LastWave 2.0 Reference manual

Emmanuel Bacry
CMAP, Ecole polytechnique, 91128 Palaiseau Cedex, France
email : lastwave@cmap.polytechnique.fr
web : http://www.cmap.polytechnique.fr/˜ bacry/LastWave

This is a reference manual of

- **LastWave Kernel 2.0**, Author: E.Bacry
- **disp 2.0, misc 2.0 and terminal 2.0 packages**, Author: E.Bacry
- **Signal package 2.0**, Authors: E.Bacry, N.Decoster and X.Surraud
- **Image package 2.0**, Authors: E.Bacry and J.Fraleu

The reference manual pages of the other packages are not included in this manual, they should be downloaded separately.

Contents

Chapter 1

Package kernel 2.0

Lastwave kernel package
*** Authors and Copyright : E.Bacry*

1.1 Defined types

1.1.1 Type &array

This type is used to store an array. The syntax is array.field

1.1.2 Type &float

Extended type : same as '&num'

1.1.3 Type &int

Extended type which corresponds to '&num' which are integers

1.1.4 Type &list

This type corresponds to a string that can be interpreted as a list.

- `&list [*opt,...]  [:]= (<string> | <listv>)`
  Get/Set characters
  Options are : `*list,*nolimit,*bconst,*bmirror,*bmirror1,*bperiodic`

  - *list : the string is considered as a list and all the extractions are performed using the list representation
  - *nolimit : indexes can be out of range
  - *bconst : border effect with same characters (last character for right handside and first character for left handside)
  - *bperiodic : periodic border effect)
  - *bmirror1 : mirror+periodic border effect (first and last points are repeated)
  - *bmirror : mirror+periodic border effect (first and last points are NOT repeated)

- `&list.length`
  Get the number of characters of the string

- `&list.llength`
  Get the number of elements of the string considered as a list

- `&list.tonum`
  Performs a conversion to a number if possible. Otherwise it returns 'null'

1.1.5 Type `&listv`

This type corresponds to list of values.
- ==,!= : test equality of all the elements
- Operator * : listv*n, repetition of the listv n times
- Operator + : listv+listv, Appending 2 listv
- Operator + : listv+value, (where value is not a listv), adding the value at the end of the listv
- Constructor : syntax {value1 ... valueN}.

- `&listv [*opt,...]  [:]= (<value> | <listv>)`
  Get/Set the element of a listv
  Options are : `*nolimit,*bmirror,*bmirror1,*bperiodic`

  - *nolimit : indexes can be out of range
  - *bperiodic : periodic border effect)
  - *bmirror : mirror+periodic border effect (first and last elements are repeated)

    - *bmirror1 : mirror+periodic border effect (first and last elements are NOT repeated)

- `&listv.length [= <length>]`
  Sets/Gets the length/size of a listv

- `&listv.size [= <length>]`
  Sets/Gets the length/size of a listv

- `&listv.nAlloc`
  Gets the allocation size of a listv

1.1.6 Type `&null`

This is the type of the element 'null'

1.1.7 Type `&num`

This type corresponds to numbers.
- +,-,*,/,^ (and +=,-=,*=,/=,^ =) : regular operators
- //,% : integer division and remainder
- ==,!=,<=,>=,<,> : regular tests
- &&,|| : and,or
- sinh,sin,cosh,cos,tanh,tan,acos,asin,atan : trigonometric operators
- log2,log,ln,sqrt,abs,exp,ceil,floor,round,frac,int : other math functions
- Constants : pi,grand,urand

1.1.8 Type `&proc`

Corresponds to procedures. The object corresponding to a defined procedure anmed 'name' can be obtained using the syntax %name. One can build anonymous procedure usinng the syntax %{arg1 ... argN}‘my script‘

- `&proc.help`
  Gets the help of a proc as a listv

- `&proc.shelp [= <helpString>]`
  Sets/Gets the help of a proc as a string

- `&proc.file`
  Gets the filename of a proc
- `&proc.package`
  Gets the package name of a proc
- `&proc.script`
  Gets the script of a proc
- `&proc.name [= <name>]`
  Gets/Sets the name of a proc

1.1.9 Type `&range`

This type corresponds to ranges.
- Constructors : syntax is first:step:last or first:#size:last. One can use first!:... to except the first point or ...:!last to except the last one. If implicit the first, last, size or step can be ommitted. Moreover @> (resp. @< or @+) refers to the 'implicit' last (resp. first or step) point.
- Most of the operators valid for signals are valid for ranges.

- `&range [*opt,...]`
  Get range values
  Options are : `*nolimit,*b0,*bconst,*bmirror,*bmirror1,*bperiodic,*x,*xlin`
  - *nolimit : indexes can be out of range
  - *b0 : border effect with 0 value
  - *bconst : border effect with constant values (last range value for right handside and first range value for left handside)
  - *bperiodic : periodic border effect)
  - *bmirror1 : mirror+periodic border effect (first and last points are repeated)
  - *bmirror : mirror+periodic border effect (first and last points are NOT repeated)
- `&range.size [= <size>]`
  Sets/Gets the size of a range
- `&range.first [= <first>]`
  Sets/Gets the first value of a range

- `&range.step [= <step>]`
  Sets/Gets the step of a range
- `&range.last [= <last>]`
  Sets/Gets the last value of a range

1.1.10 Type `&script`

This corresponds to a script. It can be obtained from a procedure using the 'script' field of a '&proc' variable. A script can be directly built using the %%‘a script‘ syntax.

1.1.11 Type `&string`

This type corresponds to strings.
- Constructors : You can use the syntax "string" or 'string'.
- Operator + (and +=) : string1+string2, concatenation
- Operator * (and *=) : string*n, repetition of the string n times.

- `&string [*opt,...]  [:]= (<string> | <listv>)`
  Get/Set characters
  Options are : `*list,*nolimit,*bconst,*bmirror,*bmirror1,*bperiodic`
  - *list : the string is considered as a list and all the extractions are performed using the list representation
  - *nolimit : indexes can be out of range
  - *bconst : border effect with same characters (last character for right handside and first character for left handside)
  - *bperiodic : periodic border effect)
  - *bmirror1 : mirror+periodic border effect (first and last points are repeated)
  - *bmirror : mirror+periodic border effect (first and last points are NOT repeated)
- `&string.length`
  Get the number of characters of the string
- `&string.llength`
  Get the number of elements of the string considered as a list

- `&string.tonum`
  Performs a conversion to a number if possible. Otherwise it returns 'null'

1.1.12 Type `&val`

This type does not correspond to a C type structure. It must be used to type an argument of a procedure.

1.1.13 Type `&valobj`

This type does not correspond to a C type structure. It must be used to type an argument of a procedure.
Extended type that indicates that the argument of a procedure is evaluated but must not correspond to a litteral number.

1.1.14 Type `&word`

This type does not correspond to a C type structure. It must be used to type an argument of a procedure.
Extended type that indicates that the argument of a procedure is not evaluated.

1.1.15 Type `&wordlist`

This type does not correspond to a C type structure. It must be used to type an argument of a procedure.
Extended type that indicates that the argument of a procedure is not evaluated and should correspond to a list.

1.2 Basic interpreter commands

- **apply**

  - **apply** `args [<level>=0] <proc> <arg1> ... <argN>`

    Apply the procedure <proc> with the specified arguments. If <level> is specified (and not 0) then the commands is executed in a different level (not in the current level.

- **apply** `listv [<level>=0] <proc> <listv>`

  Apply the procedure <proc> with the argument list being the successive values of the <listv> . If <level> is specified (and not 0) then the procedure is executed in a different level (not in the current level.

• **array**

- **array** `list <array> [<regexp>='*']`

  Gets the listv of indexes of <array> that matches <regexp>.

- **array** `new [<hashSize>=8]`

  Creates a new &array variable using a hash table of size <hashSize>.

• **break**
Gets immediately out of any 'do', 'while', 'for' or 'foreach' loop.

• **clear** `<val>`
Clears the content of a value.

• **continue**
Skips directly to the next loop of a 'for', 'foreach', 'while' or 'do' loop.

• **copy** `<val> [<val1>]`
Copy of the value <val> in <val1> if specified otherwise returns a copy.

• **delete** `[<level>=0] *varName*`
Deletes a variable.

• **do** `<bodyScript> <continueTest>`
The standard 'do' control loop. At the end of each loop, <continueTest> is evaluated using '=', if it returns 0 then the loop stops otherwise <bodyscript> is executed and the loop goes on (thus <bodysScript> is always executed at least once).

• **echo** `*arg1* ... *argN*`
Just echoes each of its arguments (no evaluation) on standard output.

• **errorf** `<format> [<val1> ... <valN>]`
Same as 'printf' but prints on stderr instead of stdout and generates an error right after printing.

• **eval** `[<level>=0] <script>`
It evals the script in the current level. If <level> is specified (and not 0) then the script is evaluated in a different level (not in the current level).

• **file**

- **file** `cd <directory>`

  Changes the working directory (Warning : only on Unix computers).

- **file** `close <stream>`

  Closes a stream.

- **file** `eof [<stream>=stdin]`

  Returns 1 if <stream> has reached an end of file.

- **file** `exist <filename> [<mode>='r']`

  Tests whether a file named <filename> could be opened using mode <mode>.

- **file** `info <file>`

  Returns a 2 element listv. The first element is the type of the file (either 'directory' or 'file') and the second is its size. If the file does not exist it returns null. (WARNING : the size is wrong on Macintoshes.)

- **file** `list <regexp>`

  Returns the listv of filenames whose filename matches <regexp>.

- **file** `listp <regexp>`

  Same as 'list' but returns complete paths.

- **file** `move <filenameSrc> <filenameTarget>`

  Changes the name of a file.

- **file** `open <filename> ('r' | 'w' | 'a')`

  Opens a file in a 'r'ead, 'w'rite or 'a'ppend mode and returns a stream number associated to that file.

- **file** `openstr <str>`

  Opens a stream associated to a string. It returns the stream number.

- **file** `remove <filename>`

  Removes a file.

- **file** `set (stdin | stdout | stderr) [<stream>]`

  Gets or Redirects 'stdin', 'stdout' or 'stderr' streams.

- **file** `tmp`

  Gets a filename that does not exist and that can be used for a temporary file.

• **for** <startScript> <continueValue> <nextScript> <bodyScript>
The 'for' control loop. The <startScript> is executed just once at the beginning of the loop. The loop is continued while <continueValue> is evaluated to a number different from 0. The <nextScript> is executed at the end of each loop (just before <continueTest> is evaluated) and <bodyScript> is the core of the loop.

• **foreach** <var> (<list> | <listv> | <signal> | <range>) <bodyScript>
The variable <var> loops successively on each element of the <list>, <listv> <signal> or <range> till its end is reached. After each assignation <bodyScript> is executed.

• **getchar** `[*var*]`
Reads from stdin a character and returns it or sets it in variable *var* if specified.

• **getline** `[*var*]`
Reads from stdin a line (ended by a newline or an eof) and returns it or sets it in variable *var* if specified.

• **global** `[*&type*] '*varName1* [*newVarName1*]' ... '*varNameN* [*newVarNameN*]'`
Same as the 'import args' command except that it imports from the global level.

• **h**
Short name for the command 'history'.

• **history**

  - **history** `index [<index>]`

    If <index> is specified, it returns the history command associated to the <index> number in the history. If not, it returns the current index number (of the next entry in the history).

  - **history** `match [<regexp>='*']`

    Gets all the history lines that matches <regexp>. The result is given as a listv which goes from the most recent command to the eldest one.

- **history** `size [<size>]`

  Gets/Sets the size of the history.

- **history**

  Displays the history of the commands (i.e., the last commands that were typed in the terminal along with their index number).

• **if** `<test> <thenScript> [elseif <test1> <thenScript1> ..... elseif <testN> <thenScriptN> else <elseScript>]`
The standard 'if then else' control. WARNING: each memeber of the <test> is evaluated (even if || or && clause).

• **import**

- **import** `args [<level>=-1] [*&type*] ‘*varName1* [*newVarName1*]‘ ... ‘*varNameN* [*newVarNameN*]‘`

  Imports N variables of type *&type* from level <level> whose names are *varName1*...*varNameN* and eventually renames them in the local environement with names *newVarName1*...*newVarNameN*. If these new names are not specified then in the case of non array variables, the same names are used. When the array syntax is involved to refer to any variable *var1* to *varN* (e.g., 'people.john.adress') and if no new name is specified, then the last index name is used (in our case the variable will be named 'adress' in the local environment). If the type is &var or &array and if any variable does not exist at level <level> it will create it in both the level <level> and the local environment. If it fails then an error occurs unless [?] is specified right after the <level> in which case it returns 0 instead of 1.

- **import** `list [<level>=-1] {{[*&type1*] *varName1* [<varDef1>]} ... {[*&typeN*] *varNameN* [<varDefN>]}} <list>`

  This action does exactly what happens when a script procedure is called : the <list> of arguments (evaluated in the calling level) are matched to the variable pattern list.

- **import** `listv {{[*&type1*] *varName1* [<varDef1>]} ... {<*&typeN*> *varNameN* [<varDefN>]}} <listv>`

  Same as the 'list' action except that a list of values is passed instead of a string list (which is evaluated).

• **info** <value>
Prints info on a value.

• **list** Old LastWave 1.7 command
Not to be used.

• **listv**

  • **listv** map <listv> <proc>

    Maps the procedure <proc> on each element of <listv> and returns the listv of the results. The procedure <proc> should take one argument and return a value that will be inserted in the result listv. If it does not return anything, nothing is insterted in the listv

  • **listv** niceprint <listv> [<colSize>]

    Prints each element of the listv using tabulated columns.

  • **listv** sort <listv> [<proc>] [-i]

    Returns the sorted listv. the flag '-i' just reverse the resulting listv. If no <proc> are given, the <listv> must be composed either only of numbers (the sorting is then performed in the increasing order) or only by strings (the sorting is alphabetical). An error occurs in any other case. If <proc> is specified, it is supposed to be the comparaison procedure. It must take 2 arguments arg1 and arg2 and must return an integer (negative if arg1<arg2, 0 if arg1==arg2 and positive otherwise

• **new** *&type*
Returns a new variable of type <type>.

• **package**

  • **package** dir [<directory>]

    Sets/Gets the directory where all the package script files are to be searched. Whenever a package named 'name' is loaded, the source directory '<directory>/name' will be added and the script file <directory>/name.pkg is sourced if it exists.

  • **package** list [<regexp>='*']

    Gets the listv of all the available packages. Each element of the listv is a listv with the name of the package, a flag indicating whether the corresponding package is loaded (flag==1) or not (flag==0), the year the package has been written, the version number, a list of authors and a one-line help.

- **package** `load [<regexp>='*']`

  Load the packages whose name matches <regexp>.

- **package** `name [<regexp>='*']`

  Gets the list of all the package whose name matches <regexp>.

- **package** `new <name> <year> <version> <authors> <info>`

  Creates a new package named <name> that is copyrighted by <authors> and which was written in year <year> and whose version number is <version>. <info> is a text decribing the package.

• **print** `<value1> [-s | <value2> ... <valueN>]`
Prints one or seveal values. Option '-s', prints it the 'short' way.

• **printf** `<format> [<val1> ... <valN>]`
Same syntax as 'printf' in C-Language : prints on stdout the values <val1>...<valN> according to <format>. Warning : Most of the '%' C-format can be used (but not to complex ones !). Moreover two other formats '%V' (resp. '%v') can be used to print the long (resp. short) string representation of a value.

• **proc**

- **proc** `Old Lastwave command`

  Not to be used.

- **proc** `clist [<nameRegexp>='[^ _]*'] [<tableNameRegexp>='*'] [<packageNameRegexp>='*'] [-m]`

  Same as 'list' action but looks for C-commands only.

- **proc** `ctable [<tableNameRegexp>='*'] [<packageNameRegexp>='*']`

  Gets the listv of all the C-procedure tables that belongs to a package whose name matches <packageNameRegexp>.

- **proc** `get <proc_name>`

  returns the script procedure or the C-procedure whose name is <proc_name>. The type of the result is '&proc'.

- **proc** `list [<nameRegexp>='[^ _]*'] [<tableNameRegexp>='*'] [<packageNameRegexp>='*'] [-m]`

  Gets the listv of all the procedure names which match <nameRegexp> and (in the case of C-commands) which belongs to a procedure table whose name matches <tableNameRegexp>. If '-m' is on then

it looks for a command named <nameRegexp> (exact string). The listv is organized as 2 sublistv, one for script commands and one for C-commands

- **proc** `slist [<nameRegexp>='[^ _]*'] [<tableNameRegexp>='*'] [<packageNameRegexp>='*'] [-m]`

  Same as 'list' action but looks for script commands only.

- **proc** `undef <name1> [<name2> ... <nameN>]`

  Removes N script procedures from the procedure table. If any variable still points to the procedure, the procedure becomes 'anonymous', consequently its name name is changed to an anonymous name.

- **proc** `var <sc_name>`

  Gets the variable list of a script command.

• **randinit** `[<init>]`
Initializes the random generator. If <init> is not specified it initializes it using the computer time. If not, it must be a positive integer.

• **return** `[<value>] [-e]`
Exits right away from a script procedure and returns <value> if specified otherwise it returns the null pointer. If option '-e' is on, <value> is not evaluated.

• **scanf** `<format> [*var1* ... *varN*]`
Same syntax as 'scanf' in C-Language : reads arguments from stdin according to <format> and puts them in the variables *var1*...*varN*. Warning : Most of the '%' C-format can be used (but not too complex ones !).

• **set** `Old Lastwave 1.7 command`
Not to be used.

• **setproc** `(*name* | -) {{[*&varType1*] *varName1* [<varDef1>]} ... {[*&varTypeN*] *varNameN* [<varDefN>]}} ["{{{*usageAction1*} {*helpAction1*}} ... {{*usageActionN*} {*helpActionN*}}}"] <bodyScript>`
Defines a new script procedure whose name is *name* (a non evaluated string) and returns it. If the name is '-' then the created procedure is anonymous (you can create anonymous procedure using the syntax '%{a b}‘script‘' too). The variables of the procedure are defined as a list where the type of each variable *&varTypeN* can be specified along with a default value <varDefN> if the argument is optional. When no type is specified the '&val' type is used (i.e., evaluation will take place with no apriori on the

type of the result). The last variable can be of the dotted form (e.g., .l) in which case all the remaining arguments will be affected to this argument. There are 2 cases : if the type '&wordlist' is specified for the dotted variable, the remaining arguments are not evaluated and the dotted variable will be a string list made of the argument names. If no type is specified each remaining argument is evaluated and the so-obtained values are stored in the dotted variable as a '&listv'. One can specify usage form and a one line help. A simple form is for instance "{{{<arg1> <arg2>} {The one line help}}}". If the procedure admits several 'actions' (as the 'var' C-procedure for instance), a simple form is for instance "{{{action1<arg1> <arg2>} {The one line help for action1}} {{action2 <arg1> <arg2>} {The one line help for action2}}}".

• **setsourcedirs** `[<listv of directories>]`
Sets/Gets the list of directories the 'source' command looks a file in.

• **setv** `[*var* | *list_of_var*] [[:*+/-^ ]=] <value> [-l <level>]`
This is the main evaluator command. The command 'a+=b' is equivalent to 'setv a += b' and 'a=b' is equivalent to 'setv a = b' or also 'setv a b'. The only advantage of the 'setv' syntax is that it lets you specify a <level> in which the evaluation takes place (not the assignement !!).

• **setvar** `Old Lastwave 1.7 command`
Not to be used.

• **shell** `*unixCommand*`
Executes an unix shell command (for Unix computers only). You can also use the syntax '! *unixCommand*'. Warning : Do not use any interactive unix command (such as 'more'), it will core dump lastwave !

• **source** `*filename1*...*filenameN*`
Source each file *filename1*...*filenameN*.

• **sprintf** `*var* <format> [<val1> ... <valN>]`
Same syntax as 'sprintf' in C-Language : sets variable *var* according to <format> with string variables <val1>...<valN>. Warning : Most of the '%' C-format can be used (but not to complex ones !). Moreover two other formats '%V' (resp. '%v') can be used to print the long (resp. short) string representation of a value. It returns the formatted string.

• **sscanf** `<string> <format> [*var1* ... *varN*]`
Same syntax as 'sscanf' in C-Language : reads arguments from <string> according to <format> and puts them in the variables *var1*...*varN*.

Warning : Most of the '%' C-format can be used (but not to complex ones !).

- **str**
  - **str** `2ascii <str>`

    Gets a listv made of the ascii codes corresponding to the characters of the string <str>.
  - **str** `ascii2 <listv_of_codes>`

    Gets the string associated to a listv of ascii codes (this action is the reverse action of the 'ascii' action).
  - **str** `inter <str> <str1>`

    Get the common substring which starts both strings
  - **str** `match <str> <regexp> [<nOcc>=-1]`

    Tests whether some substrings of <str> match the <regexp>. It returns a listv of ranges which correspond to non overlapping maximal substrings which match the <regexp>. <nOcc> is the maximum number of occurence it should return (if negative then all are returned). The wild cards characters for regexp are
    ^ : beginning of string
    $: end of string
    ?: a single character
    +: one character or more
    : zero character or more
    [+...]: one character (or more) that follows
    [+^ ...]: one character(or more) that is different from the characters that follow
    [#...]: zero or one character that follows
    [#^ ...]: zero or one that is different from the characters that follow
    [*...]: zero character (or more) that follows
    [*^ ...]: zero character(or more) that is different from the characters that follow
    |...|: range delimiters
  - **str** `substr <str> <substr> [<nOcc>=-1]`

    Same as action 'match' but optimized in the case the <regexp> is a simple string with no wild card.

- **terminal**
  - **terminal** `beep`

    Just makes the terminal beep.
  - **terminal** `cursor <position>`

    Sets the cursor to an absolute position on the current terminal line.
  - **terminal** `eof`

    Sends an 'eof' character to the terminal.
  - **terminal** `erasechars [<nChars>=1]`

    Erases <nChars> characters from cursor position on current terminal line.
  - **terminal** `eraseline`

    Erases current terminal line.
  - **terminal** `insert <string>`

    Inserts a string on current terminal line from cursor position.
  - **terminal** `line [<str>]`

    Gets/Sets the current terminal line.
  - **terminal** `mode`

    Gets the current mode of the terminal. It is either : 'getchar' (while the terminal waits for a single character), 'scanline' (while the terminal waits for a whole line to be typed in but not for a command line), 'command' (when the terminal waits for a command line to be typed in, in front of the prompt).
  - **terminal** `movecursor <nSpaces>`

    Moves cursor <nSpaces> forward (>0) or backward (<0) from cursor position.
  - **terminal** `movewindow <x> <y>`

    Moves terminal window to position <x> <y> (Warning : only on Macintosh computers).

- **terminal** `prompt [<promptProc>]`

  If no argument it gets the current prompt otherwise it sets the procedure that is supposed to return the prompt string.

- **terminal** `resizewindow <w> <h>`

  Resizes terminal window using <w> <h> (Warning : only on Macintosh computers).

• **time**
Gets the number of seconds since lastwave started.

• **type**

- **type** `exist <&type> [-s]`

  Returns 0 if type <&type> does not exist and 1 otherwise. If '-s' is on, it tests wether this type is associated to a type structure (which is not the case, for instance, of the type '&val').

- **type** `field <&type> [<regexp>='*'] [-sge]`

  Returns the listv of the field names associated to the type <&type>. The names must match <regexp>. If '-s' is on, only write-enabled fields are returned. If '-g' is on, only read-enabled fields are returned. If '-e' is on, only extract-enabled fields are returned.

- **type** `help <&type> [<field> | -n]`

  If no argument, it returns the documentation about the type <&type>. If argument <field>, it returns the listv of documentation on the field <field> of the type <&type>. The first element is the Get documentation, the second is the Set documentation, and the third one is the extract option documentation. If the corresponding method does not exist 'null' is put in the listv. In the case <field> is replaced by '-n', this command returns the documentation concerning number extraction (e.g., 10a) if number extraction is permitted or null if not.

- **type** `list [<packageName>]`

  Returns the listv of all the names of the available types. If <packageName> is specified, only types of the package are returned.

• **val**

- **val** `eval <val> [-l <level>]`

  Evals the value <val> and returns it.

- **val** `test *exprString* [<&type>] [-l <level>] [-E]`

  Returns the listv '{<type> <result>}' if the *exprString* evaluates successfully to <result> (which must be of type <&type> if argument <&type> is specified). If evaluation is not successfull it returns 'null' unless '-E' is set in which case it generates an error. Option '-l' lets you specify a level in which evaluation takes place.

- **val** `type <val> [-b] [-l <level>]`

  Returns the type of <val>. If '-b' then the 'basic' type is returned, e.g., &signal is returned instead of &signali. Option '-l' lets you specify a level in which evaluation takes place.

- **var**

  - **var** `delete [<level>=0] <var1> [<var2>...<varN>]`

    Deletes N variables from level <level>.

  - **var** `env [<level>=0] [<regexp>=[^ _]*] [<&type>]`

    Gets a list made of 3 or 2 element lists corresponding to each variable of level <level> and type <&type> whose name matches <regexp>. Each of this list is made of the name of the variable, its type and in the case it is a number or a string its value.

  - **var** `exist [<level>=0] *[@]varName* [<&type>]`

    Tests whether a variable (it can ONLY be a simple variable like 'me' or a variable stored in an array like 'me.house.kitchen') of type <&type> exists in level <level>. It returns 1 if it does not exist and 0 otherwise. It works with @variables too (in that case <level> is not used).

  - **var** `list [<level>=0] [<regexp>=[^ _]*] [<&type>]`

    Gets the list (not a listv -> this will be changed soon) of variable names of level <level> and type <&type> whose name matches <regexp>.

  - **var** `type [<level>=0] <var> [<&type>]`

    Gets the type of a variable from level <level>. It can ONLY be a simple variable like 'me' or a variable stored in an array like 'me.house.kitchen'. If it does not exist then it returns the empty string. Otherwise it returns its type.

  - **var** `unix <unixVariable>`

    Gets the string value of a unix shell variable (on unix computers only).

• **while** <continueTest> <bodyScript>
The standard 'while' control loop. At the begining of each loop, <continueTest> is evaluated using '=', if it returns 0 then the loop stops otherwise <bodyscript> is executed and the loop goes on.

1.3 Basic graphic commands

• **binding**

- **binding** `activate <groupNameRegexp> [*class*]`

  Activates all the bindings that correspond to all binding groups whose name matches <groupNameRegexp>.

- **binding** `deactivate <groupNameRegexp> [*class*]`

  Deactivates all the bindings that correspond to all binding groups whose name matches <groupNameRegexp>.

- **binding** `delete <groupNameRegexp> [*class*]`

  Deletes all the bindings that correspond to all binding groups whose name matches <groupNameRegexp>.

- **binding** `info <groupNameRegexp> [*class*]`

  Get info on the bindings that correspond to a binding group which matches <groupNameRegexp>.

• **color**

- **color** `animate <color> (rgb <r> <g> <b> | hsv <h> <s> <v>)`

  Changes interactively the definition of the color <color> using RGB or HSV convention (Warning : It only works on Unix computers).

- **color** `ilist [<colormap>=<currentColorMap>] [<index>]`

  Gets a rgb definition listv '{r g b}' of the color <index> in <colormap> or (if <index> is not specified) a listv of all the colors {r g b}.

- **color** `inew [<colorMap>=<currentColorMap>] <index> (rgb <r> <g> <b> | hsv <h> <s> <v>)`

  Creates a new indexed color of a colormap using either RGB or HSV convention.

- **color** `install`

  Install all the colors that were defined with the 'color' command.

- **color** `nb`

  Returns the total number of available colors for lastWave.

- **color** `nlist [<nameRegexp>='*']`

  Gets a listv of the rgb definition '{name r g b}' of colors whose name matches <nameRegexp>.

- **color** `nnew <name> (rgb <r> <g> <b> | hsv <h> <s> <v>)`

  Creates a new named color using either RGB or HSV convention.

- **colormap**

  - **colormap** `current [<colorMap>]`

    Gets/Sets the current color map.

  - **colormap** `delete [<colorMap>=<currentColorMap>]`

    Deletes the <colorMap>.

  - **colormap** `list [<nameRegexp>='*']`

    Gets a list of all the matching color map names.

  - **colormap** `new [<colorMap>=<currentColorMap>] <nbOfColors>`

    Creates/Modifies the <colorMap> so that it can store <nbOfColors> colors.

  - **colormap** `size [<colorMap>=<currentColorMap>]`

    Gets the size of <colorMap>.

- **draw**

  - **draw** `axis *gobject* <x> <y> <w> <h> <xMin> <xMax> <yMin> <yMax> [-title <title>] [-xlabel <label>] [-ylabel <label>] [-margin <nbOfPixels>] [-font] [-clip] [-pen <penSize>] [-color <color>] [-line ('dash' | 'solid')] [-mode ('normal' | 'inverse')] [-reverse (x | y | xy | none)]`

    Draws x and y-axis around the rectangle <x>, <y>, <w>, <h>. These axes correspond to abscissa ranging from <xMin> to <xMax>

and ordinate ranging from <yMin> to <yMax>
-reverse : The state that indicates which axis are reversed compared to the regular window coordinate system (y-axis going from top to bottom and x-axis going from left to right). Let us note that the same field exists for Views, so that they can be used along with axis. If it is equal to 'y' (the default value at initialization) then the y-axis will be going from bottom of the window to the top, if it is equal to 'x' the x-axis will be going from right to left, if it is equal to 'xy' both will be combined and if it is equal to 'none' the y-axis will be top to bottom and the x-axis left to right (as for windows and regular glists).
-frame : If not set then only two axis are drawn. If set then the whole rectangle <x>,<y>, <w>, <h> is surrounded by axes (i.e., 4 axes are drawn).
-ticksIn : The ticks on the axis are inside the rectangle instead of outside.
-margin : A margin of size <nbOfPixels> is used around the rectangle <x>, <y>, <w>, <h> to draw each axis.
-clip : The clip rectangle is set to be the *gobject* rectangle.
-color : The color <color> is used for drawing the axis.
-pen : The size of the pen <penSize> is used for drawing the axis.

- **draw** `cross *gobject* <x> <y> <r> [-clip] [-pen <penSize>] [-color <color>] [-line ('dash' | 'solid')] [-mode ('normal' | 'inverse')]`

  Draws in the *gobject* a cross centered at <x> <y> and with a pixel radius of <r>
  -clip : The clip rectangle is set to be the *gobject* rectangle.
  -color : The color <color> is used for drawing the cross.
  -line : If 'dash' then the cross will be dashed.
  -mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the cross will disappear.

- **draw** `ellipse *gobject* <x> <y> <w> <h> [-clip] [-pen <penSize>] [-color <color>] [-line ('dash' | 'solid')] [-mode ('normal' | 'inverse')] [-pixel] [-rectType <rectType>] [-centered] [-fill]`

  Draws in the *gobject* an ellipse in the rectangle <x> <y> <w> <h>.

-centered : The ellipse is centered at <x>,<y> and the respective radii are <w> and <h>.
-clip : The clip rectangle is set to be the *gobject* rectangle.
-color : The color <color> is used for drawing (and filling) the ellipse.
-fill : The ellipse is filled with the current color.
-rectType : Changes the rectangle type that is used. Read the manual pages about the 'rectType' field of the gobjects.
-line : If 'dash' then the contour of the ellipse will be drawn using dashed lines
-mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the ellipse will disappear.
-pen : The size of the pen <penSize> is used for drawing the ellipse.

- **draw** `gobject *glist* *gclass* [-clip] [list of -<fieldN> [<val1>...<va`

  Draws (without creating it) a gobject of class *gclass* in the <glist> and whose fields are specified by the -<fieldN> [<val1>...<valP>] list. If '-clip' then the current clip rectangle is not used and the entire object is drawn otherwise the current clip rectangle is used.

- **draw** `line *gobject* <x1> <y1> <x2> <y2> [-clip] [-pen <penSize>] [-color <color>] [-line ('dash' | 'solid')] [-mode ('normal' | 'inverse')]`

  Draws in the *gobject* a line from the point <x1>,<y1> to <x2>,<y2>
  -clip : The clip rectangle is set to be the *gobject* rectangle.
  -color : The color <color> is used for drawing the line.
  -line : If 'dash' then the line will be dashed
  -mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the line will disappear.

- **draw** `lineto *gobject* <x> <y> [-clip] [-pen <penSize>] [-color <color>] [-line ('dash' | 'solid')] [-mode ('normal' | 'inverse')]`

  Draws in the *gobject* a line from the last point that has been drawn to <x> <y>
  -clip : The clip rectangle is set to be the *gobject* rectangle.
  -color : The color <color> is used for drawing the line.
  -line : If 'dash' then the line will be dashed

-mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the line will disappear.

- **draw** `point *gobject* <x> <y> [-clip] [-color <color>] [-pen <pensize>] [-mode ('normal' | 'inverse')]`

  Draws in the *gobject* the point <x> <y>
  -clip : The clip rectangle is set to be the *gobject* rectangle.
  -color : The color <color> is used for drawing the point.
  -mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the point will disappear.

- **draw** `rect *gobject* <x> <y> <w> <h> [-clip] [-pen <penSize>] [-color <color>] [-line ('dash' | 'solid')] [-mode ('normal' | 'inverse')] [-pixel] [-rectType <rectType>] [-centered] [-fill]`

  Draws in the *gobject* the rectangle <x> <y> <w> <h>.
  -centered : The rectangle is centered at <x>,<y> and the respective radii are <w> and <h>.
  -clip : The clip rectangle is set to be the *gobject* rectangle.
  -color : The color <color> is used for drawing (and filling) the rectangle.
  -fill : The rectangle is filled with the current color.
  -rectType : Changes the rectangle type that is used. Read the manual pages about the 'rectType' field of the gobjects.
  -line : If 'dash' then the contour of the rectangle will be drawn using dashed lines
  -mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the rectangle will disappear.
  -pen : The size of the pen <penSize> is used for drawing the rectangle.

- **draw** `string *gobject* <str> [*hPositionMode*=left] <x> [*vPositionMode*=base] <y> [-clip] [-color <color>] [-mode ('normal' | 'inverse')] [-font <fontName>]`

  Draws the (eventually multiple line) string <str> at the position <x> <y> using the gobject font. The two parameters <hPositionMode>

and <vPositionMode> fix how these coordinates are used.
if <hPositionMode>=='left' then the string is justified to the left and <x> corresponds to its left position.
if <hPositionMode>=='rightN' then each individual line of the string is justified to the right and <x> corresponds to their right positions.
if <hPositionMode>=='right1' then all the lines (as a group) of the string are justified to the right and <x> corresponds to its right position.
if <hPositionMode>=='middleN' then each individual line of the string is justified to its middle point and <x> corresponds to their middle positions.
if <hPositionMode>=='middle1' then all the lines (as a group) of the string is justified to its middle point and <x> corresponds to its middle position.
if <vPositionMode>=='base' then <y> corresponds to the position of the baseline of the first line.
if <vPositionMode>=='down' then <y> corresponds to the position of the bottom of the last line.
if <vPositionMode>=='up' then <y> corresponds to the position of the top of the first line.
if <vPositionMode>=='middle' then <y> corresponds to the position of the middle of the whole string
if <vPositionMode>=='middleup' in case <str> has just one line, <y> corresponds to the position of the middle of the characters above the baseline (such as '1')
-clip : The clip rectangle is set to be the *gobject* rectangle.
-color : The color <color> is used for drawing the string.
-mode : If 'inverse' then the drawing uses the inverse color mode so that if the same drawing command is executed a second time, the string will disappear.

• **event** `process [<time>=0]`
Waits for <time> seconds while processing to events. If <time>==0 then it processes all the events in the queue and then returns

• **font**

  • **font** `default [<fontName>]`

    Gets/Sets the font that is used by default by all the gobjects.

  • **font** `exist <fontName>`

Returns 1 if the font named <fontName> exists and 0 if not. The name of the font is of the form <name>-<size>-<style> (e.g., Geneva-9-plain) where <style> is either 'plain', 'bold', 'italic', 'boldItalic'.

- **font** `info <font>`

  Gets an information list about the font. The list is made of 3 numbers : <ascent> <descent> and <interline>. <ascent> is the maximum of points (for all characters of the font) above the base line (excluding the base line), <descent> is the maximum of points (for all characters of the font) below the base line (including the base line) and <interline> is such that 2 base lines of 2 successive lines are vertically spaced by the number of points : <ascent>+<descent>+<interline>.

- **font** `list [<nameRegexp>='*'] [<size>='*'] [<styleRegexp>='*']`

  Gets a list (not a listv!!) of the available fonts matching the pattern <name>-<size>-<style> where <name> and <style> are pattern matching expressions and <size> is either a number or '*'. WARNING : on Macintosh computers, the returned list of fonts is a simple list of names since all the styles and sizes are available for true type fonts. Moreover, this list is not complete (it will be in a future version of LastWave). On Unix/X11 computers, the list is a list of regular font names of the type <name>-<size>-<style>. However there can be duplicates.

- **font** `name <font>`

  Gets the basic name (e.g. Geneva, Times...) of the <font>

- **font** `rect <font> <string> <hPositionMode> <x> <vPositionMode> <y> [*gobject*]`

  Gets a listv representing the bounding rectangle of the string <string> if it were drawn using <font> and using the 'draw string' command in the *gobject* (or any window if *gobject* is not specified) along with the arguments <hPositionMode>, <x>, <vPositionMode> and <y>. Read the 'draw string' manual pages to learn about these arguments.

- **font** `size <font>`

  Gets the size of the <font>

- **font** `style <font>`

  Gets the style of the <font>

- **gclass**
  - **gclass** `father <class> [<fatherClass>]`

    Gets the name of the father class of the class <class> if <fatherClass> is not specified. Otherwise it tests whether <class> inherits from <fatherClass> and returns 1 (if it is) or 0 (if not)

  - **gclass** `help <class> (setg | msge)`

    Gets the help listv on the 'setg' or 'msge' class procedure. It is a listv made of several listv (one for each field or each message) with 2 elements : a usage and a one-line help.

  - **gclass** `info <class>`

    Gets a one-line help on a class.

  - **gclass** `list [<nameRegexp>='*'] [<packageNameRegexp>='*'] [<&type>]`

    Gets all the names of gclasses which name matches the <nameRegexp>, who belongs to a package whose name matches <packageNameRegexp> and who can display the content of <&type> variables (or of any type if <&type> is not specified).

  - **gclass** `new *name* *fatherClass* (<setSCommand> | null) (<messageSCommand> | null) (<drawSCommand> | null) <info> [<isinSCommand>] [-t <&type>] [-lm]`

    Creating a new gclass of name *name* based on the class *fatherClass*. The arguments are :
    - <setSCommand> : A procedure that will be used to set/get the fields of the newly defined gobjects. Its argument must be 'obj field .l' where 'obj' is the gobject name, 'field' is the name of the field (excluding '-', e.g., 'pos' or 'size') and 'l' is a listv that groups whatever is left. If it is empty, it means that the procedure must return the value of the field (i.e., this corresponds to a 'get'), otherwise the procedure must set the field using the arguments in 'l'. If the field 'field' is not taken care by the procedure <setSCommand>, the procedure MUST not return any value. Let us note that in the case of a 'set', it must return either '1' if the gobject must be added to the current update list or '-1' if not. If the field name starts with '-?' (e.g., '-?bound' for Views or '-?wtrans' for GraphWtrans), then it means that this field is a 'read-only' field that can use the arguments in 'l'. Thus, the procedure should just return the field value.

- <messageSCommand> : A procedure name that will be used to perform messages sent to the newly defined gobjects. Its argument must be 'obj msge .l' where 'obj' is the gobject name, 'msge' is the name of the msge and 'l' the listv of all the remaining arguments. If the message is accepted, the procedure <messageSCommand> should return something (1 for instance) and it should return nothing if not. Let us note that there are 3 special messages that you can redirect : the 'init' message that is sent right after a new gobject of class *name* is created ('l' is empty and this message is first sent to the gobject as an object of the class 'GObject' and so on to all the classes it inherits from, from top to bottom), the 'delete' message that is sent right before a gobject of class *name* is deleted, ('l' is empty, and it is sent to the gobject as an object of all the gclasses it inherits from, from bottom to top) and the 'deleteNotify' message that is sent to a 'GList' (and all the classes that inherit from GList) whenever one of its gobject is asked to be deleted, the name of the gobject is 'l' and it must return 1 if it accepts that the object is deleted or 0 to forbid deletion.
- <drawSCommand> : A procedure name that will be used to draw the newly defined gobjects. Its argument must be 'obj .l' where 'obj' is the gobject name and 'l' is a listv of 4 floats representing the rectangle {x y w h} that must be redrawn (using local coordinates). The clip rectangle is automatically set to this rectangle before the procedure is called, thus you should not worry redrawing more than what is asked if it is easier to manage.
- <isinSCommand> : An optional procedure name that will be used to check whether the mouse is in one of the newly defined gobjects (if it is not specified then the mouse will be considered to be in the gobject if it is in the rectangle given by '-pos' and '-size'). Its argument must be 'obj x y' where 'obj' is the gobject name and 'x' and 'y' the local coordinates of the mouse. It must return a strictly negative number if the mouse is not in the gobject and otherwise a positive number which quantifies the distance between the mouse and the gobject. Lastwave always sends mouse events to the 'closest' gobject. If it returns 0, then lastwave will not try any other gobject and will consider this gobject to be the closest.
- <info> : This is just a one-line help to describe what the class is meant for
-m : If set then the gobjects of the newly defined class cannot be moved or resized.
-l : If set then the system of local coordinate system of the gobjects

of the newly defined class will be the same one as the father's system (this is the case of GraphSignal for instance).

- **gclass** `package <class>`

  Gets the package name the class is defined in.

- **gclass** `type <class>`

  Gets the eventual variable type associated to the class 'class' (returns it or returns ").

- **gupdate**

  - **gupdate** `add <gobjectlist>`

    Adds a list of gobjects to the current update table so that these gobjects will be redrawn when 'gupdate do' is called.

  - **gupdate** `do`

    Performs the current update. It redraws all the gobjects that needs to be updated since 'gupdate start' was called.

  - **gupdate** `start`

    When called, it creates a new 'update' table and makes it current (the old current update table will become active as soon as this new one is closed). Any gobjects whose fields change and who are not explicitely redrawn are saved in the current 'update' table. The current table is emptied when 'gupdate do' is called. At that time all the gobjects of this table are redrawn as well as all the gobjects in front of them and the ones in the back (if the background color is invisible). These commands must be used inside a script command (all the update tables are deleted when terminal is waiting for a command line to be typed in). Let us note that whenever a 'setgu' command is executed, a 'gupdate start' is executed before changing any field and a 'gupdate do' is executed at the end of the 'setgu' command. Warning : only gobjects belonging to the same window can be put in an update table.

- **msge** `*gobjectRegexp* *msge* <arg1>...<argN>`

Sends a message to all the gobjects whose name matches *gobjectRegexp*. The message is *msge* and <arg1>...<argN> are the arguments to the message. If the message is unknown it returns " otherwise it returns a value different from ".

- **ps**
  - **ps** `cpos [<cmXPos> <cmYPos>]`

    Sets/Gets the x and y distances (in centimeters) of the postscript drawing from the top-left corner of the paper. Warning : This is not quite working yet.

  - **ps** `csize [<cmXSize> [<cmYSize>=-1]]`

    If no argument, it returns a listv made of the x and y sizes of the postscript drawing in centimeters. If both arguments are specified, it sets the x and y sizes (centimeters). And if only the first argument is specified (second is -1), it sets the x-size and the y-size will be automatically computed using the x/y size proportion of the window to draw.

  - **ps** `ipos [<inchXPos> <inchYPos>]`

    Sets/Gets the x and y distances (in inches) of the postscript drawing from the top-left corner of the paper. Warning : This is not quite working yet.

  - **ps** `isize [<inchXSize> [<inchYSize>=-1]]`

    If no argument, it returns a listv made of the x and y sizes of the postscript drawing in inches. If both arguments are specified, it sets the x and y sizes (inches). And if only the first argument is specified (second is -1), it sets the x-size and the y-size will be automatically computed using the x/y size proportion of the window to draw.

  - **ps** `linewidth [<linewidth>]`

    Sets/Gets the linewidth used for postscript. A width of 1 is an average linewidth.

• **setbinding** `<bindingGroupName> *gclass* ((left | middle | right)Button('' | Up | Down | Motion) | motion | leave | enter | enterLeave | key | keyUp | keyDown | draw | delete | error) ['key1 .. keyN'] [<modifiers>] <script>`
Defines event bindings associated to gobjects of class *gclass*. Read the manual pages for examples of bindings.

• **setcolor** `[-bg <color>] [-fg <color>] [-mouse [<color>]]`
Sets any (or all) of the background (-bg), foreground (-fg) or mouse (used for inverse mode on Unix computers only) color.

• **setg** `[-] *gobjectRegexp* -*field1* [..<val>..] ... -*fieldN* [..<val>..>]`
Sets/Gets the fields *field1*...*fieldN* of all the gobjects whose name matches *gobjectRegexp*. If no <val> are specified then just one field should be used and its value is returned. Otherwise, it sets the fields *field1*...*fieldN* with the respective values specified by the <val>s (a single field might need several <val> to be set). In any case 'setg' returns 0 if the field does not exist. If a field is changed then it can return either 1 or -1. If it returns 1, it means that the object should be redrawn, thus, consequently, the corresponding gobject is stored in the current update table if any. If it returns -1, it means that the gobject should not be redrawn and thus it it is not stored in the current update table. In any case, if one wants to keep lastwave from storing the gobject in the current update table (i.e., from redrawing it later), one must insert a '-' before the object name. Let us note that one can send messages to the sons of any gobject that matches *gobjectRegexp* on the same command line by inserting in the command line the string '-*gobjectRegexp*' followed by a regular 'setg' list of argument. For instance, if you want to set the background color of any gobject inside the window 'win' to black and at the same time change the foreground color of the object 'win.view' to red, you would type "setg win -..* -bg 'black' -.view -fg 'red'"

• **setgu** `*gobjectRegexp* -*field1* [..<val>..] ... -*fieldN* [..<val>..>]`
Same as the 'setg' command but it updates the display of the corresponding gobjects. It is basically equivalent to perform successively : 1- 'gupdate start' 2- a 'setg' and 3- a 'gupdate do'. Thus any gobject which has changed will be redrawn.

• **system**

  • **system** `forceKeyUp`

    Declares that the keyboard does not send keyup events (For Macintosh computers only).

  • **system** `mouse1`

    Declares that the mouse is a 1-button mouse (For Macintosh computers only).

  • **system** `mouse3`

    Declares that the mouse is a 3-button mouse (For Macintosh computers only).

- **system** `nEvents`

  Gets the total number of events that were managed up to the current time.

- **system** `noForceKeyUp`

  Declares that the keyboard does send keyup events (For Macintosh computers only).

- **window**

  - **window** `list [<regexp>='*']`

    Gets the list of all the windows whose name matches <regexp>

  - **window** `new [<class> = 'Window'] <name> [list of fields...]`

    Creates a window of class <class> named <name> and sends all the [list of fields...] to the 'setg' command.

1.4 Script Commands

1.5 Graphic class `GList` (inherits from `GObject`)

Graphic Class which allows to group a list of gobjects
• **setg** `*GList* -add <name> <class> [-<fields>...]`
Adds a gobject named <name> of class <class> to the glist. After creating it, it sets its field according to [-<fields>...].

- **msge**

  - **msge** `*GList* add <name> <class> [<list of field initializations>]`

    Adds to the glist a gobject of class <class> and name <name>. After creating it, it sets its field according to [-<fields>...].

  - **msge** `*GList* list [<filterName>=*]`

    Gets all the gobject names of the glist that matches <filterName>.

  - **msge** `*GList* object <x> <y>`

    Gets the gobject name of the glist that is the closest to the (local coordinate) point <x> <y> (if none it returns the empty string '').

  - **msge** `*GList* remove [<filterName>=*]`

    Removes all the gobjects of the glist whose name matches <filterName>.

1.6 Graphic class GObject

Basic Graphic Class which any other class inherits from
• **setg**

- **setg** *GObject* -apos [<x> <y>]

  Sets/Gets the position of the gobject using absolute coordinates.

- **setg** *GObject* -arect

  Gets the bounding rectangle (x,y,w,h) of the gobject using absolute coordinates (this is equivalent to calling '-apos' and '-asize').

- **setg** *GObject* -asize [<w> <h>]

  Sets/Gets the size of the gobject using absolute coordinates.

- **setg** *GObject* -bg [<color>]

  Sets/Gets the background color of the gobject.

- **setg** *GObject* -class

  Gets the gclass name of the gobject.

- **setg** *GObject* -clip [no | yes | screen]

  Sets/Gets the field which specifies whether the gobject should be clipped (using its bounding rectangle) or not. If it is 'no' (or 0) it means that the gobject is never clipped. if it is 'yes' (or 1) then it is always clipped. If it is 'screen' it means that it is clipped only when displayed on the screen (not in postscript files). This last mode is used for Text gobjects (since the postscript fonts and screen fonts do not really correspond, the screen clipping rectangle is not appropriate for being used in postscript).

- **setg** *GObject* -depth [<depth>]

  Sets/Gets the 'depth' of the gobject. The possible values of this field are either 'back' (gobject is in the back) or 'front' (gobject is in the front) or any number between 1 (which is the same position as 'back') up to the number of gobjects in the father glist.

- **setg** *GObject* -fg [<color>]

  Sets/Gets the foreground color of the gobject.

- **setg** `*GObject* -font [<fontName>]`

  Sets/Gets the name of the font used for drawing the gobject.

- **setg** `*GObject* -frame [<flagOnOff>]`

  Sets/Gets the frame flag which allows (if <flagOnOff> =1) to draw a frame around the gobject.

- **setg** `*GObject* -grid [<xGrid> <yGrid> <wGrid> <hGrid>]`

  If used with no argument, it gets the grid coordinates of the gobject (if they are activated) or it returns the empty string ". If used with the four arguments <xGrid> <yGrid> <wGrid> <hGrid>, it sets the gobject on the corresponding grid coordinates and activate automatically the -grid? flag.

- **setg** `*GObject* -grid? [<flagOnOff>]`

  Activates (<flagOnOff>=1) or deactivates (<flagOnOff>=0) the grid coordinates of the gobject.

- **setg** `*GObject* -hide [<flagOnOff>]`

  Sets/Gets the flag which specifies whether the gobject is visible or not.

- **setg** `*GObject* -line [{dash plain}]`

  Sets/Gets the type of line that will be used for drawing the gobject (for now only one type of dash is available).

- **setg** `*GObject* -name`

  Gets the full name of the gobject.

- **setg** `*GObject* -pen [<penSize>]`

  Sets/Gets the pen size used for drawing the gobject.

- **setg** `*GObject* -pos [<x> <y>]`

  Sets/Gets the position of the gobject using local coordinates.

- **setg** `*GObject* -rectType [<rectType>]`

  This field specifies how the absolute coordinate boundary rectangle of the gobject (i.e., '-arect') must be extended in each direction after doing local to global change of coordinates on the local rectangle (i.e., '-rect'). This is particularly important if you want to control if the boundaries of the local rectangle '-rect' belong or not to the gobject.

At initialization, the <rectType> of a gobject is 'normal' in the sense that boundary rectangle includes the point (x,y) but not the point (x+w,y+h) (thus w and h do represent the width and height in pixels of the gobject). A <rectType> corresponds to four numbers extending the rectangle respectively to the left, the top, the right and the bottom. There are 3 predefined rectangle types (which are associated to names instead of 4 numbers) : LargeRect (=0,0,1,1) (both the points (x,y) and (x+w,y+h) belong to the rectangle), SmallRect (= -1,-1,0,0) (none of the points (x,y) and (x+w,y+h) belong to the rectangle) NormalRect (=0,0,0,0) (the point (x,y) belongs to the rectangle whereas the points (x+w,y+h) does not belong to the rectangle). Let us note that, in a View, if the y-axis (resp. the x-axis) is reversed then the top-bottom (resp. left-right) are reversed.

- **setg** `*GObject* -rect`

  Gets the bounding rectangle (x,y,w,h) of the gobject using local coordinates (this is equivalent to calling '-pos' and '-size').

- **setg** `*GObject* -size [<w> <h>]`

  Sets/Gets the size of the gobject using local coordinates.

- **msge**

  - **msge** `*GObject* back`

    Sends the gobject to the back.

  - **msge** `*GObject* class <class>]`

    If <class> is not specified then it returns the class of the gobject. Otherwise it returns 1 if the gobject is of class <class> (possibly inherited) and 0 otherwise.

  - **msge** `*GObject* delete`

    Deletes the gobject.

  - **msge** `*GObject* draw [(-l | -g) <x> <y> <w> <h>] [-clip]`

    Draws the gobject. If '-l' (resp. '-g') is set , the gobject is drawn only in the local (resp. global) rectangle <x> <y> <w> <h>. If '-clip' is set the clip rectangle of the gobject is used otherwise the current clip rectangle is used.

- **msge** `*GObject* exist`

  Returns 1 if the gobject exists and 0 otherwise.

- **msge** `*GObject* father`

  Gets the name of the father of the gobject or the empty string " if none.

- **msge** `*GObject* front`

  Sends the gobject to the front.

- **msge** `*GObject* g2l <x> <y>`

  Converts global position <x> <y> to local (gobject) position.

- **msge** `*GObject* hide`

  Hides the gobject.

- **msge** `*GObject* id`

  Returns a unique identificator string (alphanumeric string that can be used as an index of an array) that corresponds to the gobject.

- **msge** `*GObject* isin <x> <y>`

  Returns 1 if the local point <x> <y> belongs to the gobject and returns 0 otherwise.

- **msge** `*GObject* l2g <x> <y>`

  Converts local (gobject) position <x> <y> to global position.

- **msge** `*GObject* move <x> <y>`

  Moves the gobject to local position <x> <y>.

- **msge** `*GObject* name`

  Returns the full name of the gobject.

- **msge** `*GObject* pmove <dx> <dy>`

  Moves the gobject of <dx> <dy> pixels.

- **msge** `*GObject* resize <w> <h>`

  Resizes the gobject to the new (local) size <w> <h>.

- **msge** `*GObject* show`

  Shows the gobject.

Bindings

- f5 : creates a Text object in the gobject and just type in (try 'tab', 'up' and 'down').
- f1 = Delete the window the mouse is in
- delete = Delete the graphic object the mouse is in
- h = Display successively field values/Messages/binding help
- b = Display/hide binding help
- Shift + Ctrl + Middle Button = Move Gobject

1.7 Graphic class `Grid` (inherits from `GList`)

GList Graphic Class which allows to display its gobjects using grid coordinates. The glist is divided into 'm' columns and 'n' lines. The gobjects inside this glist can thus be placed and sized specifying column and row numbers (using the '-grid' field). Everything will be automatically resized and repositionned whenever the glist is resized or moved. Let us note that one can specify margins to the grid using the '-margin' fields

• **setg**

- **setg** `*Grid* -dxdy [<dx> <dy>]`

  Sets/Gets the horizontal (<dx>) and the vertical (<dy>) margins between each cell of the grid.

- **setg** `*Grid* -margin [<left> <top> <right> <bottom>]`

  Sets/Gets the margins for the Grid. All the gobjects that will use grid coordinates will be placed inside the rectangle <left> <top> <right> <botttom>. Thus, for instance, the grid coordinates 1,1 corresponds to the point <left> <top>.

- **setg** `*Grid* -mn [<m> <n>]`

  Sets/Gets the horizontal (<m>) and the vertical (<n>) size of the Grid. These two numbers must be strictly positive integers

1.8 Graphic class View (inherits from GList)

GList Graphic Class which allows to display its gobjects using float coordinates. The bounding rectangle of a view corresponds to float bounds referred to as <xMin> <xMax> <yMin> and <yMax> that one can set using the '-bound' field

• **setg**

- **setg** `*View* -?bound [<xMin> <xMax> <yMin> <yMax>]`

  Read only field that works exactly like '-bound' except that it does not change the fields <xMin> <xMax> <yMin> and <yMax> (even when arguments are specified). This allows, for instance, to question the view about the <yMin> and <yMax> within a given range <xMin> <xMax> without changing the view.

- **setg** `*View* -bound [<xMin> <xMax> <yMin> <yMax>]`

  Sets/Gets the boundaries of the View. When setting the rectangle you can use the special values '*' or '?' for any of the 4 arguments. The value '*' means that the corresponding value should not be changed. The value '?' means that the corresponding value should be set in order to frame exactly all the gobjects in the View. Thus for instance, if signals are displayed in the View, '-bound 0 1 ? ?' will display all the signals between abscissa <xMin>=0 and <xMax>=1 using their minimum and maximum values (between 0 and 1) to set the y-scale. Let us note that when you call '-bound' with arguments it changes the boundary rectangle of the View. If you just want to know what the boundary rectangle would be set to if '-bound' (with some specific arguments) was called (without changing the boundaries of the View), you must uset the '-?bound' field (e.g., '-?bound 0 1 ? ?')

- **setg** `*View* -reverse [x | y | xy | none]`

  Gets/Sets the state that indicates which axis are reversed compared to the regular axis of GLists. If it is 'y' (the default value at initialization) then the y-axis will be going from bottom of the window to top, if 'x' the x-axis will be going from right to left, if 'xy' both will be combined and if 'none' the y-axis will be top to bottom and the x-axis left to right (as for GLists)

Bindings

- Type 'c' to change cursor mode

- 'z' : changes the zoom mode just type 'z'
- Left/Right/Middle button : operate the zoom

1.9 Graphic class `Window` (inherits from `Grid`)

Basic Window Class
• **setg** `*Window* -title [<title>]`
Sets/Gets the title of the Window. At initialization the title is the name of the gobject.

• **msge** `*Window* drawps <file.ps>`
Creates a (color) postscript file of the window. See command 'ps' to control the postscript output.

Chapter 2

Package `disp` 2.0

Package that allows high level displays of any gobject which is associated to a variable's content (such as signals, wtrans1d, books...).
*** Authors and Copyright : E. Bacry*

2.1 Script Commands

• **SetCursorBindings** (in file `scripts/disp/cursor`) `*className* <listv of procedures>`
class must be a valid classname of a class which corresponds to objects that can be displayed in Views. When called this function allows to manage cursor display when the mouse points on an object of class *className*. The <listv of procedures> is a listv of different display procedures. You can switch from one to the next using the 'c' key. A valid disply procedure must have one argument 'cursor'. This argument corresponds to an array which has 2 fields, one is an input field :
- view : the name of the view gobject
and the other one is an output field (you are supposed to set it) - erase : the script that should be called to erase the cursor you displayed
The procedure should
- manage drawing the cursor (generally using the variables @x and @y) (You can use the _ViewDrawCrossHair procedure for that purpose (see examples in signal.pkg)
- set the cursor.erase variable to a valid script that will erase the drawing

• **SetSuperposeBindings** (in file `scripts/disp/superpose`) `*gclass*`

Allows superposing drawing to graphic class *gclass*. If you call this procedure then each time a gobject of class *gclass* will be drawn, you can ask that a script is called (e.g., to draw something on top of it). The scripts are specific to the *gobject* themselves. However, you can group them into groups to delete them at once. You can use the procedure 'SuperposeAdd1' to trigger a script for a specific gobject and 'SuperposeDelete' to delete a group of scripts

• **SetZoomBindings** (in file `scripts/disp/zoom`) `*class* <listv of modes>`
class must be a valid classname of a class which corresponds to objects that can be displayed in Views. <listv of modes> is a listv that combines the 4 strings 'rect', 'xrect', 'yrect' or 'normal'. Calling this function enables some mouse bindings so that zoom can be performed when the mouse points to this object. The 'z' key will switch between the different zoom modes.
- 'normal' : uses the left/right/middle button (as default for signals)
- 'rect' : uses the left/middle button (as default for images)
- 'xrect' : same as 'rect' but constraint the x-direction

• **SuperposeAdd1** (in file `scripts/disp/superpose`) `*gobject* <group> <script>`
Adds a script to be executed whenever the gobject *gobject* is drawn. The <group> is the name of the group the scripts belongs to.

• **SuperposeDelete** (in file `scripts/disp/superpose`) `*gobject* <group>`
Deletes all the scripts associated to the group <group>.

• **disp** (in file `scripts/disp/disp.pkg`) `[<window>] [<theVariables>] [<setg fields>]`
This function is a high level functions for displaying all sorts of objects such as signals, wavelet transforms, extrema representation in the same window using (almost) any layout. The display takes place in a <window> of class WindowDisp. Each WindowDisp has a 'type' which corresponds to the type of objects it currently displays. A WindowDisp with only signals in it will be of type '&signali', one with only 1d extrema representation will be of type '&extrep'... A WindowDisp that mixes objects of different types is of type '&mixed'. The type of a WindowDisp is indicated in its title which is of the form '<windowName> (<type>)'. A WindowDisp is a window that contains one 'FramedView' for each graph to be displayed. These FramedViews are named fv1,..,fvN. A FramedView consists basically in a View (named 'view') in the middle and a one line text-box of type Box (named

'box') at the bottom. Moreover it automatically draws axis around the 'view'. This 'view' is actually not directly of type 'View' but it is of type 'EView' (for Extended View) which directly inherits from the class 'View'. It extends the View class in order to take care of eventual x and y-scale synchronizations between different EViews (c.f. the 'synchro' command). Let us note that if 'disp' is sent with no argument (or just <window>) it just refreshes the current window, i.e., the last window visited by the mouse or in which drawing was performed. The arguments of the 'disp' command are the following : - <window> : The WindowDisp the display will take place in. If it is not specified, then, the last WindowDisp of the right type that has been used or that has been visited by the mouse will be used. If such a window does not exist, it will be automatically created. The position and size of the newly created WindowDisp is the one specified in the global variable 'disp.<type>.rect' where <type> is the type of the WindowDisp to be created (without the first '&' character). Thus, for instance, in the 'scripts/signal.pkg' file, the first line is 'disp.signali.rect={20 55 330 330}'. Let us note that, if you want 'disp' to use a new type of window (that inherits from WindowDisp, in which you would have added, for instance, some buttons), you can specify it by changing the global variable (in the 'scripts/disp/disp.pkg' file) :'disp.windowClass'. Actually you can also specify for each <type> (e.g., signali, wtrans,...) a disp.<type>.axis variable which (dis)able axis drawing, a disp.<type>.reverse variable to set the 'reverse' field of the corresponding framed views and in the same way the disp.<type>.margin (which is used only if the type of the window disp is not mixed). - <theVariables> : The variables appearing here are 'extended' variables (such as the signal '0a' or the wavelet transform 'a'). The way it is organized controls the way it will appear on the screen. This argument can be seen as a succession of lists. The window will be divided vertically in as many sections as the number of lists. Each list 'represents' one of this 'horizontal' section (from top to bottom). Moreover, each of these lists are themselves organized as lists of extended variables. Thus, each horizontal section will be divided (vertically) in as many sections as the length of its corresponding list. Thus, each of the so-obtained sections corresponds to a variable list. The command 'disp' displays all the objects corresponding to each variable list superposed in a FramedView in its corresponding section. For instance, if one wants to display 4 objects (three signals) '0','1','2' and (a wavelet transform) 'a' one on top of the other one, one would simply type 'disp 0 1 2 a'. Now if one wants the first 2 ones to be at the same horizontal level, one needs to type 'disp {0 1} 2 a'. Just play around with it.... you will master it very easily. Let us note that if this argument is not specified

then it will just redraw the corresponding <window> using the fields of the next argument. This allows to change the way some objects are currently displayed on a window. - <setg fields> : It corresponds to a succession of arguments of the 'setg' command. Actually, the following command will be executed 'setg <window> <setg fields>' where <window> is the name of the window the display takes place in. One has to know that the FramedViews are named fv1,....,fvN and the graphic objects corresponding to the variables are simply named 1,2,...n (in the same order as they appear in the 'disp' command). Thus if one wants to display the signal '0' on top of the wavelet transform 'a' and makes the signal '0' appear as big red dots and 'a' use the 'global' normalization mode, one would type 'disp 0 a -..1 -fg red -curve o -4 -..2 -norm global'. One must look to the documentation concerning the different gobjects (GObject, GraphSignal, GraphWtrans,....) to know about the fields they support. You can also change the fields of the WindowDisp, e.g., 'disp 0 a -x 0 1 -..1 -fg red' which displays the signal '0' in red and between abscissa 0 and 1 (the 'x' field is a WindowDisp field).

- **synchro** (in file `scripts/disp/eview`)

  - **synchro** `add {x | xy} <gobject> <eviewList>`

    Adds a x or/and y-scale synchronisation between eviews of <eviewList>. Each time the boundaries of an eview in <eviewList> is changed, all the other ones will be changed accordingly. If 'x' is the first argument then only the xMin and xMax boundaries are synchronized. If it is 'xy' then xMin, xMax, yMin and yMax are synchronized. If <gobject> is not "" then this synchronization is automatically deleted when <gobject> is deleted. Let's note that one can use wild card characters in <eviewList>. This command returns a number that is associated to this newly defined synchronization (This number can be used with the delete action). ** Warning : Only one synchronization definition (the first one found) will be used when changing the boundaries of an eview. They are not called recursively.

  - **synchro** `delete <n>`

    Deletes the synchronization number <n>

  - **synchro** `get <eviewName>`

    Gets the synchronization definition that must satisfy the eview <eviewName>. The first one found is returned.

  - **synchro** `list`

Displays all the currrently active synchronization

2.2 Graphic class EView (inherits from View)

Graphic Class that extends the View class so that one can synchronize the x or/and y-scales between different EViews. This class is used in FramedViews.

• **setg**

- **setg** *EView* -bound <xMin> <xMax> <yMin> <yMax>

  Changes the boundaries of the eview and synchronizes eventual other EViews.

- **setg** *EView* -boundNoSync <xMin> <xMax> <yMin> <yMax>

  Changes the boundaries without synchronization of eventual other EViews

- **setg** *EView* -string [<string>]

  Works exactly the same way as '-string' field for the Box inside the FramedView which contains the EView.

2.3 Graphic class FramedView (inherits from Grid)

Graphic Class to display a View (actually it uses EViews) with axis and a text box at the bottom. It is used in WindowDisp and thus by the 'disp' function.

• **setg**

- **setg** *FramedView* -axis [<flagOn>]

  Sets/Gets the flag that allows the axis to be displayed or not.

- **setg** *FramedView* -axisFont [<font>]

  Sets/Gets the font used for axis.

- **setg** *FramedView* -axisFrame [<flagOn>]

  Sets/Gets the flag that allows that four axis (instead of 2) are displayed around the view.

- **setg** *FramedView* -bound [<xMin> <xMax> <yMin> <yMax>]

  Just send the message to the '-bound' field of the EView inside the FramedView.

- **setg** *FramedView* -boundNoSync <xMin> <xMax> <yMin> <yMax>

  Just send the message to the '-bound' field of the View class of the EView inside the FramedView. It thus changes the boundaries without synchronization of other views.

- **setg** *FramedView* -graph <exprString>

  Sets the graph corresponding to <exprString> that will be displayed in the FramedView.

- **setg** *FramedView* -graph+ <exprString>

  Adds the graph corresponding to expression <exprString> to be displayed in the FramedView on top of the other ones.

- **setg** *FramedView* -graph1 <exprString>

  Same as graph but do not update the boundaries of the FramedView.

- **setg** *FramedView* -graph1+ <exprString>

  Same as graph+ but do not update the boundaries of the FramedView.

- **setg** *FramedView* -reverse [x | y | xy | none]

  Gets/Sets the state that indicates which axis are reversed compared to the regular global axis. If it is 'y' (the default FramedView value at initialization) then the y-axis will be going from bottom of the window to top, if 'x' the x-axis will be going from right to left, if 'xy' both will be combined and if 'none' the y-axis will be top to bottom and the x-axis left to right (as for GLists).

- **setg** *FramedView* -string [<string>]

  Works exactly the same way as '-string' field for the Box inside the FramedView.

- **setg** *FramedView* -ticksIn [<flagOn>]

  Sets/Gets the flag that allows the ticks on the axis to be inside the view instead of outside.

- **setg** `*FramedView* -title [<string>]`

  Sets/Gets the title string. Let us note that it creates a Text object named 'title' that you can adress/move direcly.

- **setg** `*FramedView* -xLabel [<string>]`

  Sets/Gets the label string for the x-axis. Let us note that it creates a Text object named 'xlabel' that you can adress/move direcly.

- **setg** `*FramedView* -yLabel [<string>]`

  Sets/Gets the label string for the y-axis. Let us note that it creates a Text object named 'ylabel' that you can adress/move direcly.

• **msge** `*FramedView* clear`
Clears the FramedView.

2.4 Graphic class `RectSelect` (inherits from `Box`)

Graphic Class to display a rectangular selection of a view that can be editable using the mouse. Some graphic objects allow the user to create a RectSelect performing a drag and drop with the left mouse button along with the ctrl key. For instance, if the sound package has been loaded, one should be able to create, in this way, a selection of a signal displayed in a window. This means that the sound package allowed the GraphSignal objects to manage these creation events. In order to make a graphic class allow these events, use the '_SetRectSelectBindings' procedure. Once a RectSelect has been created, it can be edited using the mouse : if you drag and drop a corner of the rectangle you move the selection, otherwise, you can drag and drop any border of the selection that is editable (the sound package makes only the vertical borders editable). When the mouse is 'close' to any border of the selection, it will become active, the active zone is lit up.
• **setg**

- **setg** `*RectSelect* -edit [x | xy]`

  Sets/Gets the 'edit' mode of the SelectRect. If it is 'x' then just the vertical borders can be moved, otherwise any border can be moved

- **setg** `*RectSelect* -thickness [<thickness>]`

  Sets/Gets the thickness of the object. It corresponds to how close (in points) the mouse should be from the borders to be able to edit it.

The mouse is considered in the object only if it is close enough to the borders.

- **setg** `*RectSelect* -type [image | signal]`

  Sets/Gets the type of the SelectRect. If it is 'signal' it means that the rectangle will be displayed below the objects which are in the view using a grey background. If it is 'image' it will be displayed above the other objects. This is meant so that the SelectRect can be seen on the screen. Do not change this value after creating the SelectRect.

- **setg** `*RectSelect* -yRescaled`

  Gets an internal flag used by the EView objects (it is read only). When the flag is 1 it means that the object must be automatically rescaled in y in order to reach the min/max y-values when the view is zoomed.

Bindings

- Ctrl + Left button = Allows to edit the RectSelect by drag/drop on any border.)

2.5 Graphic class `WindowDisp` (inherits from `Window`)

Graphic Class to manage windows used by the 'disp' function. It implements a whole system for remembering which window was the last used for displaying a type of variables.

• **setg**

- **setg** `*WindowDisp* -S [<flagOnOff>]`

  Same as '-superpose'.

- **setg** `*WindowDisp* -s [{x xy none}]`

  Same as '-synchro'.

- **setg** `*WindowDisp* -superpose [<flagOnOff>]`

  Sets/Gets superposition flag. If 1 then all the eviews are superposed.

- **setg** `*WindowDisp* -synchro [{x xy none}]`

  Sets/Gets the synchronization mode for the WindowDisp. - 'x' means that the abscissa scale of all the EViews will be synchronized - 'xy' means that the abscissa and the ordinate scales of all the EViews will be synchronized - 'none' means that no synchronization is made.

- **setg** *WindowDisp* -type [<type>]

  Sets/Gets the type of the WindowDisp. It is defined as the type of variables it displays or '&mixed' if it displays variables of different types.

- **setg** *WindowDisp* -x [<xMin> <xMax>]

  Sets/Gets the xMin and xMax used in the WindowDisp. You can use '*' and '?' characters in the same way as for '-bound' fields of views.

- **setg** *WindowDisp* -y [<yMin> <yMax>]

  Sets/Gets the yMin and yMax used in the WindowDisp. You can use '*' and '?' characters in the same way as for '-bound' fields of views.

• **msge** *WindowDisp* clear
Clears the WindowDisp.

Chapter 3

Package misc 2.0

Package that regroups very useful miscellenaous script commands and graphic classes
*** Authors and Copyright : E. Bacry*

3.1 Script Commands

• **Demo** (in file `scripts/misc/miscScripts`)
Command that explains how to run the demos of the different numerical packages

• **StartDemo** (in file `scripts/misc/miscScripts`) `*baseProcName*`
When calling this function it will print a one-line-help that tells you how to navigate through the demo : 'N' for next step 'R' for repeat and 'P' for previous step. At each step of the demo, the procedure which name is the '_' character followed by *baseProcName* followed by the step number (starting from 0) is called until such a procedure does not exist in which case the Demo is supposed to be over.

• **cmdisp** (in file `scripts/misc/color`) `[<colormapName>=<current>]`
Displays the colormap in a window.

• **cminit** (in file `scripts/misc/color`) `[[-]<numOfColors>=100] [<flagColor>=1] [<colorMapName>=color]`
Sets the colormap <colormapName> with <numOfColors> colors from black to red (going through blue) or (if <numOfColors> < 0) from red to black. If <flagColor>==0 then grey levels are used from black to white or white to black (if <numOfColors> < 0)

• **nice** (in file `scripts/misc/miscScripts`) <val>
Recursive display of an array or a listv

• **wait** (in file `scripts/misc/button`) <delay>
Returns after a certain numlber of seconds specified by <delay>

3.2 Graphic class `Box` (inherits from `GObject`)

Graphic Class to display (clipped) text in a box
• **setg**

- **setg** `*Box* -centered [<flagOnOff>]`

  Sets/Gets the centered flag. If 1 then the string will be centered in the box both vertically and horizontally. If 0 then it will be centered vertically but justified at a distance of 5 to the left border.

- **setg** `*Box* -string [<val>]`

  Sets/Gets the string that will be displayed in the box.

• **msge** `*Box* set <val>`
Changes the string that is displayed.

3.3 Graphic class `Button` (inherits from `GObject`)

Graphic Class to implement buttons
• **setg**

- **setg** `*Button* -behavior [switch | simple]`

  Sets/Gets the type of button. It is either a 'switch' that has 2 positions (On and Off) or a 'simple' button which behaves like a trigger.

- **setg** `*Button* -colorOff [<color>]`

  Sets/Gets the color that will be used when button is Off.

- **setg** `*Button* -colorOn [<color>]`

  Sets/Gets the color that will be used when button is On.

- **setg** `*Button* -draw [<drawProcedure>]`

  Sets/Gets the procedure name that will be called to draw the button. This procedure must have 1 argument which corresponds to the name of the button to be drawn. There are two already defined procedure : '_GButtonDrawPlain' for 'plain' button and '_GButtonDraw3d' for 3d buttons (it assumes that the background is 'grey'

- **setg** `*Button* -handle [<handleProcedure>]`

  Sets/Gets the procedure name that will be called whenever the button is pushed. This procedure must have 2 arguments which corresponds to the name of the button to be drawn and its state.

- **setg** `*Button* -state [<flagOnOff>]`

  Sets/Gets the state flag (1 corresponds to the button pushed).

- **setg** `*Button* -title [<title>]`

  Sets/Gets the label of the button.

- **msge**

  - **msge** `*Button* off`

    Turns the button off.

  - **msge** `*Button* on`

    Turns the button on.

  - **msge** `*Button* push`

    Pushes the button.

Bindings

- Click on the button using the left mouse button

3.4 Graphic class `Colormap` (inherits from `GObject`)

Graphic Class to draw colormaps
• **setg** `*Colormap* -cm [<colorMap>]`
Sets/Gets the colormap.

3.5 Graphic class Line (inherits from GObject)

Graphic Class that corresponds to a simple line
• **setg**

- **setg** *Line* -point1 [<x1> <y1>]

  Sets/Gets the coordinate of the first end point.

- **setg** *Line* -point2 [<x2> <y2>]

  Sets/Gets the coordinate of the second end point.

- **setg** *Line* -slope

  Gets the slope of the line.

3.6 Graphic class Numbox (inherits from Box)

Graphic Class to display a numerical value in a box
• **msge** *Numbox* set <val>
Changes the value that is displayed.

3.7 Graphic class Shape (inherits from GObject)

Graphic Class to draw a rectangle or an ellipse. The position can be set using the '-pos' field which corresponds either to the center of the shape (if the '-center' field is 1) or a corner of the framing rectangle (if the '-center' field is 0). In both cases, the '-radius' field allows to set the width and height of the shape. These distances are expressed either using local coordinates (if the '-pixel' field is 0) or as a number of pixels (if the '-pixel' field is 0)
• **setg**

- **setg** *Shape* -centered [(0 | 1)]

  Sets/Gets the centered flag. If it is 1 (which is the default value), it means that the shape will be centered at the point specified by the '-pos' field with radii specified by the '-radius' field. If it is 0 then it means that the shape will be framed in a rectangle which one corner is specified by the '-pos' field and the width and height are specified by the '-radius' field.

- **setg** `*Shape* -filled [(0 | 1)]`

  Sets/Gets the field which indicates whether the shape is filled or not.

- **setg** `*Shape* -pixel [(0 | 1)]`

  Sets/Gets the pixel flag. If it is 0 (which is the default value), it means that the radii are specified using local coordinate otherwise the radii indicate a number of pixels in each directions.

- **setg** `*Shape* -radius [<rx> <ry>]`

  Sets/Gets the radii. By default these are specified using local coordinate. If you want to specify them using a number of pixels, you should use '-pixel' field.

- **setg** `*Shape* -shape [(rect | ellipse)]`

  Sets/Gets the name of the shape to be drawn.

3.8 Graphic class `Text` (inherits from `GObject`)

Graphic Class to draw text without any box around it. The position if the text is precisely controlled.

• **setg**

- **setg** `*Text* -flagMax [(0 | 1)]`

  Sets/Gets the 'flagMax'. It changes the way the bounding rectangle is computed when the string to be displayed is changed. If it is 0, then the bounding rectangle is recomputed each time. If it is 1 then the union of the old bounding rectangle and the new one is used. It allows to avoid full redrawing.

- **setg** `*Text* -margin [<left> <top> <right> <bottom>]`

  Sets/Gets the margins (in pixels) around the string (at init they are 0).

- **setg** `*Text* -posMode [<hPosMode>] [<vPosMode>]`

  Sets/Gets the horizontal and vertical position modes that are used to display the string. To learn about these modes, you should read the help of the 'draw string' command. Let us note that the '-pos' coordinates of the Text corresponds to the coordinates sent to the 'draw string' command and thus does not correspond to the top left

corner of the bounding rectangle (you directly adress the position of the text).

- **setg** `*Text* -string [<string>]`

  Sets/Gets the string that is displayed.

Bindings

- In the edit mode, any character can be typed.
- up/down : change the font size
- tab : switch bold/plain
- escape : enters/leaves the edit mode.

Chapter 4

Package `signal` 2.0

Package allowing to deal with signals.
*** Authors and Copyright : E.Bacry, N.Decoster and X.Suraud*

4.1 Defined types

4.1.1 Type `&signal`

This type is the basic type for signals. Uniformly sampled signals (i.e., Y-signals) can be built using the <value1,...,valueN> syntax. The values can be either a float, a signal, a range or a listv of floats, signals and ranges. The different operators are
- +,-,*,/ (and +=,-=,*=,/=) : regular operators
- ==,!=,<=,>=,<,> : regular tests
- x^ f (and ^ =) : each value of |x| is taken to the popwer f
- x*^ n : each value of x to the power n where n is a positive integer
- ~ : transposition operator (returns a single columnimage)
- //,% : integer division and remainder
- is,isnot : test if 2 signals correspond or not to the same C object
- sinh,sin,cosh,cos,tanh,tan,acos,asin,atan : trigonometric operators
- min,max : if 1 argument, returns the min or max value of a signal, if 2 arguments returns the signal made of the min/max of each value.
- log2,log,ln,sqrt,abs,exp,ceil,floor,round,frac,int : other math functions
- der,prim : derivative and primitive of a signal
- sum : computes the sum of all signal values
- mean : same as sum but divides by the total number of points
- any : returns 1 if at least one of the values is different from 0
- all : returns 1 if all of the values are different from 0

- find : returns a signal made of indices which correspond to non 0 values
- YSIG Constructors : <...>,Zero,One,I,Grand,Urand
- XYSIG Constructors : XY(xsignal,ysignal). In the ysig expression you can use the 'X' notation which refers to xsig.

- `&signal [*opt,...] [:]= (<float> | <range> | <signal> | <listv>)`
  Get/Set the signal values
  Options are : `*nolimit,*b0,*bconst,*bmirror,*bmirror1,*bperiodic,*x,*xlin`
  - *nolimit : indexes can be out of range
  - *b0 : border effect with 0 value
  - *bconst : border effect with constant values (last signal value for right handside and first signal value for left handside)
  - *bperiodic : periodic border effect)
  - *bmirror1 : mirror+periodic border effect (first and last points are repeated)
  - *bmirror : mirror+periodic border effect (first and last points are NOT repeated))
  - *x : index values are replaced by x-values. Interpolation is piecewise constant
  - *xlin : index values are replaced by x-values. Interpolation piecewise linear
- `&signal.X [*opt,...] [:]= (<float> | <range> | <signal> | <listv>)`
  Get/Set the X field of a signal
  Options are : `*nolimit,*b0,*bconst,*bmirror,*bmirror1,*bperiodic,*x,*xlin`
  - *nolimit : indexes can be out of range
  - *b0 : border effect with 0 value
  - *bconst : border effect with constant values (last signal value for right handside and first signal value for left handside)
  - *bperiodic : periodic border effect)
  - *bmirror1 : mirror+periodic border effect (first and last points are repeated)
  - *bmirror : mirror+periodic border effect (first and last points are NOT repeated))

  - *x : index values are replaced by x-values. Interplation is piecewise constant
  - *xlin : index values are replaced by x-values. Interplation piecewise linear

- `&signal.Y [*opt,...]  [:]= (<float> | <range> | <signal> | <listv>)`
  Get/set the Y field of a signal
  Options are : `*nolimit,*b0,*bconst,*bmirror,*bmirror1,*bperiodic,*x,*xlin`

  - *nolimit : indexes can be out of range
  - *b0 : border effect with 0 value
  - *bconst : border effect with constant values (last signal value for right handside and first signal value for left handside)
  - *bperiodic : periodic border effect)
  - *bmirror1 : mirror+periodic border effect (first and last points are repeated)
  - *bmirror : mirror+periodic border effect (first and last points are NOT repeated))
  - *x : index values are replaced by x-values. Interplation is piecewise constant
  - *xlin : index values are replaced by x-values. Interplation piecewise linear

- `&signal.index [[*nolimit],<xValue>]`
  Get the index corresponding to a given <xValue>
  Options are : `*nolimit`

  - *nolimit : indexes can be out of range

- `&signal.size [= <size>]`
  Sets/Gets the size of a signal. In a case of a Set no initialization is performed. Moreover, if the asked size is smaller than the allocation size no additional allocation is performed.

- `&signal.dx [= <dx>]`
  Sets/Gets the dx of a Y-signal

- `&signal.x0 [= <x0>]`
  Sets/Gets the x0 of a Y-signal

- `&signal.name [= <name>]`
  Sets/Gets the name of a signal

- `&signal.xy [= (0|1)]`
  Sets/Gets 'xy' flag signal. If 0 it means that the signal is a Y-signal, otherwise, it is a XY-signal.

- `&signal.sizeAllocX [= <sizeAllocX>]`
  Gets/Sets the allocation size of the X array of a signal.

- `&signal.sizeAllocY [= <sizeAllocY>]`
  Gets the allocation size of the X array of a signal.

- `&signal.firstp [= <firstp>]`
  Sets/Gets the 'firstp' field of a signal ('firstp' is the index number used for storing the first index affected by border effects).

- `&signal.lastp [= <lastp>]`
  Sets/Gets the 'lastp' field of a signal ('lastp' is the index number used for storing the last index affected by border effects).

- `&signal.tolistv`
  Gets a listv made of the y-values of the signal

4.1.2 Type `&signali`

This type corresponds to non empty signals.

4.2 Commands related to signals

• **cantor** <signalOut> <depth> <s1> <s2> <s3> <p1> <p3>
Generates a 1–>3 cantor (with a hole in the middle) given a number of iterations <depth> the relative sizes <s1>, <s2> and <s3> (must be integers!) of each 3 interval and the weights <p1> and <p3> of the first interval and the last one. This algorithm stops after <depth> iteration. It is not adapted to the local resolution (as for the command 'ucantor')

• **conv** <signal> <filter> <signalout> <border_effect> [-f] [-x [<xmin> <xmax>]]
Computes the convolution of <signal> by the compact support <filter>. Border effect of <signal> can be chosen among 'b0', 'bconst', 'bperiodic', 'bmirror' or 'bmirror1' (same effect as for *option extractions of signals). If

the flag -f is not set, the convolution is computed directly (no FFT) otherwise it is computed using an FFT. If the flag -x is not set, the abscissa interval on which the result is computed is (i) the abscissa interval of <signal> (for 'bperiodic') (ii) double the abscissa interval of <signal> (for 'bmirror' and 'bmirror1') (iii) the abscissa interval + the filter border effect (for 'b0' and 'bconst'). The flag -x allows to fix the abscissa interval <xmin> <xmax> of the result. Default values for <xmin> <xmax> are the abscissa interval of <signal>. The command retursn the time ellapsed (in seconds).

• **fft** `<signalInReal> [<signalInImag>] <signalOutReal> <signalOutImag> [-is]`
Computes the Fourier transform or the inverse Fourier transform (if '-i' is set) of a signal (complex or real) which must has a size which is a power of 2. By default, the fourier transform of a complex signal is supported by [-Fs/2,Fs/2[where Fs = 1/signal->dx is the sample frequency (in Hertz) and has the same number of points as the original signal. If '-s' is set then no shift is performed after the fft algorithm has been performed, thus the resulting fourier transform is represented between [0,Fs[. Consequently, for the inverse fourier transform if '-s' is not set then the input fourier transform is supposed to be between [-Fs/2,Fs/2[and otherwise it should be between [0,Fs[. In the case you want to do the fourier transform of real signal, you should ommit <signalInImag> in the command line. The computed fourier transform is supported by [0,Fs]. It has one more point than the original signal. If you want to invert the so obtained fourier transform, you should ommit <signalOutImag> in the command line and use the '-i' option. The so-obtained real signal will be placed in <signalOutReal>. Let us note that in the case of a real input signal, the option '-s' is useless. In any case, 'fft' returns the elapsed time in seconds.

• **histo** `<signalIn> <signalOut> <n> [-x <xMin> <xMax>] [-y <yMin> <yMax>] [-w <signalWeight>] [-c]`
Computes in <signalOut> the <n> branch histogram of the values of <signalIn>.
-x : Only the values between the abscissa <xMin> and <xMax> are taken into account.
-y : Only the values between the ordinate <yMin> and <yMax> are taken into account.
So the histogram will be made of <n> bars between <ymin> and <ymax>.
-c : Only points corresponding to indexes between firstp and lastp are taken into account.

• **padd** <signalIn> [<signalOut>=<signalIn>] [*border*= (*bconst | *b0 | *bmirror | *bperiodic)] [-s <newSize>]
Padds a signal so that it ends having <newSize> points (if '-s' specified) or the closest size which is greater than the actual size and which is a power 2 (if '-s' not specified). The padding values are specified by the argument <border>. The resulting signal is stored in <signalOut> (by default it is stored in <signalIn>).

• **read** <signalOut> (<filename> | <stream>) [[<xCol>] <yCol>] [-f <firstPoint>] [-s <sizeToRead>] [-r [('little' | 'big')]] [-b]
Reads a signal from a <file> or a <stream>. The file must be in the Last-Wave format (i.e., created with 'write') or in the raw format (option '-r'). If option '-r' is not set then you can specify the column number for the x-values and the column number for the y-values (first column is 1). If none are specified then this command tries to read the first two columns as x and y or, if there is only one colum, directly the y values. If a stream is specicified and if option '-r' is not set then, at the end of the command the stream is positionned at the end of the signal (even if just a few points are read). The options are
-f : It reads the signal starting from index <firstPoint> (first point is at index 0)
-s : It reads only <sizeToRead> values
-r : The data (just the y's) are in raw format (binary floats) with no header. In that case you can specify whether you want to read them as little endian data or big endian data. If nothing is specified the endianness of the computer is used.

• **readinfo** (<filename> | <stream>) [-p]
This command is used in order to get information about a signal file or stream that will be read (later) by the 'read' comand. If '-p' is set then information on the file or stream is printed in a in fully explained english. If it is not set then either it does not return anything (which means that the file is not readable) or it returns a listv which gives some information. The first element of the listv is 0 if there is no header or 1 there is one. If the file has no header then the remaining is of the form 'nColumns' 'size'. If it has one the next element describes the way the values are coded : it is either 'ascii', 'binary little' or 'binary big'. The next element is the type of the signal (either 'x' or 'xy'). If the signal is of type 'y' then the next two elements are the signal fields 'x0' and 'dx'. Finally the last two elements are the sigal fields 'firstp and lastp'.

• **sort** <signalIn>
Sorts the values of <signalIn> according to X (if 'xysig') or Y (if 'ysig')

• **stats**

- **stats** `corr <signalIn> [<signalIn1>] [-c]`

  Computes the correlation function between the x array and the y array of <signalIn> or (if <signalIn1> is specified) between the y arrays of <signalIn> and <signalIn1> ('-c' : same as in 'mean' command).

- **stats** `fit <signalIn> [-x <xMin> <xMax>]`

  Computes a linear fit y = ax+b of <signalIn> between abscissa <xMin> and <xMax>. Returns the listv {a sigma_a b sigma_b iMin iMax}.

- **stats** `kurt <signalIn> [-c]`

  Computes the kurtosis of a signal ('-c' : same as in 'mean' command).

- **stats** `lp <signalIn> <p> [-c]`

  Computes the Lp norm of a signal ('-c' : same as in 'mean' command).

- **stats** `mean <signalIn> [-c]`

  Computes the mean of a signal ('-c' is the "causal" flag : if set, it means that the mean is computed only from the indexes 'firstp' to 'lastp'of <signalIn>).

- **stats** `minmax <signalIn> [-c]`

  Computes the minimum and the maximum values of a signal and sends back the corresponding indexes in a listv {<imin> <imax>} ('-c' : same as in 'mean' command).

- **stats** `nth <signalIn> <n> [-cCa]`

  Computes the (NON centered) <n>th moment of a signal ('-c' : same as in 'mean' command). If '-C' then the moment is centered. If '-a' the absolute moment is computed (<n> can be a float).

- **stats** `print <signalIn> [-c]`

  Prints some statistical information about a signal ('-c' : same as in 'mean' command).

- **stats** `skew <signalIn> [-c]`

  Computes the skewness of a signal ('-c' : same as in 'mean' command).

- **stats** `var <signalIn> [-c]`

  Computes the variance of a signal ('-c' : same as in 'mean' command).

• **thresh** `<signalIn> <signalOut> -(x | y) (<min> | *) (<max> | *))`

Thresholds the <signalIn> and sets the result in <signalOut>. All the values between <min> and <max> are set to 0.
- If <min> is '*' then <min> is chosen as the minimum of <signalIn>
- If <max> is '*' then <max> is chosen as the maximum of <signalIn>

• **ucantor** `<signalOut> <size> <r1> <p1> <r2> <p2> [... <rN> <pN>]`

Generates a 1->N cantor of size <size> down to the smallest resolution (the smaller the interval the greater the number of iterations). If you want not adapt the number of iterations to the local resolution use the 'cantor' command instead. The <ri>'s correspond to the relative size of each interval (the sum must be equal to 1) and the <pi>'s correspond to their respective weights (the sum must be equal to 1).

• **write** `<signalIn> (<filename> | <stream>) [('xy' | 'yx' | 'x' | 'y']) [-h] [-b] [-r [('little' | 'big')]]`

Writes a signal into a <file> or a <stream>. It writes the file either in a raw format (using '-r' option) or (by default) in the LastWave format with ascii coding (unless '-b' is specified in which case binary codes are used).
In the LastWave format, there are different modes :
- 'xy' : First column is X and second is Y (in ascii mode), or the X-values are stored first then the Y-values (in binary mode).
- 'yx' : First column is Y and second is X (in ascii mode), or the Y-values are stored first then the X-values (in binary mode).
- 'y' : A single column made of Y
- 'x' : A single column made of X

By default it uses 'xy' mode for xy-signals and 'y' mode for y-signals. The options are :
-h : No Header (ONLY in ascii mode).
-b : Binary writing (instead of ascii).

4.3 Script Commands

• **dirac** (in file `scripts/signal/signal.pkg`) `<size> [<pos>]`
Returns a dirac function in the middle of the signal.

• **sget** (in file `scripts/signal/signal.pkg`) `*GraphSignalObject*`
Returns the signal displayed in *GraphSignalObject* using the actual boundaries

• **sin** (in file `scripts/signal/signal.pkg`) `<size> [<freq>=1]`
Returns a sinus function with <freq> oscillations.

4.4 Graphic class `GraphSignal` (inherits from `GObject`)

Graphic Class that allows to display signals
• **setg**

- **setg** `*GraphSignal* -causal [<flagOnOff>]`

  Sets/Gets the causal flag. If 1 then it will not display all the values which were affected by border effects.

- **setg** `*GraphSignal* -cgraph [<signal>]`

  Gets/Sets the signal to be displayed by the GraphSignal with a copy of <signal>.

- **setg** `*GraphSignal* -curve [<symbol> [<parameter>]]`

  Sets/Gets the symbol which is used to draw the signal. There are several choices
  - '_' : A plain line is used.
  - '-' : A dashed line is used (since this symbols is also used for specifying fields, for using it in a 'setg' command, you must escape it using two successive '-', e.g., 'setg ..signal -curve - -').
  - '|' : A histogram-type display will be used. The argument <parameter> specifies the y-value the boxes of the histogram will start at (default is 0).
  - '+' : Crosses of size <parameter> will be used.

- **setg** `*GraphSignal* -graph [<signal>]`

  Gets/Sets the signal to be displayed by the GraphSignal with <signal>.

Bindings

- Shift + Right button = FFT spectrum
- Shift + Middle button = linear fit
- Shift + Left button = draw a line (just click once to remove it)
- Type 'c' to change cursor mode
- 'z' : changes the zoom mode just type 'z'
- Left/Right/Middle button : operate the zoom

4.5 Demos

Here is a list of all the Demo files and for each of them all the corresponding Demo commands. To try a Demo command, you should first source the corresponding Demo file then run the command. (When sourcing the Demo file, LastWave tells you about all the commands included in this file).
The Demo files corresponding to this package are :

Demo file **DemoSignal**

• **Demo** (in file `scripts/misc/miscScripts`)

Command that explains how to run the demos of the different numerical packages

• **DemoSignalDisp** (in file `scripts/signal/DemoSignal`)

Demo that displays different signals and teaches you what you can do with the mouse

Chapter 5

Package `terminal` 2.0

Package that regroups all the bindings for managing terminal history, help and file completion systems. It just adds bindings to LastWave. It almost does not include any script commands that can be directly callable (most of the names of the script commands that are defined in this package start with a '_'). You can change the script file 'scripts/terminal/keys' to redefine the keys associated to the bindings.
*** Authors and Copyright : E. Bacry*

5.1 Script Commands

• **Help** (in file `scripts/terminal/help`)
Displays some basic help about the help system.

• **HelpTerm** (in file `scripts/terminal/keys`)
Command that tells you what the binding keys are for command line editing capabilities

• **apropos** (in file `scripts/terminal/help`) `*word1* [*word2* ... *wordN*]`
Search for all the commands whose help contain the strings *word<i>* (any order). For each package, it prints each of the so-obtained command with the corresponding part of the help around the occurence of *word1*

• **help** (in file `scripts/terminal/help`) `(*commandName* [*action*] | *type* [*fieldName*])`
Displays help on a command or a &type. If *commandName* is not a valid command name then it prints a list of commands which name begins with *commandName*. if a *type* is specified (i.e., a string starting with '&'), it displays the doc on the corresponding type and on all the fields. If *field*

is specified then only the corresponding field is displayed. If *fieldName* is not a valid field name then it prints a list of fields which name begins with *fieldName*.

• **helpp** (in file `scripts/terminal/help`) `[*packageName*]`
Displays help on a package and all the commands available. If *packageName is not a valid name, it prints the names of available packages that start with *packageName*. If no *packageName* is specified it gives a list of all the available packages.

• **helpv** (in file `scripts/terminal/help`) <val> `[*fieldname*]`
Displays help the fields of a value. If *fieldName* is not a valid field name then it prints a list of fields which name begins with *fieldName*. The value should not be a number (numbers do not have any field!).

Index