simulation, forcing a specified torsion through 360 degrees, and saving MD snapshots along the way. Subsequently, each snapshot structure is recalled and minimized, with the resulting energy saved in a separate file. Note that AMBER/INTERFACE not only makes it simple to perform this simulation, but also makes it very easy to follow the input. The script has been well-annotated to aid the novice user. It would, of course, look more compact without the many comments.

```
! This is an "interface" input file which does:

!   1) An md simulation of the molecule inhib, driving the
!       sidechain torsion 0 -> 360 degrees. Sample for 20000 steps every
!       10 degrees, and archive the coordinates every 1000 steps.
!
!   2) A series of minimizations, starting with each of the archived
!       sets of coordinates. The minimized coordinates are stored in a
!       second archive file, and the corresponding minimized energies are
!       logged to a separate file.
!
! The program SANDER is used. It includes the ability to incorporate the
! required restraints.
!
! Author: David A. Pearlman

! set up the MD sampling run:
! -----

    sander
    md
    title = "searching about the sidechain torsion for INHIB"
    time_limit = 999999.

! read the coordinates

    read / formatted

! do 37 runs of 20000 steps each.

    steps = 20000/runs=37
    temp / target=300. / constant=temp / tau_solute=0.4 / initial=300.0

! (note we aren't using SHAKE)

    timestep = 0.001

! (effectively infinite cutoff):

    pairlist / update=99999 / cutoff=30.0
    com/remove / steps = 1000
    dielectric = 4r
```

```

report / steps=5000
archive / coord=1000

! We use the CHANGE and RESTRAINT commands to effect the torsional driving
! restraints. The NMR command is used to cause the current values of the
! restraints to be printed in the output file at the beginning
! and end of the run.

nmr/listin=POUT/listout=POUT

! bring the restraint in slowly over the first 20000 steps. After that,
! keep it fixed at its full value.

change REST/from=0.0001/to=1.0/step1=0/step2=20000/every=50/geometric
change REST/from=1.0/to=1.0/step1=20001/step2=0

! Keep the temperature fixed at 300K the entire run (this command not
! really needed here; included so the user sees how easy it is to specify
! the temperature changes:

change TEMP0/from=300.0/to=300.0/step1=0/step2=0

! The following set up and modify the target value of the restraint to
! be placed on the side-chain torsion:

! Keep the "alpha" torsion fixed about trans the whole run (a flat
! bottomed well flat between 150-210 degrees).

restraint /at1=1:C2' /at2=1:N1 /at3=1:C7 /at4=1:C8 \
          /step1 = 0 /step2 = 0 \
          /r1=120.0 /r2=150.0 /r3=210.0 /r4=240.0 /k2=50.0 /k3=50.0

! For the first 20000 steps (while restraint weights are slowly being
! increased), the "beta" sidechain torsion is fixed at 0

restraint /at1=1:N1 /at2=1:C7 /at3=1:C8 /at4=1:C9 \
          /step1 = 0 /step2 = 20000 \
          /r1=-30.0 /r2=0.0 /r3=0.0 /r4=30.0 /k2=50.0 /k3=50.0

! for the next 720000 steps force the "beta" torsion 0-360, changing
! the target value by 10 degrees every 20,000 steps. Use a restraint
! weight of 50.0

restraint /at1=1:N1 /at2=1:C7 /at3=1:C8 /at4=1:C9 \
          /step1 = 20001 /step2 = 740000 /every=20000 \
          /r1=(-30.0,330.0) /r2=( 0.0,360.0) \
          /r3=( 0.0,360.0) /r4=(30.0,390.0) \
          /k2=( 50.0,50.0) /k3=(50.0, 50.0)

! creat md1.inp, the formatted input file corresponding to the above,
! which is to be read by "SANDER":

generate /namelist /output = md1.inp

! do the md; inhib.prm and inhib.equil are the parameter and coordinate
! file for this simulation.

run / input      = md1.inp          \
  / output      = md1.out          \
  / param       = inhib.prm        \
  / coord       = inhib.equil      \
  / restart     = md1.rst          \
  / arc_coord   = md1.arc          \
  / inf         = md1.inf          \

```

```

! -----
! -----
! Now we've carried out the md sampling portion. Call "CLEARALL" to
! clear all the registers, then start the second leg of our
! simulation: Sequential minimizations.
!
    clearall

! scratchname gives a suffix appended to the names of temporary files
! created when using commands such as direct, getarchive, putarchive,
! etc. Setting this to a different value for each Amber/Interface
! job run from the same directory allows you to safely run multiple
! jobs from the same directory at the same time (not needed if only
! one job at a time will be run).

    scratchname = "srchl"

! convert the archive file from md to direct access format (for faster
! access)
!
    direct / in = mdl.arc / out= mdl.direct / restart = mdl.rst
!
! The following do loop cycles through all the structures that were
! archived during the md phase. In turn, each is minimized (still
! with the appropriate restraint on the "beta" sidechain torsion)
! then saved to a direct access archive file.

!
    do i = 1,36

!
! echo the current set being minimized to the user. Note the use of the
! variable construct <i>, which is replaced by the value of "i" before
! the string is echoed...
!
        echo "Starting set # <i>"

! GETARCHIVE retrieves the coordinate set indicated.

        getarchive / archive = mdl.direct \
                  / output  = mdl.xyz    \
                  / number  = <i>

! set up the minimization (turn off info messages)

        info/off

        sander
        minimize /steep=100 /steps=9999 /rms_energy=0.10
        dielectric = 4r
        read /formatted
        title = "INHIB minimization"
        pairlist /update=99999 /cutoff=30.0
        report /steps=100

! -----
! Define restraints. The "alpha" sidechain torsion is always
! fixed about at trans. The "beta" torsion rotates 0-360.

! torsion of interest ("beta") was fixed @ 0 first 20,000 steps
! For 20,000-740,000 incremented from 0-360 by 10 degrees every 20,000
! steps

```

```
! Each archived structure (incremental value of DO-loop variable i)
! corresponds to 1000 steps.

    if (i.le.20) then
        gassign targ2 = 0.0

    else if (i.le.740) then
        gassign targ2 = ((i-20)/20) *10.
    end if

    restraint /at1=1:C2' /at2=1:N1 /at3=1:C7 /at4=1:C8 \
        /r1=120.0 /r2=150.0 /r3=210.0 /r4=240.0 /k2=50.0 /k3=50.0

    restraint /at1=1:O3 /at2=1:C7 /at3=1:C8 /at4=1:O4 \
        /r1=targ2-30.0 /r2=targ2 /r3=targ2 /r4=targ2+30.0 \
        /k2=50.0 /k3=50.0

! -----

! Create the formatted file which will be used as input to SANDER to
! perform the desired minimization:

    generate /namelist /output = minxx_vac.inp

! do the minimization

    run / input    = minxx_vac.inp    \
        / output  = minxx_vac.out    \
        / param   = inhib.prm        \
        / coord   = mdl.xyz           \
        / restart = minxx_vac.rst
```

```
! PUTARCHIVE archives the minimized output coordinates in the named
! direct access archive file (min1.arc). The coordinates are archived in
! position "i" of the file.

    putarchive / input = minxx_vac.rst \
              / archive = min1.arc   \
              / number  = <i>

! LOGENERGY logs the energy, and an associated specified sequence number to
! a named file (in this case, min1.nrg).

    logenergy / infile = minxx_vac.out \
             / logfile = min1.nrg     \
             / sequence = <i>

! cleanup unneeded files:

    delete minxx_vac.out minxx_vac.rst minxx_vac.inp md1.xyz

! clear all memory at the end of loop iteration, since no variables
! are carried over between iterations (except loop index). Note that
! this should be done either just before the end do at the end of every
! loop iteration, or immediately at the beginning of each loop iteration

    clearall

end do

! at the end of the run, the min1.arc file containing the archived
! minimization coordinates (generated from putarchive commands) will be
! direct access and will contain one extra line at the top of the file.
! To convert this to a sequential AMBER-format sequential access file,
! (which can subsequently be used in any program expecting an Amber format
! archive file) use the SEQUENTIAL command. The /delete qualifier removes
! the extra line in the file.

    sequential /in = min1.arc /out = min1.archive /delete = 1

    delete md1.direct
```

Appendix V

Sample Script to Run Gibbs

The following script sets up and runs a GIBBS simulation. The many gibbs options have been reduced to a manageable set of intuitive commands. This particular script sets up a Dynamically Modified Windows (DMW) simulation.

```

!
! This is gibb.example.
! It is a sample UC file which can be used with the
! AMBER/INTERFACE front-end to Amber to run the GIBBS program.
!
! Author: David Pearlman
!
  gibbs
  title = "Test of interface program for gibbs"

! Set up the stanard run control parameters:

  time_limit = 20000.0
  read / velocities / box / formatted
  steps = 20 / runs = 1
  temp / target=300.0 / initial=300.0 / constant=temp / coupling=single \
    / tau_solute=0.20
  pbc / pressure / all_solute_nb / target=1.0 / tau=0.6

  timestep = 0.001
  pairlist / update=50 / cutoff=6.0
  dielectric = 1.0

  report / steps=200

! Shake all bonds, and calculate the free energy contribution due to
! constrained bonds changing with lambda:

  shake / all / tolerance = 0.00001 / retry=0
  constraints / energy / tol=(0.00001,0.0001) / move=fewer

! Set some control parameters specific to free energy calculations:
! Do only the vdW changes on this leg. Include all intra-group
! contributions.

  decoupled = vdw
  mixing / vanishr = 0.0
  intragroup / non_bond / 1_4 / internals

! Choose the method (windows) and request dynamically modified adjustment:

  windows /start_lambda=1.0 /decrease /equil=200 /coll=200

  dmw /points=8 /correl=0.8 /target=0.01 /initial_dlambda=1.0D-5 \
    /min_dlambda=1.0D-6 /max_dlambda=1.0D-2 /reset=0.1

! Generate the formatted file corresponding to the above commands, which
! can be used as input to GIBBS:

```

```
generate / output = gibb_form.inp
```

```
! Now run the program. Note we must enclose any filenames containing  
! slash characters with double quotes:
```

```
run \  
  /input      = gibb_form.inp          \  
  /param      = "/usr5/dap/amber4/demo/methane/me.prm"  \  
  /coord      = "/usr5/dap/amber4/demo/methane/me.crd"  \  
  /output     = me.out                  \  
  /restart    = me.rst                  \  
  /inf        = me.inf                  \  
  /window_stat = me.sta
```

Appendix VI

Creating Customized Redirect Files

It is frequently the case that you run many of your simulations with the same values for many of the variables controlling the simulation. In this case, it may be helpful to create a file containing these definitions which can then be used to create simulation scripts. This file can be cleanly inserted into an Interface script using the “redirect” command.

For example, suppose you run many of your MD simulations under the following conditions

! This is mymd.script:

```
md

info/off
time_limit = 9999999.0
read/formatted
dielectric = 1r
pairlist / update = 50 / cutoff = 8.0
report/steps = 1000

! Use shake; Use standard temp. coupling with a single scale factor

shake/toler=0.0005
temp/target=300./constant=temp/tau_solute=0.04/initial=300.0 \
/coupling=single/dtemp=10.0

timestep = 0.0020
com/remove/steps = 1000
```

You could create a file, “mymd.script”, containing the above commands. Then to run an MD simulation, you would simply create the following Interface script:

```
sander

! read standard MD conditions from mymd.script:
redirect = mymd.script

! assign vr, which is the version of this run.
assign vr = iloop

steps = 10000/runs=1
generate /namelist /output=md<vr>.inp

! Now run the md. Note that any filename that contains a
! forward slash (/) must be enclosed in double quotes.

run \
/input      = md<vr>.inp           \
/param      = "../parm.prm"       \
/coord      = "min1.rst"          \
/output     = md<vr>.out          \
/restart    = md<vr>.rst          \
/inf        = md<vr>.inf          \
```

Appendix VII

Extracting PDB Files from An Archive File

MD simulations are often run to sample conformation space. In such a simulation, the coordinates are periodically archived (see the “ARCHIVE” command), creating a large file containing many coordinate sets. Subsequently, one may wish to analyze these coordinates using other programs. Many programs work best with “PDB” format coordinates. The following Interface script shows how one can easily create such PDB coordinates from the archive file.

```
! This is a sample amber/interface script that will create
! multiple pdb files from the output coordinate archive file created
! during an amber md run. Note that by changing the indices on the do
! loop or by adding the appropriate goto statements, you can fully
! describe the pdb files you would like.
!
! This script will create a series of pdb files named "frameN.pdb"
! (e.g. frame5.pdb, frame10.pdb, etc.).
!
! Author: David A. Pearlman
! Date: 8/92

! Create a direct access file from the sequential access file generated
! during Amber. This only needs to be done once:

direct /input = mdl.arc /output = mdl.direct /restart = mdl.rst

do i = 5,100,5

! A goto line of the following type can be used to omit certain sets:

    if (i.eq.15 .or. i.eq.30) goto skip

! Get a specified coordinate set from the direct access archive file:
! The coordinate set will be in Amber "restart" format. Note that
! if the system was run with periodic boundary conditions on, you should
! add the flag "/periodic" to the following statement.

    getarchive /archive = mdl.direct /output = tmpfil.rst /number = i

! Create a pdb file from the restart file using pdbgen:

    pdbgen /input = tmpfil.rst /output=frame<i>.pdb /param=parm.prm

! Delete the restart-format file, which we don't need now that we've
! generated the pdb file

    delete tmpfil.rst

    skip:
    clear
end do

! All done. Delete the direct access archive file.
```

```
delete mdl.direct
```

Appendix VIII

Modifying the Standard Amber/Interface

In most installations, the scripts for the standard Amber/Interface will be located in a common directory that you cannot (or should not) modify. As you become comfortable with the Interface, you may desire to make modifications/additions to the Amber/Interface. This can be done without having to alter the standard scripts, by using the “gettemplate” command.

For example, suppose you have written a script, `setalias.def` defining the command “`setalias`”, which allows you to create an alias name for a chosen command name:

```
command = setalias
  defhelp SETALIAS (command_name,alias_name)
  defhelp
  defhelp This command allows an alias to be created for a command name\
  from the UC file.
  defhelp alias_name will be a valid alternative name for\
  command_name.
  markmem

  arguments (2a) / names = (command_name, alias_name) /equivalence=^
  alias <alias_name> <command_name> /command

  clear/mark
end command
```

To include this command in your personal version of the Amber/Interface, you would then include the following command at the beginning of your Interface script:

```
getttemplate def = setalias.def # temp=setalias.dat # overwrite
```

The `setalias` command would then be appended to the list of commands recognized by the Amber/Interface for the duration of the session, and you could subsequently change command aliases using `setalias`, e.g.

```
setalias (time_limit, maxtime)
```

As another example, suppose you wished to replace an existing command definition with a new version you had modified. Again, this can be done without altering the “master” directory. All you need to do is specify the “/replace” qualifier when you define the command. Then when you issue the “gettemplate” command, your version of the command will take the place of the default distribution version.

You could define a new version of the “`random_seed`” command, which was placed in file `newran.def`:

```
command = rand*om_seed /replace
  defhelp RANDOM_SEED = ig
```

```
defhelp
defhelp Assign ig, the seed for the random number generator. The \
        default is usually acceptable. In this new version do not \
        allow an even number to be used as the seed.

arguments (i)/name=ig
if (mod(ig,2).eq.0) \
    stop "Error: An even number should not be used as random seed."
end command
```

Then you could force replacement of the standard `random_seed` definition by reading this new definition:

```
getttemplate def = newran.def # temp=newran.dat # overwrite
```

See the Interface manual for more information on writing an Interface script.