THE Crowd PROGRAM.
Version 1,0

A Program for NMR Relaxation Calculation

Copyright : M.A.Delsuc, T.E. Malliavin
Centre de Biochimie Structurale
Faculté de Pharmacie
34060 MONTPELLIER FRANCE

THE Crowd PROGRAM. Version 1.0

This program has been developed in Montpellier with the help of Marc-André Delsuc and Jean-Luc Pons. Please contact :

T.E. Malliavin
Centre de Biochimie Structurale
Faculté de Pharmacie,
15 avenue Charles Flahault
34060 Montpellier Cedex FRANCE.

E-Mail : terez@cbs.univ-montp1.fr (Internet)
Tel : (33) 67 04 34 36 (In case of emergency)

PRESENTATION

The Crowd program is a program designed for the NMR relaxation intensity simulation (nOe, rOe,...) and the iterative inter-atomic distance calculation using the experimental NMR intensities. The program was built using the Gifa command parsor and internal langage. Thus, as in Gifa, a complete macro language permits to builds sophisticated processing. The Graphic User Interface is fully designable and programmable by the user.

The program currently runs on HP workstations. The graphics can be displayed on X-Windows terminals.

This version V1.0 is not stabilised yet, this distribution is still a beta version. This version is distributed under a licence. There is no fees for academic laboratories, however the licence requires that you refer to Crowd in any published work which depend in some manner on the Crowd program.

This program can be downloaded by anonymous FTP on Internet, the home site is the following address : ftp://tome.cbs.univ-montp1.fr/pub/crowd_v1.

Classical processing includes :

- •Manipulating the differents NMR relaxation objects (coordinates, distances, relaxation rates, ...) by a set of modular ommands.
- •Availability of several motion models: isotropic-rigid, Lipari-Szabo, local correlation time models.
- •Aptitude to read free-format coordinate files.
- •Aptitude to define hydrogen CH2 and methyl group, by residu name, and group atom names. This allows calculation on various kind of biomolecules: proteins, DNA, RNA, glycoproteins...

A Complete set of Macro including:

- •simulation of nOe, rOe and off-resonance ROESY intensities from a free-format coordinate file.
- •implementation of the algorithms proposed in the MARDIGRAS and IRMA programs.

The User Interface is characterised by :

- •Most command are available through a graphic interface
- •Interactive commands for beginners as well as fast entry mode for advanced users
- •Complete on-line HELP.
- •Comprehensive manual.

•Versatile command parser which permits to build macro commands with alias, tests, local and global variables, loops, graphic interaction and display capabilities, support for data-bases and associative arrays.
•graphical interface completely modifiable and definable by the user.
•Possibility to run in interactive mode as well as Batch mode (display less).
•The operating system is fully available from Crowd.

FIRST CONTACT WITH THE PROGRAM.

Entering the program

First insure that the program have been correctly installed from the distribution kit, contact your system manager or your local expert.

You enter the program by typing `crowd` on your favourite system (don't panic, just type EXIT to get out of it!) . The program should give you a short greeting, the size of the static buffers, and then respond in the text window with the prompt :

```
Crowd>
```

If every thing as been set up correctly, you should also see a horizontal menu bar showing off. To get the graphics, you should make sure that the environment variable DISPLAY has been correctly set to point on your current terminal (with the command setenv DISPLAY when running UNIX). If no X display is available, Crowd will tell you that no graphic will be available, but will start correctly however.

Principe of operation

There are two ways of issuing commands to Crowd. The first one is to type commands directly in the text window, at the prompt level. Such commands are of two kinds, built-in commands, directly executed by Crowd, and macros, which are command files written in the Crowd command language. From the user level, there is little difference between the two kind of commands.

The other method consists in hitting a button in the graphic interface. The graphic interface consists in sets of related commands, arranged in vertical button boxes, each such boxes being called by clicking the associated button in the menu bar. Each button box can be opened and closed at will.

The two methods are strictly equivalent, graphic buttons are internally associated to commands (or list of commands). One day, you will eventually learn how to do your own buttons.

Graphic windows can be closed at any time, simply by clicking on their close-box.

Entering Parameters

You are now ready to enter your first command. If you hit the `About CROWD` button in the `File` button box (the box will show off if you hit the `File` button in the menu bar), you will get a short introduction.

Entering

```
Crowd> HELP
```

at the prompt level will give you the same message. The command HELP is here in uppercase, it can actually be entered also in lowercase, or even in a mixture of upper and lower case letters. In the whole manual, we will use the rule that every built-in Crowd command is given in uppercase. Macros, however have to be typed exactly as defined (this is due to the UNIX file-system), they will always appear in the manual as lower case.

Typing

```
Crowd> HULP
```

will give you an error message, since there is no command nor macro named hulp (at least on the distribution kit)!

The HELP command can also be used with a parameter :

Crowd> HELP timer

Some command may require one or several parameters, you will be prompted for such parameters, either in graphic dialogue boxes if the command was entered from a menu, or at the text terminal if the command was typed from it. When entering the prompt level, it is also possible to put the parameter directly after the command on the same line.

For instance, let us try the TAUC command which permits to define the molecule global correlation time.

Typing

```
Crowd> TAUC
```

will prompt you for the global correlation time value currently defined, just typing return will leave the value unchanged.

Typing

```
Crowd> TAUC 5.0
```

Will set the current global correlation time value to 5.0 nsec.

On the other hand, clicking on the Tauc button, in the **Param** menu will pop-up a dialogue box which realises the same thing.

BASIC GRAPHIC USER INTERFACE

The standard graphic interface is characterised by a set of menus, button boxes, and dialogue boxes used for graphic user interaction.

Menu Set-up

The basic-menu graphic user interface is installed in the standard start-up procedure. It is installed whenever the startup.g macro is executed. This macro loads the basic environment.

However it is completely optional, and Crowd can work without any menu bar (even without any graphic window !), or with a completely different graphic environment.

This user interface is actually quite easy to modify (see below in the manuel) so if you are really lacking a function in the user interface, just add it !

The environment macros are found in the /usr/local/crowd/macro directory.

If a menu is buried below other windows, clicking on the menu name will pop the menu again.

Basic environment

The following menus are usually present whenever the graphic interface is activated, they are independent from the kind or work currently in progress.

If the graphic is missing, try typing the following command basic_menu.g, which executes the macro which loads the default menu-bar, from which you can add all the specific menus.

Entries followed by 3 dots (...) indicates that some input will be asked to the user by the command.

Info menu

This menu holds the basic commands for getting informations on the program, and quitting the program.

About CROWD

Will print introduction text, giving the some ideas on the Crowd program, as well as some hints on how getting more information or help.

This button is equivalent to the HELP command without parameter.

Help...

Will prompt you for a command/macro name and will display the help associated with that command/macro.

This button is equivalent to the HELP command with a parameter.

APropos...

Will prompt you for a word, and will show the name of all the commands/macros that have this word in their help file (case insensitive).

This button is equivalent to the apropos macro.

Documentation

This button launches the Mosaic program, with an hyper-text version of the current manual.

Config

Will give all the details on the internal configuration of the program. You will have information on the current version number and the licence of the program, as on the static buffer sizes. These values are hard-wired in the program at compile time.

This button is equivalent to the CONFIG command.

bug report

Permits to edit and mail to me (terez@cbs.univ-montp1.fr) a bug report.

Quit CROWD

Will quit the Crowd program loosing all data in memory not saved with the output commands, and removing the journaling file. If you want to keep the journaling file (a file called crowd.log in your home directory, which holds all the command issued during the Crowd session), you will have to type in the QUIT command directly.

This button is equivalent to the QUIT N command.

Topology menu

This menu permit to manipukate the spin types, which are implemented into the program as a chained list for each diffrent residu.

Define...

Will add a spin type to the spin type chained list. For spin group of equivalent atoms, the name given to the spin will be the name of the first atom in the group, where the first character will be replaced by a 'Q'.

This button is equivalent to the ADDTYP command.

Delete...

Will remove a spin type to the spin type chained list.

This button is equivalent to the DELTYP command.

List...

Will list the atom type defined in a given residu name.

This button is equivalent to the LSTTYP command.

Initialize

Will initialize the atom type chained list.

This button is equivalent to the INITYP command.

...XPLOR amino acids

Will add a menu permitting definition of XPLOR amino acid hydrogen type (file topallh6x.pro)

... XPLOR nucleotides

Will add a menu permitting definition of XPLOR nucleotide hydrogen type (file topallhdg.dnc)

... Unix

Will add a menu permitting Unix-like commands.

Molecule menu

This menu holds the commands permitting to handle a molecule (residus and coordinates). The molecule residus are defined as an array, the molecule coordinates as a chained list.

Define res...
: Will permit you to define the residu name of a given residu number.
This button is equivalent to the ADDSEQ command.

Delete res....
: Will undefine the residu name for a given residu number (the residu name is set to 'UNK')
This button is equivalent to the DELSEQ command.

List res
: Will list the molecule sequence
This button is equivalent to the LSTSEQ command.

Initialize res
: Will undefine the residu name, whose the number is smaller than the number of residus. The undefined residu name is set to 'UNK'.
This button is equivalent to the INISEQ command.

Number of residus...
: This button will permit you define the number of residus in the molecule.
This button is equivalent to the NB_RES command.

Define crd...
: Will permit you to add an atom to the atom chained list, and to define its type and its (X,Y,Z) coordinates.
This button is equivalent to the ADDCRD command.

Delete crd...
: Will permit you to remove an atom from the atom chained list.
This button is equivalent to the DELCRD command.

List crd...
: This button will list the atom names, types and coordinates of a given residu number.
Equivalent to the LSTCRD command.

Initialize crd
: This button will initialize the coordinate chained list.
Equivalent to the INICRD command.

Read PDB File...
: This button will read a PDB free-format file.
Equivalent to the rdpdb.g macro.

Param menu

This menu holds all the commands defining parameters and experimental intensities used in intensity simulation iterative distance calculation.

Motion model...
: Define the motional model used during the relaxation calculation. This button is equivalent to the `MOTION_MODEL` command.

Tauc...
: Define the molecule global correlation time. Equivalent to the `TAUC` command.

Internal motion parameters...
: Will add a menu permitting to define the internal motion parameters: local correlation times and order parameters.
Equivalent to the `env_lipa.g` macro.

Hyd. reson. freq...

Define the hydrogen resonance frequency. Equivalent to the `OMEGA` command.

Nb of mixing times...
: Define the size of the mixing time array. Equivalent to the `NB_TMIX` command.

Define mixing time...
: Define the value of a given mixing time. Equivalent to the `TMIX` command.

List mixing times...
: List the mixing time array. Equivalent to the `LSTMIX` command.

...Exp.int.
: Will add a menu permitting to handle experimental intensities. Equivalent to the env_inte.g macro.

Output menu

Theo intensities...
: Will write to a given output file the theoretical intensities. Equivalent to the `WR_INT` command.

Build-up curve file...
: Calculate and write to filename the intensities corresponding to the atome pairs, for which an experimental intensity was read, at the mixing times defined in the mixing time array. The build-up file looks like a 2D NMR data matrix, where each column is a build-up curve between a given atom pair. The file format is binary and can be read by the command READH in Gifa. The order of the atom pairs is given by the command LSTINTE.
This button is equivalent to the WRIBLDP command.

PDB coord file...
: Write a PDB coordinate file from the current atom chained list. This button is equivalent to the `wr_pdb` macro.

DG file...
: Write a Distance Geometry input file from the current interatomic distances. Equivalent to the macro `wr_dg`.

Current iteration report
: Write a report of the current iteration, giving a list of all atom pairs, for which an experimental intensity has been read. This output line format is: res_name1 res_no1,atn1, res_name2, res_no2,atn2,disti, expint, rate,theoint,dstc, where res_name are residu names, res_no are residu numbers, atn are atom names, disti is the initial inter-atomic distance, expint is the experimental intensity, rate, theoint and dstc are the current relaxation rate, theoretical intensity and interatomic distance.

Pente...
: Calculate the interatomic distances from the experimental intensities, by using the initial slope approximation: intensity is proportional to the inverse of the distance sixth power. Write the interatomic distances to the journal unit.
This button is equivalent to the PENTE command.

Look...
: Shows the distance, the relaxation rate and the intensity between a given spin pair. This button is equivalent to the LOOK command.

Calculation menu

CORMA...

Calculata the theoretical NOE intensities from a PDB coordinate file. Equivalent to the `corma` macro.

ROE Intensities...
: Calculate the ROESY intensities from a PDB coordinate file. This button is equivalent to the `roesy` macro.

Off-ROESY Intensities...
: Calculate the off-resonance-ROESY intensities from a PDB coordinate file. This button is equivalent to the `offroesy` macro.

NOE Build-up curves...
: Calculate and write the NOE build-up curves from a PDB coordinate file. This button is equivalent to the `calcbldp` macro.

R factor...
: Calculate two R factors and a mean error between theoretical and experimental intensities. Equivalent to the calc_r command.

MARDIGRAS...
: Will perform the interatomic distance iterative calculation using the algorithm proposed in MARDIGRAS. Equivalent to the `mardigras` macro.

IRMA...
: Will perform the interatomic distance iterative calculation using the algorithm proposed in IRMA. Equivalent to the `irma` macro.

OPERATION PRINCIPLE

As you have seen if you have been through the primer, Crowd is an interactive program : it waits for commands from the user. Some commands have parameters some don't; some commands may even have a variable number of parameters depending on some context value (it is a so-called context-dependent grammar).

Different kind of commands

There are two kinds of commands in Crowd : regular commands which actually execute some action (such as CRD2DST for inter-atomic distance calculation from the atom coordinates), the others are commands which may execute an action but will also change the value of internal parameters of the program (such as TAUC for global correlation time). Such commands will be called contexts.

A good example of the distinction between a context and a regular command is given by the context OMEGA which defines the default value used for hydrogen resonance frequency, as compared to the related CREL_NOESY command which calculates the NOESY relaxation rates from the inter-atomic distances, using the hydrogen resonance frequency defined by the command OMEGA.

Most of the behaviour of the program depends on these internal parameters (contexts). Another example of a context is MOTION_MODEL which describes the kind of motion model used for spectral density function calculations. The effect of the command CREL_NOESY depends also on the state of the context MOTION_MODEL.

Most contexts have associated Crowd variables which permits to evaluate its value during macro execution. See the chapter on variables for details.

Commands can be issued alone, in which case, the command will prompt you for all the values it needs, proposing default ones. Default values may be kept by hitting the <enter> key, or may be changed by typing

new ones. You can also type the parameters on the same line than the command; the effect is then immediate. When using in-line entry, the different items will be separated by blanks.

Several commands can even be typed on a single line, separated with a blank. For instance, you could either type :

```
Crowd> TAUC 3.0
Crowd> CREL_NOESY
```

or, on a single line :

```
Crowd> TAUC 3.0 CREL_NOESY
```

The effect would be the same.

UNDERSTANDING THE MEMORY SET-UP

The Crowd program holds all the data in memory. It presents several buffers to store the differents objects it handles:
molecule residus (RES): res(szres)
atom coordinates (CRD) : crd(szat)
inter-atomic distances (DST): dst(szmat)
relaxation rates (RATE): rateh(szmat)
theoretical intensities (INT): inttheo(szmat)
mixing times (TMIX): tm(sztm)
experimental intensities (INTE): intexp(szmat)
internal correlation times (TAU): tau(szmat)
order parameters (S2): s2(szmat)
where szres is the maximum number of molecule residus, szat is the maximum number of atoms and szmat = szat*(szat+1)/2. The current szat, szres and sztm values are given by the command CONFIG.
The other arrays are used for handling chained list, and as working arrays for matrix diagonalisation.

MORE SPECIFIC COMMANDS

PRINT, ALERT, MESSAGE

PRINT *text* will display the content of *text* to the user. ALERT *text* displays the text to user in a graphic box. ALERT will work whenever the graphic has been used once.
MESSAGE is used in macro, it is equivalent to PRINT, except that the string will be output to the user only if no parameters are available on the call line. It is thus nearly equivalent to

```
if (!$arg) PRINT text
```

However, it is different in the sense that the string will be presented in the graphic dialogue box when the macro is called from a menu button.

TEXT FILE INPUT / OUTPUT

Crowd has the capability of reading and writing on ASCII text files.

LOG_FILE CONNECT DISCONNECT

In permanence, all the input generated by the user, and the output generated by the program are journaled in a file called crowd.log. This file can be either kept or removed when exiting the program. At any time the user may redirect the journaling to another file with the command : CONNECT file_name. The program will then create a file called file_name where all the journaling will go. The command DISCONNECT will resume the journaling to the crowd.log default file, and close the previously CONNECTed file. If the command CONNECT is issued while a file is already CONNECTed, the former file will closed, and the new file will be opened.

OPEN CLOSE FPRINT <file_name eof(file_name)

It is also possible to read and write directly into text files. The command OPEN file_name, will open the file file_name for input or output. The command CLOSE will close the file. Ten different such files may be opened at once. An OPENed file may be written with the command FPRINT file_name output. A OPENed file may be read with the <file syntax (see below). When OPEN is issued on an already OPENed file, the file is rewound to the first line. The eof(file_name) function (see below) will have a value of 1 if the end of file was reached during the last <file_name operation and 0 otherwise.

SPECIAL AND GENERAL PURPOSE COMMANDS

HELP

permits to general info on a specific item. HELP with no arguments gives you the list of the commands available. (H or ? can also be used).

EXIT, QUIT, or BYE

are the normal exit of the program. These commands ask you whether you want or not to keep the log file which as been built during the processing and which contains a copy of all the interactive during the session. This file is called crowd.log. On UNIX system, be careful to rename it (mv command) before rerunning Crowd otherwise the saved log file will be erased by the new log file.

^C

The control C key will abort the process in progress and bring you back to the prompt level (on most UNIX machines). This may sometime take a few seconds before executing the abort.

TIMER

is a context that when set to one will activate the display of elapsed and cpu time taken by every command. Useful for benchmarking.

VERBOSE and DEBUG

These 2 contexts will generate verbose output from macro files.

SH

The SH command will send its parameter to the operating system. For instance you can type :

```
Crowd> sh "ls -l /usr/data"
```

You can also type SH alone, this has the effect of creating a sub process at the operating system level (csh is used), you then get back to Crowd by loging out (^D on UNIX).

Using SH, several macros have been created that mimic UNIX : more, rm, ls, vi, pwd, etc... There are also two special editing command : vip will edit in your $HOME/macro directory, and vim in /usr/local/crowda/macro

CD

`cd` *`dir`* permits to change the current working directory to *`dir`*. Equivalent to UNIX. Note that this is very different from :

```
sh 'cd dir'
```

which actually creates a sub-process which executes cd, thus having no effect on the current job.

SYNTAX OF THE CROWD INPUT.

Each command line is composed of a series of "words" separated by blanks. A word can be a command or a parameter. Words are strictly separated by blanks, however a word can be of several nature :

•literal are as typed:

```
tauc
```

•literal string are enclosed within single (') or double(") quotes:

```
'a string'  or "another string" or "yet an'other valid string"
```

•The special character `%` (to be used only as a parameter for a command) takes the place of the default value of the command:

```
OMEGA %
```

•The special string `%%` takes the place of the whole series of all the default values of the remaining parameters of the command :

```
ADDCRD HA % % % %   and  ADDCRD HA %%
```

are equivalent

•variables are substituted before use (more about this later):

```
$variable1
```

•expressions, enclosed within parentheses, are evaluated before use (more about this later) :

```
( 2*(cos(3) + 1))
```

•any of the preceding syntaxes may be preceded with a < symbol. In which case the input is interpreted as a file name, and the parameters takes the value of the next line read in the file (if already OPENed). A complete line is read at each time.

Each of these syntax can be used in any places; however, a single word cannot match a series of words :

These are valid syntax :

```
tauc $x              ; define the global correlation
                     ; time as the variable $x value.
tauc (%+3)           ; add 3.0 to the global tauc value
omega (2*$y +1)      ; define the hydrogen freq as the
                     ; (2*$y+1) value
set cmd = 'tauc'     $cmd 0.5        ; perform tauc 0.5
wr_int ("/usr" // $user // $exp) 0.1 Title   ; write an
                                     ; intensity file
("add" // "CRD")          ; execute the command ADDCRD
set f = test open $f  tauc <$f      ; set the global
coorelation to the value
          ; found in file called test
```

These are invalid syntax :

```
set corr = 'tauc 3.0'  $tauc
```

a single word (here 'tauc 0.5') cannot be used to match several words

```
TAUC 1+2
```

the parenthesis are needed for expression to be evaluated

etc...

A line cannot be longer than 256 characters, otherwise the end the line will be lost. If a longer line is needed, a continuation sign is available : \

e.g.

```
Crowd>  print 'A line cannot be very long'     print \
\Crowd> 'But can be continued'   print 'as many time as
you wish'
```

You can put as many continuation sign as you want, however, these signs are considered as “word”, so they should be blank separated, and cannot be used within evaluated expressions (see below).

VARIABLES

Variables can be used in Crowd, this feature is mainly useful when writing macro commands, however, it can be used at any time in the program. Variable are set (and created if not yet defined) with the command : SET :

```
Crowd> set b = 'hello world'  set c = 3
Crowd> set d := 'hello mom'             ; other syntax,
see below
```

Variables are then just used by prefixing the name with a $:

```
Crowd> tauc $c print $b
```

Variables can be used anywhere a regular entry is needed. Variables are not typed, but contain always a string. The maximum string is currently of 256 characters. The string is interpreted as a number when needed. Variable names are not case sensitive, the name is a maximum of 31 characters long; and must be built from purely alpha-numeric characters only (including the underscore : _). The number of available variables is limited at compile time (and reported by the command CONFIG, and variable $VAR_MAX (see below)), but a larger table can easily be built by recompilation if needed.

Associative arrays can be built using the construction [index] :

```
Crowd> set d[$k] = (cos($k/10))
```

Associative arrays means that each entry is created when needed, entries do not have to be sequential, and the index does not even have to be integer :

```
$d[1] ; $d[3] ; $d[$z] ; $d['foo bar']
```

is a valid array. Associative arrays are coming from the standard UNIX program awk, search for a manual of this program if you some help with associative arrays. The function nextlm() permits to go through all the entries of a given array (see below).

Variables are allocated the first time they are set; referencing a variable not yet allocated generate an error. Variable are automatically deallocated when exiting the macro; they cannot be accessed by any other macros that would be executed during the life of the variable. In other words, variables have local scope, and are volatile (dynamically allocated and removed).

Variables created at interactive level (not within a macro) have a special treatment : their life will be as long as the program (unless an UNSET command is explicitly used), and they can be accessed by any other macro : they are static and have global scope.

If you which to create such a static variable from a macro (useful to remember parameters from one call to the other), you can use the following syntax :

```
set d := 'this variable is static'
```

If `d` was already declared as volatile previously in the macro, the preceding command has the effect of creating a new variable, called d, but in the static storage, independent of the volatile d.

SET, UNSET, MUNSET, DUMP

The commands associated to variable handling are :

```
SET varname = value ; set (and create if
needed) the variable
SET var2 = value    ; create and set a global
variable
UNSET varname       ; remove the variable from
the table
MUNSET list of vars finishing with a *
                    ; removes all the
variables in the list.
```

```
DUMP            ; dump the content of all
currently defined
                        ; variables
```

The format of the DUMP is : name of the variable ; context of the variable : (20 is static, 21 to 29 are macro call levels) ; content of the variable.

Using DUMP you will find that there might be entries in the variable table that do not correspond to user variables. This is due to the fact that the parser uses this table for particular use (dbm arrays, positions of label, location of WHILE, FOR controls). This entries have different formats from the user variables, thus cannot be mistaken with them. Some of these entries holds binary data that might disturb the terminal when using the DUMP command.

Internal variables

There are also special variables, not defined in the variable table, which takes the value of the internal parameters (contexts) of the Crowd program. These variables can be used, but should never be SET, they should be changed by using the associated commands. The list is currently :

```
$_   value of the next parameter present on
the calling line. CANNOT BE USED WITHIN
EVALUATED EXPRESSIONS.
```

if the following command is used :

```
@test 3 test.001
```

within the file test, $_ will be 3 the first time and test.001 the second time.

If no value is present on the calling line, the user will be prompted for the value

```
$ARG True (1) if arguments are present on the
calling line (within macro only)
$CONFIG_OS     The system type, as returned by
the CONFIG command
$CONFIG_WIN    The window manager, as returned
by the CONFIG command
$DNOE the mean error between theoretical and
experimental intensities.
$LEVEL the intensity threshold.
$LICENCE  The licence as returned by the
CONFIG command
$MOTION_MODEL the value of the motion model
identificator.
$NB_RES the number of residus
$NB_SPIN the number of spins
$NB_TMIX the number of mixing times
$NBTYP_RES the number of residu types.
$OMEGA the hydrogen resonance frequency.
$RANDOM   a random variable in the range 0..1
with equiprobable distribution
$RANDOMG  a random variable with normal law,
unit variance and zero mean
$RANDOMZ  same as $RANDOM but resets the
random series
$RFACTD the distance R-factor value between
theoretical and experimental intensities.
$RFACTI the intensity R-factor value between
theoretical and experimental intensities.
```

```
$SEQ[$i] the ith residu name in the molecule
sequence.
$SZAT maximum number of places in the atom
chained list.
$SZRES maximum number of residus in molecular
sequence.
$SZTM maximum number of mixing times.
$SZTYP maximum number of places in the atom
type chained list.
$TAUC the global correlation time.
$THETA the spin-lock nutation angle.
$TMIX[$i] the ith mixing time value.
$VAR_MAX  The total number of user variable
available, as returned by the CONFIG command
$VERSION  The current version, as returned by
the CONFIG command
```

SUPPORT FOR DATA-BASES

There is in Crowd a support for the UNIX standard data-base file format dbm. A dbm data-base is composed of two file : base_name.pag and base_name.dir which together hold the information of the base_name data-base. Each entry consist of a field indexed with an access key. Each entry is accessed with the access key. Data can be retrieve, stored, modified in the data-base. Every operations are realised on the file, but the disk accesses are minimised.

In Crowd, such a dbm data-base is accesses through a binding to an associative array. Each entry of the data-base appear as a element of the array. Reading, storing and modifying entries is performed by fetching and assigning array values. The access key of the entry in the dbm file appears as the index of the associative array, and the content appears as the value of the associative array. Due to internal Crowd limitation, only key up to 31 characters and entry up to 256 characters can be handled.

DBOPEN, DBCLOSE

These two commands permits to associate the dbm file with a pseudo internal array variable. The command

```
dbopen array_name file_name
```

will associate a pseudo associative array with the dbm files of base name : *file_name.* The file is created if it does not exist yet. No actual variable *array_name* is created, but each operation on it is performed on the dbm file.

DBCLOSE closes the file and forget about the array *array_name* .

The functions dbm() and nextlm() permits to handle the dbm file in macro processing.

EVALUATED EXPRESSIONS

Expressions can be evaluated in the Crowd program. Expressions should be enclosed in parentheses. Expressions can use most of the entries found in the paragraph SYNTAX, at the exception of the special entries : `%%` ; `<from_file` and the pseudo variable `$_` .

(2*(5-1)) (cos($i)) are examples of expressions. Expressions can be used whenever a Crowd input is needed (command as well as parameters). Expressions must fit on one line (i.e. 256 characters long), continuation mark cannot be used within expressions.

Values can be numeric, string or logical. Strings should be enclosed within '..' or "..". Logicals are stored as 0 for false and 1 for true (or any non-zero value).

Liste of functions and operators

The following operators and functions are implemented :

- the regular 4 operations : + - / * e.g. : (2*3 - 1)
- the modulo function : % (12 % 5)

• the power ^ operator (3^2.1)
• the regular mathematical functions : sqrt(x) cos(x) sin (x) atan(x) log(x) exp(x) abs(x) int(x) max(x,y) min(x,y)
• the special function power2(n) will have the value of the closest power of 2 below or equal to the number n : power2(130) will be 128 (=2^7)
• the concatenation operator : ('string1' // "string2")
• the formatting operator : ; equivalent to // ' ' //
("Value is:" ; $a) is equivalent to ("Value is:" ; // ' ' // $a)
• the alphanumeric operators :
toupper(st) put string in upper case
tolower(st) put string in lower case
sp(i) generates a string of i blanks
len(st) is the length of the string
index(st1,st2) is the position of st2 located in st1, 0 if not present
subst(st,i,j) is the substring from st, starting at i, ending at j
head(st) will be the first word in string st (blank separated)
tail(st) will be the string st but the first word.
headx(st,c) and tailx(st,c) are equivalent to head and tail but the character c is used rather than blank.

• the numeral comparison operators : == != < > <= and >= for comparing numbers : ($x<=0.5)
• the string comparison s= (equal) and s! (different) for comparing strings : ($s s= 'name')
• the logical operators : | (or) and & (and) : (($x<=0.5)&($s s= 'name'))
• the not operation : ! : (!($i==1)) (!$arg)
• the function eof(file_name) will be true if the last input from file file_name (with <file_name) had reached the end of the file, will be false otherwise.
• the function exist(var_name) will be true if var_name is a user variable either local or global
• the function dbm(array_name) is true if array_name is bound to a dbm file with the DBOPEN command
• the next element function : nextlm. If i is an entry in the associative array $array, the construction nextlm(array,i) will have the value of the next available entry. The end of the array is notified with an empty string. The series is also initialised with an empty string. The series is gone through in the internal order which is not related neither to the input order nor the sequential order. For instance the following macro will print all the entries of the array $table :

```
set $index = (nextlm('table',' '))
while ($index s! ' ')
     print ($index // ' : ' // $table[$index])
     set $index = (nextlm('table',$index))
endwhile
```

Check also the macro tunset, which permits to remove all the entries of an array.

When evaluating expressions, all the internal computations are not typed and performed on strings. This permits to mix integer and string freely. Thus the following expressions are perfectly valid :

```
( cos(1) // 'a string' // ($i==1))
( toupper($file // ($i+1)) )
( log ('0.1') )
```

PROGRAMMING IN MACRO

Introduction to the control language

It is possible in the Crowd program to write command files (macros). The macro is then invoked by the command :

```
Crowd> @file_name
```

or simply

```
Crowd> file_name
```

In UNIX the macro name is the file name, sometime names with a .g extension are chosen. Any characters (but the blank) can be used in macro name, so ../test/my_macro or /usr/local/gifa/macro/startup.g is a valid name.

A macro consists of a series of regular Crowd commands used as in the interactive mode. Up to 9 recursive calls may be nested (i.e. you can call a macro from within a macro, down to 9 nested levels). The macro execution can always be stopped with the ^C command.

When calling a macro, the file is first searched in the current working directory. If the file is not found, it is then searched in the macro sub-directory of the home directory of the user (`$HOME/macro` in UNIX). If it is still not found, it is searched in the standard directory : `/usr/local/crowd/macro` in UNIX.

This is reminiscent with the notion of *path* in UNIX. This permits to build a library of macros, available for every Crowd user, as well as a set of commands personal to each user and to each project.

When the Crowd program is first entered, it tries to execute the macro file called : startup.g, the file is searched in the directory as any regular macro. Thus you can easily set-up you own environment. For instance your startup.g file (in your home directory, or in your working directory) may look like :

```
...any personal set-up
/usr/local/crowd/macro/startup.g ; to be sure to
have general set-up
```

You will find a comprehensive library of such macros in the /usr/local/crowd/macro sub-directory of the distribution kit.

The following commands are available for efficient command programming:

FOR, WHILE, IF, ERROR, GOTO

There are several control structures in the Crowd language. They are all pretty classical, and should not make you any problems.

```
FOR var_name = initial TO final { STEP step}
     .. some Crowd code
ENDFOR
```

This construction permits to write loops. The variable called *var_name*, will be created local to the macro if it does not yet exist, set to the value *initial*, and incremented by 1 for each loop. The increment can be different from 1 (but always numeric) with the optional field STEP *step*. *step* can even be negative, in which case the variable will be decremented. In any case, the test performed to determine whether the loop is finished or not is (*var_name* > *final*) if step is positive or (*var_name* < *final*) if step is negative.

```
WHILE some_value_to_be_evaluated
     .. some Crowd code
ENDWHILE
```

The Crowd code will executed as long as *some_value_to_be_evaluated* is true (is non zero). The value to be evaluated can be a variable or a complex evaluated expression into parentheses.

The IF command has two distinctive syntaxes : a one-line IF without possible nesting nor else; and a complete IF .. THEN construct with ELSIF and ELSE possibilities.

The multi-line IF is as follow :

```
IF test THEN
..commands on several lines
{ ELSIF test2 THEN
..commands }   (eventually many exclusive
tests )
{ ELSE      (default case)
..commands }
ENDIF
```

The different commands will executed conditionally on the value of the tests. Any non-zero value is considered as true. If may be nested, with no limit on the number of nesting.

The one-line IF is as follow :

```
IF logical_value   ...remaining of the line
...
```

All the commands on the remaining of the command line are executed only if *logical_value* holds true (is non-zero).
examples :

```
IF ($X =< 0)  GOTO SKIP
print 'Hello' IF ($TAUC == 1)    set x =
($x+1)
IF (!eof(input)) print 'Still reading the
file' \
                 set line = (%+1) \
                 IF ($line<$linemax) goto
continue
```

tests to be used are described in the “evaluated expression” paragraph.

```
GOTO label
```

will redirect the execution of the macro to the symbol =label appearing somewhere else in the file.
Example

```
=loop
     ...any kind of processing...
     goto loop
```

GOTO should not be used to jump into FOR or WHILE loops, nor into IF .. THEN structures, Unpredictable execution will occur.

All these commands may freely nested. However, except for GOTO and the one-line IF, these commands (FOR, ENDFOR, WHILE, ENDWHILE, =label) should appear alone on one line, with no code preceding nor following the command (or the label), but eventually followed by a comment.

INITINPROGRESS, INPROGRESS

These 2 commands permits to build a bar showing the progress of the current command of the form :

```
In Progress : 0%....25%....50%....75%....100%
```

with a dot every 1/20th of the process
INITINPROGRESS tells Crowd how many iterations are to go, INPROGRESS actually does the display. INPROGRESS can be called at any rate, there is no need to do arithmetic to call INPROGRESS.

It is to note that, for a clean result, no terminal output should be performed during a INITINPROGRESS INPROGESS pair

Example :

```
set max = 1000
initinprogress $max
for i = 1 to $max
   ... do something
   inprogress $i
endfor
```

Interactive macros

During macro execution, it is possible to get input from the user in several ways :

• if a command misses a parameter, the user will be prompt for it, even if the macro is called deep in a nested loop. The user will be prompted with the current value. e.g.

```
print 'Enter new size' chsize
```

(actually the message is not really needed, since the command CHSIZE will issue one)
If the macro is called from the graphic interface (with BUTTONBOX), either directly or indirectly called, a dialogue box will be used to prompt the user for the value, the message from the command (here chsize) will be in the dialogue box.

• by the same token, it is possible to assign the input into a variable :

```
set b = 10   print 'Enter new value'  set b =
```

The user here will be prompted with the value 10 as default for b.
The command MESSAGE permits to put the message in the dialogue box if the macro is called from a menu button, and on the terminal if called from the prompt. Thus this is a better construct :

```
set b = 10   message 'Enter new value'  set b
=
```

• The ALERT command permits to build alert boxes, and BUTTONBOX and FORMBOX permit to build more sophisticate graphic interfaces.

Passing parameters to macros

Parameters can be sent to macros when called by following the call with the parameters to pass; within the macro file, these parameters will be obtain with the pseudo variable $_. each use of $_ "consume" one parameter on the call line. If no parameter is available on the calling line, the $_ pseudo variable will prompt the user directly. The variable $arg permits to test the presence of a value obtainable with $_.

For instance, if the file test_arg is as follow :

```
; to test argument passing
print $_
set b = $_   print (2*$b)
set c = $_
```

the following call

```
test_arg hello 3
```

will produce the following output

```
hello
6
```

and will prompt for the value to be applied to c. Thus $_ can be used to both the command line or the user, much in the same way as regular Crowd commands do. It is even possible to have an optional message, depending whether there is a parameter available or not, with the variable $arg :

```
if (!$arg) print 'Enter new value'
set b = $_
```

The MESSAGE command, has similar built-in mechanism, the string is sent to the user only if no parameters are available on the calling line. But it is better to use the MESSAGE command, since the message will then go to a dialogue box if the macro is called from the graphic interface.

```
message 'Enter new value'
set b = $_
```

Examples :

The PRINT command permits to output to the user a string. If you want to show more than a single line you can always use sh :

```
sh 'more text_file'           ; UNIX
```

A more useful example: write a PDB file using the current atom coordinates.

```
if (!$arg) print "PDB Output Filename ?"
set outfile = $_
sh ('rm tmp')
connect tmp
for $i = 1 to $nb_res
  lstcrd $i
endfor
disconnect
sh ('awk -f mkpdb.awk tmp > '//$outfil)
sh ('rm tmp')
quit
```

Here is a small (silly) interactive example :

```
; to compute the square root of a given number
sh 'more intro.txt'           ; some greeting
message'
set i = 1      ;i should be set here for the
following test to work!
while ($i <> 0)
     print 'Enter your number (0 to end)'
     set i = 0                ; pre-set the value
and ask for it
     set i =
     if ($i>0) then
          print ('Square root of';$i;'is';sqrt($i))
     elsif ($i<0) then
          print 'No real square root !'
     endif
endwhile
exit
```

For speed consideration it is always better, when working on a large macro, to make a small file for a very repetitive piece of work and call it. The looping as well as the parsing may get slow on a large file. It is also slightly faster to call a macro with the @file syntax.

GRAPHIC USER INTERFACE

It is possible to modify and adapt the user interface within the Crowd program. The graphic user interface (GUI) is completely built from a simple set of basic commands which permits to build menus, dialogue boxes, etc...

BUTTONBOX, CLOSEBOX

The command BUTTONBOX creates a new entry in the menu bar, and create the menu bar for the first time. Parameters of the command are the name of the menu as it appears in the menu bar, then a list of entries consisting of the name of the button and the Crowd command associated to it. The lists end with a *. The special entry *separator* does not set a button but a separator in the box.

Any legal command can be used to be associated with a given button, a built-in command as well as a macro, and will be executed as if entered form the terminal. So the WHILE, FOR, IF .. THEN, GOTO commands are not available but the IF .. *any_command* syntax is valid. The action of the command in independent from the way it is called, except for the user prompting for values, which is performed with dialogue boxes in the case of a command called from a button.

Example given :

```
BUTTONBOX "my first menu" \
     Hello  "print 'Hello World'" \
     separator \
     Omega omega \
     "Test Omega" "if ($omega == 600) print
'We are at 600 MHz'" \
     *
```

Note :

• How the command should on a single line, but how the continuation sign (\) can be used

• How the quotes are needed only to isolate field with blanks in it, for button names as well as commands (`Hello Omega`)

• How single and double quotes can be mixed to build composite commands (`Hello`)

• How the on-line IF can be used (`Test Omega`)

The command CLOSEBOX closes the menu bar and all the associated menus. It is equivalent to closing the menu bar from the close box.

MESSAGE

This command permits to build macros which have exactly the same behaviour than built-in commands. The macro `example`

```
message "Enter new correlation time" set
newtauc := $_
```

will react as follow :

nothing will appear if called with a parameter :

```
example 3
```

the user will be prompted if called without parameters :

```
example
```

the user will be prompted in a dialogue box if the macro is called without parameters from a menu button.

In any case, this is independent of the way the macro is actually called, either directly or from within another macro.

FORMBOX, DIALOGBOX

These two commands are closely related.

DIALOGBOX permits to build sophisticated dialogue boxes, several fields can be built into the dialogue box, that the user has to fill-up. Each filed is internally associated to a Crowd variable, that the remain of the macro processes. The command appears as follow : DIALOGBOX [series of entry] *

Each entry consists of a name, that appears as such in the dialogue box, and of a type for the field. Types can be : **message, string, int, real, file, enum**.

message type consists of a string that is displayed on the form and permits to put text in it.

The others type of field have two other entries which give the name of the associated variable and its default value. These types correspond to editable fields that the user has to fill-up. **string, int, real** correspond to string, real values or integer values respectively; **file** corresponds to a file-name entry, and presents a small additional button in the dialogue box, which permits to pop-up a standard Motif file selection box.

The last type : **enum** realises a pop-up menu, for an enumerated choice. It needs an additional entry which described the list of choice. Example given :

```
Dialogbox "my first dialogue" \
     "Enter below the file name to work on"
message \
     "Enter file name" file var_file $name \
     separator \
     "Action to apply" enum "rd_pdb,wr_pdb"
var_act % \
     *
$var_act $var_file
```

This macro will build a dialogue box with 2 editable entries : (filename and action) and will apply the action to the selected file if the user clicks on Ok.

Note :

• How the command should on a single line, but how the continuation sign (\) can be used

• The separator special field which builds a thin line in the dialogue box

• How the enumerated list is comma separated

• How the default value can be anything, here a Crowd internal variable ($name), and the default prompted value (%). In this last case, the default prompted value will be the value of the variable itself if the variable exists before the dialogue is built. If this is not the case, it will be the first entry for **enum**, 0 or **real** and **int** and the empty string for **string** and **file.**

• How the returned values are usual variables, and can be used for anything (here even as Crowd commands)

FORMBOX is the static version of DIALOGBOX. It builds a form that will remain after the completion of the command (as BUTTONBOX does) and will survive as long as the user does not explicitly closes it. FORMBOX needs an additional field (the callback) which describe the Crowd command to be executed when the user clicks on the Ok or Apply buttons. Apart from this, the definition of FORMBOX is strictly equivalent to that of DIALOGBOX. If not global, a variable associated to a field in a FORMBOX is local to the FORMBOX, and cannot be accessed in any other macros but the callback line. Example given :

```
Formbox "my first Form" \
     "print ('Hello';$y_name) print ('today it
is';$the_wet) " \
     "Enter informations below" message \
     "Your name : " string y_name "unknown" \
     "The weather today :" \
     enum "Sunny,Rainy,Cold,Stormy,Terrible"
the_wet % \
     *
```

Note :
• How the form variables can be used in the callback, in expression, as parameters for other commands
• Check other points in DIALOGBOX and BUTTONBOX

ERROR HANDLING

When an error is encountered, the program stops the execution and resumes to the user with a specific message. If the error happens during the execution of a string of commands on a single line, the string is aborted. If it happens during a macro, the macro is aborted.

Typical error conditions are :
• unknown command •wrong parameter for a command (e.g. TAUC -1; MOTION_MODEL 4 etc..). • A impossible read or write command is tried (e.g. rd_pdb DATA.001 and DATA.001 does not exist). • mistyping a parameter (e.g. OMEGA 600,0 instead of 600.0), • a ^C was typed by the user, • an ERROR command was issued.

If the error occurs when the command was executed from a menu button, a alert box will be displayed.

BATCH MODE

It is perfectly possible to create files in order to run the program in background.

On UNIX machines, type a file containing the command as you would type them and just start Crowd by "piping" the file as standard input, and the output on another file. The & sign run the process in background :

```
% gifa < name_of_the_COM_file > process.out &
```

Again, DO NOT forget the final EXIT. Also you may experience problems if you use interactive shell commands such as 'vi' or even 'more'.

The input in the batch file is entered to the program as a user would do, so it is different from macro (of course not for macros called during the batch) in at least to points : •control structures are not allowed •error conditions will not stop the execution, but the program will continue, So be careful when deleting important files within batch files!

When running in batch mode it is usually useless (though not forbidden) to activate the display. However if you want to have Crowd running while you are not logged on, you should choose a graphic-less mode. This mode is entered if there is no X_window DISPLAY available. For instance with

```
unsetenv DISPLAY
```

Do not forget to adapt a special startup.g macro (for instance in the current disrectory) in order not to launch the standard graphic environment (which would give an error anyway).
It is perfectly possible to print results while in batch mode.

ACKNOWLEDGEMENT

The Crowd program has been written during the last two years in C.B.S. (Montpellier). This program uses most of the user-interface developed in Gifa, and has been made available because of the help of Marc Andre Delsuc and Jean Luc Pons. I want to gratefully acknoledge them.

I hope that you will have fun, and enjoy working with the Crowd program. In any case don't hesitate to contact me (bug reports and propositions are specially welcomed).

Therese MALLIAVIN.
terez@cbs.univ-montp1.fr