SIC

a gildas working group software

20-oct-2015

sic
sympathetic interpretor of commands
sympathique interpréteur de commandes
language SIC version 9.3
language GUI version 1.2
language VECTOR version 4.0
language TASK version 3.0
language ADJUST version 2.0

The GILDAS working group is a collaborative project of the Observatoire de Grenoble (1,3) and IRAM (2), and comprises: F. Badia[2], D. Broguière[2], G. Buisson[1], L. Desbats[1], G. Duvert[1], T. Forveille[1], R.Gras[3], S. Guilloteau[1,2], R. Lucas[1,2], R. Neri[2] and P. Valiron[1].

(1) Laboratoire d'Astrophysique
Observatoire de Grenoble
BP 53 X
414 Rue de la Piscine
F-38402 Saint Martin d'Hères CEDEX

(2) Institut de Radio Astronomie Millimétrique
300 Rue de la Piscine
F-38406 Saint Martin d'Hères

(3) CEPHAG
Observatoire de Grenoble
F-38402 Saint Martin d'Hères CEDEX

Contributions from and invaluable discussions with J. Cernicharo, P. Begou[3], S. Delahaye[3], A. Dutrey[1,2], C. Kahane[1], P. Monger, J.L. Monin[1] and all GILDAS users are gratefully acknowledged.

Contents

1 Introduction

SIC (*) is a command line interpretor, written in FORTRAN and callable as a subroutine by any program. It provides a command language with the following major features:

- resolution of command abbreviations
- definition of symbols
- procedures with arguments substitution during execution
- log file
- multi-language structure
- loop buffers for repetitive actions
- variables, arithmetic and logical expressions evaluation
- structured logical tests
- error recovery
- stack buffer
- keypad editing of command lines
- Graphic User Interface (widgets) using simple scripts

This manual contains several chapters. Chapter 2 (The SIC monitor) should be read by every user before using SIC. Chapter 3 contains a copy of the internal help files of SIC. Chapter 4 is a list of all possible SIC error messages and of their most usual recovery procedures. This reference may be useful if you encounter an error message which you do not understand while running SIC.

2 The SIC Monitor

2.1 Basic Features

2.1.1 Syntax

All commands parsed by SIC must have the following syntax:
`[LANG\]COMM [ARG1 [ARG2 [...]]] [/OPT1 [ARG11 [...]] [/OPT2 [...]]`

Where `LANG` is the language name, `COMM` the command name, `OPT1` and `OPT2` are option names, `ARGs` are the arguments of command and options, and brackets indicate optional fields.

The language, command and option names can be abbreviated and SIC checks for ambiguities. Arguments are separated by any number of separators (Spaces or Tabs). An option is a word beginning with a Slash. The options, like the commands, may have arguments. The syntax analyzer converts to upper case letters, strips useless separators, translate the symbols (See 2.1.7, expands the command and option names by looking through its current "dictionary".

The language field is optional. If present, it restricts the resolution of command name abbreviations to all languages of which it is an abbreviation. Otherwise, all languages are searched for.

The first action of the syntax analyzer is to suppress redundant separators. This is of course not always wanted (e.g. a figure caption) and SIC offers one way to circumvent this problem: the so-called strings, which are arguments included between double quotes (`"`). Strings are not modified by the syntax analyzer, but the outer double quotes will be ignored when the argument is used. However, strings are not protected against symbol substitution.

The program calling SIC may expect arguments of various types: character strings, real numbers, integers or logicals. For a specified type, e.g. a real number, the argument can be either a constant value (e.g. 3.14159), a variable (e.g. PI), or an expression (e.g. 2*ASIN(-1.0)). Conversion to the specified type is done automatically if possible at all (See command `SIC PRECISION` in 2.6.4).

A "-" sign as the last significant character in a command line indicates that a continuation follows on the next line, e.g.

```
SIC> HELP -
SIC> EDIT
```

is interpreted as

```
SIC> HELP EDIT
```

No special procedure is used to cut strings for continuation lines. It is simply done according to the following example

```
SIC> LABEL "A very long-
SIC>  string must be cut"
```

which is interpreted as

```
SIC> LABEL "A very long string must be cut"
```

Comments may appear at the end of any line. The comment area starts with a "!" sign; all the following text is ignored. Comments can be used in conjunction with the continuation mark :

```
SIC> DRAW RELOCATE 13-       !This is an example of comments
SIC> .45 15.00 /USER         ! and continuation
```

is interpreted as

```
SIC> DRAW RELOCATE 13.45 15.00 /USER
```

BUT do not try this one

```
SIC> LABEL "A lon-    ! A long string
SIC> g caption"       ! with continuation mark
```

which produces the message

```
E-INTER,  Unbalanced quote count
```

immediately after the first line as been typed in, because the syntax analyzer was considering the - and ! signs as part of a string. Comments may be convenient to self-document complex procedures.

2.1.2 The Prompt

The prompt is defined by the calling program. In addition, SIC modifies the prompt aspect according to the execution level. In this example the calling program is assumed to pass the string `'GAG'` to SIC. Then the following prompts may appear :

```
GAG>     ! Lowest execution level
GAG_3>   ! Third execution level
GAG_5:   ! Compile mode for the Loop buffer at level 5
```

2.1.3 The on-line HELP

The `HELP` utility of SIC is structured at three different levels. Without an argument, `HELP` gives the list of available help. Since SIC is a multi-language system, `HELP Language_Name\` prints one line description of all commands of the named language. For example, `HELP SIC\` gives a one line description of all the SIC monitor commands. `HELP Command_Name` prints more detailed information on that command. For some command, the help text is divided in subtopics which can be accessed by typing `HELP Command_Name Subtopic`.

If you consult a long `HELP` text, you may use the `PAGE` mode. You will then be prompted for continuation when the screen is full. The page mode is usually set by default on "intelligent" terminals, and you can switch between `PAGE` and `SCROLL` mode for the help using command

```
SIC> SIC HELP PAGE     or     SIC HELP SCROLL
```

On-line access to the documentation can be obtained by

```
SIC> SIC HELP CONTENT  or     SIC HELP INDEX
```

In `CONTENT` mode, `HELP Command` displays the documentation page indicated for the specified `Command` by the table of content, while in `INDEX` mode, the first page indicated in the index is displayed.

2.1.4 The Stack

The Stack is an internal buffer where commands are automatically placed. It may be considered as a real time logfile, from which you can retrieve commands. The stack buffer is organized as a circulating buffer, with a `"first in - first out"` replacement procedure when the buffer is full, or when the maximum number of commands is reached. The command number always increase, even when the buffer fills up.

The command

```
RECALL [Arg]
```

recalls command from the stack for execution. Depending on whether line editing mode is possible (see keypad line editing), the recalled command may be edited prior to submission, or not. If no argument is present, the last command is recalled. If the argument is a number N, the N-th command of the stack is recalled. If the argument is a string, the stack is scanned backwards to find a command beginning by this string.

If line editing is possible, commands may also be retrieved from the stack using the Up and Down arrows on the terminal keyboard **a documenter plus correctement**.

The command

```
EDIT
```

without arguments will dump the stack on a file named `stack.DefExt` (where `DefExt` is the current default procedure extension), and calls the default editor to edit this file. It can then be executed as any other procedure.

2.1.5 Line Editing Facility

Line editing is normally available to edit command lines prior to submission. The following control keys can be used

```
<^A>            Move to beginning of line
<^B>            Backspace one character (BACKWARD)
<^F>            Advance one character (FORWARD)
<^H>            Delete previous character (BACKSPACE)
<^J>            Delete to beginning of word, or previous word
<^M>            Submit command line (RETURN key)
<^N>            Recover Next command
<^P>            Recover Previous command
<^U>            Delete to beginning of line
<DEL>           Delete previous character
```

These commands are identical to the standard line editing in the Emacs editor.

SG: Is it still valid? If you are using an ANSII compatible terminal (VT100 series and upwards), the numeric keypad is also available to perform additional actions

```
<PF4>           Delete to end of line
<->             Delete to end of word, or next word
<LF>            Delete to beginning of word, or previous word
<,>             Delete character
<DEL>           Delete previous character
```

```
<1>             Move to next word
<2>             Move to end of line
<4>             Set advance mode (for WORD moves)
<5>             Set backup mode (for WORD moves)
<ENTER>         Submit the command line
<RETURN>        Same as above
<Up Arrow>      Retrieve previous command of Stack
<Down Arrow>    Retrieve next command of Stack
<Left Arrow>    Backward one character
<Right Arrow>   Advance one character
```

For lines longer than the screen width, the display uses a window and automatically centers it on the current character when the cursor position reaches one end.

SG: Is that still true? The keypad editing mode can be turned off by the `SIC EDIT OFF` command and turned on by the `SIC EDIT ON` command. Line editing mode affects the error recovery system and the RECALL command.

2.1.6 The Log File

The *Log File* is a post-mortem listing of all commands issued and successfully executed during a working session. It is kept on leaving SIC (by using `EXIT`). Some commands like `HELP` are not put into the *Log File*. Most programs using SIC put the *Log File* in the `GAG_LOG:` area (usually your default directory, but see "Customizing") to avoid multiplication of files in a directory tree, and purge it upon exit keeping the last two or three versions. *Log Files* can be used as the basis of subsequent procedures. If you are short of disk space, purge and delete yourself the log files.

2.1.7 Symbols

SIC allows the user to define symbols, which are abbreviations of any character string. Any command line is first parsed for symbols in the symbol table. The command interpretor assumes that the following entities may be symbols :

- the first word of command line (e.g. `AA` in command `AA /OPTION`)
- any string (without spaces) included between single quotes (e.g. `'AA'`). This syntax may also indicate a character variable (See "Character Variables and Implicit Formatting").

There is no recursive analysis of the line for the symbol translation. Note that, contrary to character variables, the symbol translation occurs even within strings, and that case does not matter.

Symbols are defined by issuing the following command

```
SIC> SYMBOL TOTO "Whatever you want"
```

A symbol definition may refer to an other already defined symbol. Whenever `'TOTO'` is found in a command line, it will be replaced by the string Whatever you want e.g.

```
SIC> LABEL "Units 'TOTO'"
```

will be interpreted as

```
SIC> LABEL "Units Whatever you want"
```

The Symbol Table can be listed using command `SYMBOL` without arguments. If `SYMBOL` has a symbol name for argument, the translation of this symbol is given. `DELETE /SYMBOL TOTO` will delete symbol `TOTO` from the current symbol table. Note that symbols should only include alphanumeric characters.

2.2 Variables and Expressions

Most of the power of SIC comes from its ability to handle variables and perform operations (arithmetic or logical) on them. When used in combination with the GREG program for display, SIC variables can be used to performed efficient data analysis.

2.2.1 Definitions and Assignments

SIC supports variables and arithmetic or logical expressions evaluation. Variables can be defined either by the program or by the user. Program-defined variables may have the Read-Only attribute which prevents them from being overwritten by the user (see SIC PROGRAMMING MANUAL). Variable names are up to 64 characters long, case insensitive, and must begin with a letter. A variable can be LOCAL or GLOBAL. GLOBAL variables are valid at any execution level in SIC. On the opposite, LOCAL variables are valid only in the procedure where they have been declared, or in any loop started within this procedure, or in any interactive level generated from this procedure (by a `PAUSE` or an error). Variables declared by a program are always GLOBAL.

Arithmetic and logical expressions are automatically evaluated when used as arguments to commands. The evaluation is done in single or double precision arithmetic, according to command `SIC PRECISION`. Parentheses are allowed, but there is a limit on the complexity of arithmetic and logical expressions. A local variable has precedence over a global variable of the same name. Local variables are deleted when the creating procedure terminates.

Variables may be defined using command `DEFINE`, and assigned values using command `LET`. SIC is a declarative language in which all variables must be defined before being used. For convenience, command `LET` has an option `/NEW` which allows declaration of the assigned variable. Five type of variables are allowed : `REAL, INTEGER, LOGICAL, DOUBLE` (for double precision real variables), and `CHARACTER`.

Variable values may be typed using the `EXAMINE` command, which also indicates whether the variable is GLOBAL or LOCAL, and in the latter case, the corresponding procedure level.

2.2.2 Functions and Operators

For arithmetic expressions, the known operators are

```
-          Subtraction
+          Addition
*          Multiplication
/ or |     Division (the | sign has been added because the /
                   is the option separator)
** or ^    Exponentiation
```

Known single argument functions are

```
ABS       Absolute value
ACOS      Arc Cosine
ASIN      Arc Sinus
ATAN      Arc Tangent
COS       Cosine
COSH      Hyperbolic Cosine
EXP       Exponential
INT       Integer Part
LOG       Natural logarithm
LOG10     Decimal Logarithm
NINT      Nearest Integer
FLOOR     Integer floor
CEILING   Integer ceiling
SIN       Sinus
SINH      Hyperbolic Sinus
SQRT      Square Root
TAN       Tangent
TANH      Hyperbolic Tangent
```

Known two arguments functions are

```
ATAN2   Arc tangent with two arguments
MAX     Maximum of two values
MIN     Minimum of two values
MOD     Modulo (true modulo, even for negative numbers)
SIGN    Sign transfer
```

For logical expressions, the operators are

```
.OR.      .AND.     .NOT.
.GT.      .GE.      .LT.      .LE.
.NE.      .EQ.
```

Note that there must be no blanks in the logical expressions.

The known functions are

```
EXIST   Returns .TRUE. if its argument is a defined variable
FILE    Returns .TRUE. if its argument is an existing file.
```

Additional arithmetic functions may be declared by the calling programs. Two special functions are always declared :

```
NOISE(x)     Gaussian Noise of Sigma X
RANDOM(x)    Random Number between 0 and X
```

These additional functions, and their number of arguments, can be listed using command `EXAMINE /FUNCTION`.

2.2.3 Vector Operations

SIC supports *array variables* of up to 7 dimensions. Array dimensions are specified between brackets (not parentheses) with the comma as separators. `CHARACTER` and `LOGICAL` variables can also be arrays. Arithmetic operations always work on arrays on an element by element basis. Scalar variables are considered like arrays of any size. For example,

```
DEFINE REAL A[5] B[5]
LET A = ABS(B)
LET B = 1.0
```

assigns to each element of A the absolute value of the corresponding element of B, and then assigns to each element of B the value 1.0. Dimensions of arrays must match in arithmetic expressions.

Any subset of an array variable can be referenced in an expression, for example

```
DEFINE REAL A[5] B[5,10] C[5,10,3]
LET B[,6] = A
LET C[,10,1] = B[,3]
```

Dimensions of the sub-arrays must also match. Transposition is now supported : e.g. in the precedent example, C[1,,1] could have replaced C[,10,1]. The indices can be previously defined scalar variables. Leading commas may be omitted.

```
DEFINE REAL A[5] B[5,10] C[5,10,3]
LET B[,6] = A
LET C[,10,1] = B[,3]
```

It is also possible to specify a range of indexes rather than a single index.

```
DEFINE REAL A[5] B[3,10] C[4,4,4]
LET B[,6] = A[2:4]
LET A[:3] = B[,8]
LET A[2:5] = C[3,,4]
```

2.2.4 Implicit Loops

It is sometimes convenient to assign to an array values which are functions of the array indices. This can be done using "implicit loops", such as

```
DEFINE REAL A[10,10]
LET A[I,J] = EXP(-((I-5)|2)**2-((J-6)|3)**2)
```

in which I and J have NOT been declared as known variables. I and J are known as "implicit variables". The preceding expression is equivalent to the following commands

```
DEFINE REAL A[10,10]
FOR J 1 TO 10
FOR I 1 TO 10
LET A[I,J] = EXP(-((I-5)|2)**2-((J-6)|3)**2)
NEXT
NEXT
```

but it executes thousands of times faster... Mixing of implicit and declared (or loop) variables is strictly forbidden at present. It can usually be avoided by using intermediate arrays.

2.2.5 Conditional Assignment

Another convenient function is to assign to an array new values only in for some parts of the array, based on a logical mask or logical expression. The option `/WHERE` of command `LET` allows such operations. For example

```
DEFINE REAL A[10,10]
LET A[I,J] = EXP(-((I-5)|2)**2-((J-6)|3)**2) /WHERE (I+J).LT.10
```

will set only a part of the A array (note that I and J are "implicit variables").

The preceding expression is equivalent to the following commands

```
DEFINE REAL A[10,10]
FOR J 1 TO 10
FOR I 1 TO 10
  IF (I+J).LT.10 THEN
    LET A[I,J] = EXP(-((I-5)|2)**2-((J-6)|3)**2)
  ENDIF
NEXT
NEXT
```

but it executes thousands of times faster...

Conditional assignment can be mixed with implicit loops, as shown above, but there are some syntax restrictions: please refer to the internal help for more details.

2.2.6 Size casting

Typing the explicit dimensions when declaring a new variable may be tedious. SIC allows to create arrays of dimensions identical to those of existing variables through the `/LIKE` option. For example, if A is an existing integer variable of dimensions 10,5,30 the following command

```
DEFINE REAL B C D /LIKE A
```

is equivalent to

```
DEFINE REAL B[10,5,30] C[10,5,30] D[10,5,30]
```

and defines three real arrays of dimension [10,5,30]. This feature, called size casting, is specially convenient to declare arrays that match images (see below).

2.2.7 GILDAS Images

Variables are normally allocated using virtual memory, and hence are lost once the program terminates. It is possible to allocate variables as disk files, called **Images**, which are mapped into the virtual memory of the program, using command

```
DEFINE IMAGE Variable File Keyword
```

where

`Variable` is the desired variable name. This variable name should be at most 3 characters to hold the header variables.

`File` is the name of the file holding the image. Default extension is `.gdf` (GILDAS Data Format), but any extension is valid.

`Keyword` indicates how the file must be used and may be

`READ` use an existing image as Read-Only variable

`WRITE` use an existing image as Read-Write variable. The user must have Write access to the file.

`EXTEND` Allows to extend the last dimension of an existing image. The user must have write access to it.

`REAL` create a new image of type `REAL`

`DOUBLE` create a new image of type `DOUBLE`

`INTEGER` create a new image of type `INTEGER`.

For already existing images (`READ, WRITE`), the type and size of variable are determined by the program. For new images, the size must be specified in the variable name, and the variable is always declared as Read-Write. To `EXTEND` image, the new value of the last dimension must be specified in the variable name. Header variables (see next section) are defined with the same status (Read or Write) as the image.

Images (i.e. files in the GILDAS data format) are used by most programs distributed by the Groupe d'Astrophysique. See the GILDAS (Grenoble Image and Line Data Analysis Software) documentation. **Tables** usually refer to images with 2 dimensions, but more generally to images which do not define a coordinate system.

The `DEFINE IMAGE` command has two variants, `DEFINE TABLE` and `DEFINE HEADER`. `DEFINE TABLE` only defines a single variable pointing to the image content. On the opposite `DEFINE HEADER` defines only the header variables, as described in next section.

2.2.8 GILDAS Headers

The `DEFINE IMAGE` command allows SIC to access not only to the content of an image (the data value), but also to all its associated parameters; `DEFINE HEADER` allows access only to these associated parameters. The header variables have names derived from the generic name by adding the special character % and an extension (such as e.g. `NDIM` for the number of dimensions) to the generic header name. For example, command `DEFINE HEADER VAR file.gdf READ` also creates the following variables:

```
VAR%GENE       Integer        Length of general section
VAR%NDIM       Integer        Number of dimensions (ReadOnly)
VAR%DIM        Integer[7]     Dimensions           (ReadOnly)
VAR%CONVERT    Double[3,7]    Conversion formulae for the 7 axes:
                              Reference pixel,
                              Value at reference pixel,
                              Increment
VAR%BLAN       Integer        Length of blanking section
VAR%BLANK      Real[2]        Blanking and tolerance
VAR%EXTREMA    Integer        Length of extrema section
VAR%MAX        Real           Maximum
VAR%MIN        Real           Minimum
```

VAR%MINLOC	Integer[7]	Position of min value
VAR%MAXLOC	Integer[7]	Position of max value
VAR%DESC	Integer	Length of units and system section
VAR%UNIT	Char*12	Image unit
VAR%UNIT1	Char*12	First axis type
VAR%UNIT2	Char*12	Second axis type
VAR%UNIT3	Char*12	Third axis type
VAR%UNIT4	Char*12	Fourth axis type
VAR%SYSTEM	Char*12	Coordinate system
VAR%POSI	Integer	Length of position section
VAR%SOURCE	Char*12	Source name
VAR%RA	Double	Right Ascension
VAR%DEC	Double	Declination
VAR%LII	Double	Galactic longitude
VAR%BII	Double	Galactic latitude
VAR%EQUINOX	Real	Equinox of coordinates
VAR%PROJ	Integer	Length of projection section
VAR%PTYPE	Integer	Projection type (code)
VAR%A0	Double	first coordinate of projection center
VAR%D0	Double	second coordinate of projection center
VAR%ANGLE	Double	position angle of projection
VAR%X_AXIS	Integer	First projected axis
VAR%Y_AXIS	Integer	Second projected axis
VAR%SPEC	Integer	Length of spectroscopy section
VAR%LINE	Char*12	Line name
VAR%FREQRES	Double	Frequency resolution
VAR%IMAGFRE	Double	Image Frequency
VAR%RESTFRE	Double	Rest Frequency
VAR%VELRES	Real	Velocity resolution
VAR%VELOFF	Real	Velocity offset
VAR%F_AXIS	Integer	Frequency/Velocity axis
VAR%BEAM	Integer	Length of beam section
VAR%MAJOR	Real	Major axis of beam
VAR%MINOR	Real	Minor axis of beam
VAR%PA	Real	Position angle of beam
VAR%SIGMA	Integer	Noise section length
VAR%NOISE	Real	Theoretical noise
VAR%RMS	Real	Actual noise
VAR%PROPER	Integer	Proper motion section length
VAR%MU	Real[2]	along RA and DEC, in mas/yr
VAR%PARALLAX	Real	Parallax in mas

In addition, the following variables are also created when accessing GILDAS **UV Tables** through the `DEFINE UVTABLE` command:

VAR%NCHAN	Integer	Number of channels
VAR%NVISI	Integer	Number of visibilities
VAR%NSTOKES	Integer	Number of Stokes parameters
VAR%NATOM	Integer	Complex visibility size

```
VAR%BASEMIN    Real           Minimum baseline
VAR%BASEMAX    Real           Maximum baseline
```

VAR becomes a dummy variable of type header, which can only be referenced in a further DELETE /VARIABLE command. The VAR%item variables are ReadOnly or ReadWrite according to the keyword following the filename, except for the dimension variables (VAR%DIM and VAR%NDIM), which cannot be modified. VAR%READONLY indicates whether the variable can be modified or not.

Full headers can be copied to one another, using the command

```
LET A% = B%
```

which copies the header of image B into that of image A (dimensions are not modified, however). Thus a full copy of a 4 dimensions GILDAS data file can be obtained within SIC as follows:

```
DEFINE IMAGE A Oldfile.gdf READ
DEFINE INTEGER N1 N2 N3 N4
LET N1 A%DIM[1]
LET N2 A%DIM[2]
LET N3 A%DIM[3]
LET N4 A%DIM[4]
DEFINE IMAGE B[N1,N2,N3,N4] Newfile.gdf REAL
LET B A      !  Copy A data into B
LET B% A%    ! Copy A header into B header
DELETE /VARIABLE B ! Deletes the SIC variables, but not the file...
```

A simpler (and more generic) way to declare new images is to use the size casting provided by the /LIKE option:

```
DEFINE IMAGE A Oldfile.gdf READ
DEFINE IMAGE B Newfile.gdf REAL /LIKE A  ! Define B like A...
LET B A      !  Copy A data into B
LET B% A%    !  Copy A header into B header
DELETE /VARIABLE B ! Deletes the SIC variables, but not the file...
```

2.2.9 Structures

SIC variables can also be structures, which are related ensembles of variables similar to the derived types in Fortran-90. The naming conventions follows the Fortran-90, with structure elements separated by the % sign.

A structure is defined by

```
DEFINE STRUCTURE A
```

and any element then by

```
DEFINE REAL A%B
```

Sub-structures can be defined, too

```
DEFINE STRUCTURE A%C
DEFINE REAL A%C%E
```

An entire structure (e.g. A as above) and all its element can be deleted using

```
DELETE /VAR A
```

Structures can be assigned as a whole by command LET. This command will copy all elements of the same name, and leave the others untouched. HEADERs are actually special structures.

2.2.10 Character Variables and Implicit Formatting

In contrast with the Symbols, which are substituted in the command line before the parsing, variables and expressions are evaluated after the command line analysis. In general, a real (resp. integer and logical) argument is considered as a mathematical (logical) expression and evaluated when read by the program calling SIC. The command line stored in the stack and logfile contains the mathematic expression, not the current value.

The behaviour for Character variables is slightly different, in the sense that only items included between single quotes are considered as possible character variables, if they have not yet been expanded as known symbols of course. Using character variables in logical expressions is an exception to this rule because translation should be avoided in this case, see next chapter. Contrary to Symbols, Character variables translation does not occur in strings.

Not only Character variables but also any mathematical and logical expression may be included between quotes. Mathematical expressions are evaluated and formatted using the shortest possible format. Logical expressions are evaluated as YES or NO. The formatted command string is substituted to the expression and quotes, and used in the string returned as character argument to a command. This feature is known as "Implicit Formatting". In this way, non-character variables and expressions can be used where a character argument is required. The reverse is not true however: Implicit Formatting should not be used if a non-character argument is expected.

Any variable can be typed using the `EXAMINE` command which will display the variable name followed by its current value. More than one variable may be displayed at the same time using the `SAY` command.

Concatenation of variables is easily obtained by mixing explicit strings (between double-quotes) and implicitely formatted variables. For example, is A is character variable of content "I am", the following command

```
LET B "You know "'A'" happy"
```

attributes to B the content "You know I am happy".

2.2.11 Initializing variables from external files

SIC allows easy initialization of variable from files in "foreign" format (i.e. not GILDAS images). This can be done with the `ACCEPT` command, which allows reading variables from formatted or unformatted files. This command is available in 3 major modes

- `ACCEPT /ARRAY`
  `ACCEPT Var /ARRAY File [/FORMAT String] [/LINE Begin [End]]`
  reads in free format or user specified format variable `Var` from file `File`, selecting a line range if specified. `Var` may have up to 4 dimensions.

- `ACCEPT /BINARY`
  `ACCEPT Var /BINARY File [Skip]`
  reads the binary file `File` to set variable `Var`. `Skip` is a number of BYTEs to skip before reading. Note that there is no type conversion in this command: the binary content of the file must match the type declared for the variable.

- `ACCEPT /COLUMN`
  `ACCEPT V1 [V2 [...]] /COLUMN File [/FORMAT String] [/LINE Begin [End]]`
  reads the formatted file `File` to set one or several 1-dimensional variables `V1 V2 ...`. Free

format is used by default, unless the /FORMAT option is specified. The special argument * can be used instead of the variable name to indicate a "dummy" variable, which is read from the file, but not assigned. This allows to skip a column in the input file.

2.3 SIC as a programming language

The second most important power of the SIC command language is its programming features. SIC supports command procedures, loops and conditional execution of statements in procedures.

2.3.1 Procedures (or Command Files)

SIC has command procedure capabilities. A procedure is an external file containing valid commands. The SIC variables PRO%NARG (number of arguments) and PRO%ARG[:] (arguments as an array of character strings) can be accessed from within the procedure; they describe the arguments passed to the procedure when it was invoked. The tokens &1, &2, ..., &8 can also be used in the body of the procedure, its arguments will be substituted for these tokens. Substitution occurs also within the character strings. A procedure is executed by

```
SIC> @ Procedure_Name [P1 ... PN]
```

Commands are read from the file Procedure_Name (with a default extension depending on the calling program or the user) and executed. P1 is a parameter string to be evaluated from PRO%ARG[1] or substituted to the token &1 during execution. Up to 32 parameters may be passed to the procedure. The commands are echoed to the user's terminal if the VERIFY switch is set ON. Most programs using SIC define a default procedure extension equal to the program name (such as .greg .class, etc...). If not specified, the default macro extension is .pro. It can be listed and changed using command SIC EXTENSION.

Procedures (or any text file indeed) can be typed from within SIC using the command TYPE.

```
SIC> TYPE Procedure_Name
```

If no argument is given to TYPE, the stack buffer is listed.

2.3.2 Loops

```
FOR Variable List
```

opens a FOR-NEXT loop to be executed for values given in the list.

This command may have the following format :

```
FOR I n1 n2 n3 to n4 by n5 n6 to n7
```

where I is the loop variable name. Loop variables must not be previously defined, and are undefined when the loop execution is finished. The prompt changes to 'SIC_n: ', where n is the current SIC execution level, and all subsequent commands until NEXT are the body of the loop. The loop variable can be used as any other SIC variable, e.g. in arithmetic expressions such as EXP(-(I+3.5)**2). In addition, it can be used in a formatted way when it is included between quotes, e.g. in NAME.EXT.'I'. In this case, the substitution occurs also within the character strings (see "Implicit Formatting"). The commands are echoed to the user's terminal if the VERIFY switch is set ON.

Up to 127 loop, procedure or IF block levels can be nested, and there is no restriction upon the loop and procedure nesting, e.g.

```
SIC> FOR I List1
SIC> FOR J List2
SIC> ..
SIC> NEXT
SIC> FOR J List3
SIC> ...
SIC> NEXT
SIC> NEXT
```

is perfectly valid.

```
FOR /WHILE Logical_Expression
```

This is another possible syntax for `FOR-NEXT` loops. The loop is executed conditionally provided "Logical_Expression" is `TRUE`. "Logical_Expression" must be any valid logical expression, possibly including arithmetic sub-expressions in it.

2.3.3 Structured Programming and Logical Expressions

SIC includes structured logical tests of the form (if-blocks):

```
IF Logical_Expression [THEN]
  ...
ELSE IF Logical_Expression [THEN]
  ...
ELSE
  ...
ENDIF
```

or of the form (if-statements):

```
IF Logical_Expression One_Command_Line
```

The if-block form is similar to FORTRAN with two differences. First, the `THEN` keyword is optional. Second, `ELSE` and `IF` must be separated by at least one space or tab in the `ELSE IF` command. In addition, provided the restriction on the total number of loops, procedures and `IF` blocks is met, any nesting between loops, procedures and `IF` blocks is allowed.

Variables can appear in the logical expressions, and this is one of the most frequent use for variables. An `IF` block must be complete in a procedure or loop, otherwise an error occurs.

Logical expressions may include operations on arithmetic, logical or character variables. In logical expressions, strings (i.e. text included between double quotes) are recognized as character constants. Character variables should not be included between single quotes, since their current values would be substituted by SIC before logical expression analysis. Arithmetic sub-expressions are allowed.

Assuming `GOOD` is a character variable whose current value is `"Let it be"`, and `PI = 3.1415926535897932` (Double precision), examples of valid logical expressions are :

```
L = ("I am happy".EQ.GOOD) (.FALSE.)

L = ("Let it be".EQ.GOOD) (.TRUE.)
```

```
    L = ("I am happy".EQ."'GOOD'") (.FALSE.)
evaluated literally since GOOD is not substituted in a string

    L = PI.EQ.ACOS(-1.0) (.TRUE.)

    L = 'PI'.EQ.2*ASIN(1.0) (.FALSE.)
evaluated as 3.141592653589793.EQ.2*ASIN(1.0)
because of implicit formatting, one digit being lost in the
formatting because of binary to decimal conversion.

    L = ("I am happy".NE.GOOD).OR.(PI.EQ.ACOS(-1.0))
```

But the following expressions are invalid :

```
    L = (PI.EQ.GOOD) Variable type mismatch.

    L = ("I am happy".NE.'GOOD')
Because it is evaluated as ("I am happy".NE.Let it be).
```

2.3.4 Execution Level

Procedures and Loops can be nested. Hence, SIC may operate at different *Levels of Execution*. Commands are provided to activate some level (`@, FOR`), suspend (`PAUSE`), resume (`CONTINUE`) or abort its execution (`NEXT, QUIT, BREAK, RETURN`). Errors occuring within a non-interactive execution level generate a `PAUSE`, which returns interactive control to the user at a level immediately higher. The prompt at level I changes to `'SIC_I> '` to remind the user what SIC is doing.

It is also possible to interrupt a sequence of commands (procedure or loop) by pressing `<^C>` at any time. The current command is then normally completed (unless it traps the `<^C>` by itself), but a `PAUSE` is generated when the command terminates.

Related commands :

BREAK
: Aborts Loop execution without generating an error. The loop is considered to have completed successfully, and execution resumes at the command line following the NEXT command of the loop.

CONTINUE
: resumes Procedure or Loop execution after a `PAUSE`, either explicit or generated because of an error condition. *C is always a non ambiguous abbreviation of* `CONTINUE`, unless you redefine it as a Symbol.

EXIT
: exit from the program.

NEXT
: The effect of this command depends on the context:

  - If encountered while entering loop commands (during loop compilation), it ends the loop definition and activates its execution.

- If encountered during loop execution, all commands left in the loop are skipped and loop execution starts again for next index value. This typically occurs when it is specified as error handling (`ON ERROR NEXT`), or typed interactively after a `PAUSE` has occured.

PAUSE
: sets a break point in the Loop or a Procedure. `PAUSE` returns control to the user when executed in any of the non-interactive modes (Loop and Procedures). Any valid command can be executed while in interrupt mode. The normal execution of the interrupted level can be resumed by typing `CONTINUE`. `NEXT` and `QUIT` may also be valid continuation sequences. The PAUSE command can be followed by a character string argument which is printed before the PAUSE becomes effective.

QUIT
: If `QUIT` is typed after a `PAUSE` occured in a Procedure or Loop, the execution of the interrupted procedure is aborted, and one returns to the previous level of execution. In this case, an error condition is transmitted to the previous level to allow the user to take the appropriate decision.

RETURN
: Terminates procedure execution, and returns to the previous level of execution. An implicit `RETURN` is always executed at the end of the command file. Command `RETURN BASE` returns to the normal interactive level (level 0). Command `RETURN ERROR` returns to the previous level of execution, but also transmit an error status to this level.

2.3.5 Error Recovery

SIC has a powerful error recovery system. Every command returns to the SIC monitor a status to indicate if any error occured. If so, by default SIC attempts to make a `PAUSE`. In an interactive session, there is a (presumably intelligent) user to decide what to do, and who can hopefully correct the error (a typing mistake for example) and then type `CONTINUE` to proceed. In a non-interactive session (Batch or command procedure), no such intelligent decision is possible and the `PAUSE` causes an Abort of the program.

If line edition is possible, the command which caused the error is automatically displayed for correction by the user. The corrected line will be submitted whenever `<RETURN>` or `<ENTER>` keys are pressed.

It is possible to override this default behaviour by command

```
ON ERROR [Other command]
```

After this command has been issued, any error will attempt to execute the `"Other command"`. If this command happens to fail, SIC will try to make a `PAUSE`. This command can be any command of the program, including `@, CONTINUE, EXIT, NEXT`. They will behave exactly has usual, except command `QUIT`.

In interactive mode, the `QUIT` command is usually typed to abort an erroneous procedure. In this case, it decreases the execution level by 2, and transmit an error to this new level to signal an abnormal end of some procedure. In error recovery mode, `QUIT` directly transmits the error to the previous level. It is in fact translated into `RETURN ERROR`.

Judicious use of the `ON ERROR` command may enable you to make batch jobs very conveniently. In particular, think of the behaviour of `ON ERROR NEXT` and `ON ERROR BREAK` when using loops, and `ON ERROR RETURN` when using procedures.

The `ON ERROR` command is a *local* command: that means it is only valid within the procedure which declared it (and loops executed within this procedure). However, if a `PAUSE` occured from a procedure (or loop), an interactive execution of the `ON ERROR` command will reset the error processing behaviour of the interrupted procedure.

2.4 The GUI ("Graphics-User-Interface") Mode

SIC provides facilities (called GUI mode) to create documented input windows ("widgets") by which the user can modify variables and execute pre-defined commands.

SIC can create 3 types of widgets:

- Detached menus
  which are menu bars created by the `GUI\PANEL /DETACH` command. These menus have no associated parameters, but run in parallel with the main program. Pre-defined commands are activated by pressing the various buttons.

- Main panel window
  which is created by the `GUI\PANEL` command. The user can modify variables using widgets, and execute one or several actions by pressing the appropriate buttons. The variables are actually modified only when a button is pressed.

- Sub-panels
  which are created by the `GUI\BUTTON` command. Such windows are associated with a specific command, and are hidden by default. They are typically used to hold variables which are seldom modified by the user. The variables defined in these windows and in the main window are modified when the `GO` button is pressed.

User input with this system is fairly intuitive. Help is available by clicking on the help button(s) or on the prompt area for each variable.

The following description rather concerns advanced users who want to create their own window interfaces.

2.4.1 Detached menus

Detached menus are created using the `GUI\PANEL /DETACH` command. Once created, buttons and pulldown menus can be defined within the detached menu using respectively the `GUI\BUTTON` and `GUI\MENU` commands respectively. There are no associated variables to the detached menus. Help is available through a Help button. The menu is mapped when the `GUI\GO` command is typed.

The following procedure illustrates how to create a detach menu; it creates a menu bar with 3 pulldown menus and a help button.

```
!
GUI\PANEL "GRAPHIC X-Window interface" PR:GRAPHIC_SIC.HLP /DETACH
!
GUI\MENU  "SIC"
GUI\BUTTON "SIC\PAUSE" PAUSE
```

```
GUI\BUTTON "SIC\CONTINUE" CONTINUE
GUI\BUTTON "SIC\QUIT" QUIT
GUI\BUTTON "SIC\NEXT" NEXT
GUI\BUTTON "SIC\BREAK" BREAK
GUI\BUTTON "SIC\EXIT" "Exit"
!
GUI\MENU "Graphic"
GUI\BUTTON "DEVICE IMAGE WHITE"  DEVICE
GUI\BUTTON "CLEAR PLOT"    CLEAR
GUI\BUTTON "ZOOM"          ZOOM
GUI\BUTTON "HARDCOPY /PRINT" HARDCOPY
GUI\MENU /CLOSE
!
GUI\MENU "Program(s)"
GUI\BUTTON "@ PR:X_WHOLE.GRAPHIC"    "Interfero"
GUI\BUTTON "@ PR:X_DISPLAY.GRAPHIC"  "Display"
!
GUI\GO
```

Several detached menus can be activated at once.

2.4.2 Assigning variables in "Window" mode

When the "Main panel window" has been created, the LET command behaves in a different way when any of the following options is set: `/PROMPT, /CHOICE, /INDEX, /RANGE, /FILE`

Rather than taking the variable value from the keyboard-typed command line, the LET command has no immediate action, but defines a widget in the "Main panel window" (or "sub-panels"). This widget will allow the user to define the variable value using the windowing system. 5 types of widgets are available:

- Text widget
  This is the default widget created when option `/PROMPT` is present. The content of the text widget will be used to set the variable.

- Slider widget
  This widget is activated when option `/RANGE` is present. The widget consists both in a numeric area and a slider limited by the given range, and can be used to set a real or integer variable.

- Choice widget
  This widget is activated when option `/CHOICE` is present. The widget consists in a text widget and a pulldown menu containing all specified choices. The user can select its choice with this menu. If the last choice is a "*", any other text can also be entered.

- Index widget
  This widget is activated when option `/INDEX` is present. This widget is similar to the Choice widget, but the returned value is an integer corresponding to the sequence number of the selected choice.

- File widget
  This widget is activated when option `/FILE` is present. The widget consists in a text widget and a file selection widget with the specified file filter.

The "Main panel window" is actually created with all defined widgets when command `GUI\GO` is typed.

2.4.3 Actions and Buttons in "Window" mode

The "Main panel window" is created when command `GUI\GO` is typed. Four buttons are defined by default in this window:

- the OK button,
  which sets all variables defined in the main *and sub-panel* windows. This button also executes the command passed as argument to the `GUI\GO` command (if specified).
- the UPDATE button,
  which sets all variables defined in the main *and sub-panel* windows, without executing the (optional) command passed as argument to the `GUI\GO` command.
- the ABORT button.
  Variables are not modified, and an error is sent to the main program.
- the HELP button.
  This button displays the help file specified in the `GUI\PANEL` command.

Additional buttons can also be added to the "main panel window" using the `GUI\BUTTON` command. Two types of buttons exist

- Buttons with no associated parameters. These buttons appear just after the 3 main buttons.
- Buttons with optional parameters. These buttons appear sequentially with the variables, and have an associated parameter or "*optional*" window. When such a button is defined, all subsequent *LET* commands create a widget in a "sub-panel" window. This window is by default hidden, but can be unveiled by the user.

The "sub-panel" windows provide a way to hide some non-essential parameters, and/or to create a "main window" with a control panel defining many actions, each action having its own input window and separate help.

2.4.4 Help file structure

Command `GUI\PANEL` allows to associate an help file to the main window or detached menus, and command `GUI\BUTTON` does the same for optional windows.

The `HELP` files format should be

```
1 Description
    General help for the window or menu
2 NAME1
    help for variable NAME1
2 NAME2
    help for variable NAME2
1 ENDOFHELP
```

where 1 and 2 are in the first column of the text file, and followed by a single space.

Clicking on the HELP button will display the complete help file in a scrolled window. Clicking in the prompt area of an input variable will display the associated variable name and the help for this variable (if it exists).

2.5 Interacting with the Operating System

Since SIC was designed to be portable on various operating systems (Unix, Mac-OS, MS-Windows, and in the past the defunct VMS), interaction with the operating system is normally kept to a minimum.

However, many operations eventually deal with files handled by the operating system. To avoid platform dependencies, SIC allows some basic file operations through the `SIC` command.

When control is desired at the operating system level, the command `SYSTEM` can be used to access it without loosing the SIC context. For user convenience, SIC also accepts the short-cut

```
$ operating_system command
```

instead of the more conventional SIC-like syntax

```
SYSTEM "operating_system command"
```

2.5.1 File Operations

Operations on the file system can be done directly within SIC, using the `SIC` command.

`SIC> SIC DIRECTORY [NewDir]`
to control the working directory.

`SIC> SIC MKDIR NewDir`
to create a new directory.

`SIC> SIC\SIC APPEND FileIn FileOut`
Appends file `FileIn` to `FileOut`.

`SIC> SIC\SIC COPY FileIn FileOut`
Copies file `FileIn` to `FileOut`.

`SIC> SIC\SIC DELETE File`
Deletes file named `File`. Caution: no confirmation is required.

`SIC> SIC\SIC RENAME FileOld FileNew`
Renames file `FileOld` to `FileNew`. Both files should reside on the same disk. To move files across different disks, use `SIC COPY` and `SIC DELETE`.

`SIC> SIC\SIC FIND FileFilter`
search for files with names matching the specified filter, and return the result in a structure named `dir%`. `dir%nfiles` is the number of files found, and `dir%files[1:dir%nfiles]` their names. See the internal help for details.

The file operations through the SIC command should be used preferentially to operations through the `SYSTEM` command for portability, specially in command procedures.

2.5.2 SYSTEM Command: Unix-like operating system

`SYSTEM ["Command"]`

Without argument, the `SYSTEM` command will create a subshell, using the user's default shell (sh, csh, ksh, etc...). Control will return to the calling program once the subshell terminates (i.e. using the 'exit' or 'bye' or 'logout' command).

With an argument, the `SYSTEM` command will execute the corresponding Unix command in a subshell.

Note that because subshells are used, you cannot change environment variables in this way. In particular, to change your working directory use the `SIC DIRECTORY` command.

2.6 Customizing

2.6.1 Logical Names

All programs based on SIC read in files specifying *Logical names* which are used when SICmust refer to external files (such as procedure or images). *Logical names* are similar in syntax and functions to VMS logical names. Two files define general logical names required for SIC based programs to work properly (where to find help files for example) and site-specific features (such as printer name, scratch space, etc...).

On UNIX and Mac-OS systems, users can specify there own logical names in the file

`$HOME/.gag.dico`

On MS-Windows systems, "personal" logical names should be set in the file

`$GILDAS\dico.lcl`

where `$GILDAS` is the top directory of the Gildas software as defined at installation (normally `\Program_file\Iram\Gildas` see in the `autoexec.bat` file on Windows-95).

Logical names can also be added or modified at run time using the `SIC LOGICAL` command.

2.6.2 User Defined Commands

SIC allows the user to define new command, by means of the `DEFINE COMMAND` command. The syntax is the following:

```
DEFINE COMMAND Newcom "Old Command with arguments" [Help_File]
```

where `NEWCOM` is the name of the new command and `HELP_FILE` is an optional text file used to provide help about the command. Because `DEFINE COMMAND` does not provide any specific syntax to specify the use of the new command arguments, it is in practice used essentially to allow documented access to procedures, as for example in

```
DEFINE COMMAND INPUT "@ gag_pro:p_input.greg" gag_pro:input_greg.hlp
```

2.6.3 Initialization File

Although this is not a default feature of SIC, many programs using SIC call a initialization macro at run time to define symbols, execute startup commands and so on. This file is typically named

```
GAG_INIT:INIT.DefExt
```

where `DefExt` is the default macro extension used by the program, and is usually the program name. Consult the specific programs documentation. `GAG_INIT` is a *logical name* that normally points to your login directory, or (by default) on $HOME/.gag/init/ on Unix-like systems.

2.6.4 The SIC Command

The `SIC` command controls several internal parameters of the SIC monitor. It has several categories of actions

File system related operations, described in Sect.2.5.1 `APPEND COPY DELETE DIRECTORY FIND MKDIR MODIFIED PARSE RENAME`

Procedure related operations: `EXPAND EXTENSION MACRO OUTPUT SAVE VERIFY WHICH`

Customization `EDIT HELP INTEGER LOGICAL MESSAGE PRECISION SYSTEM TIMER WINDOW`

Command interpretation `Language\ PRIORITY SYNTAX`

Miscellaneous actions `BEEP CPU DATE DEBUG DELAY FLUSH LOCK RANDOM_SEED USER WAIT`

The `SIC` command without arguments produces a summary of the internal SIC status, and (when that makes sense) with a single argument it shows the status of that argument. Please refer to the internal HELP for details.

3 Running Tasks

This section contains the minimum information required to use the GILDAS image processing tasks.

To run tasks, use the commands `RUN` and `SUBMIT` from the `VECTOR\` language. Both commands are very similar. The `RUN` command will execute the task as a detached process, and the `SUBMIT` command in a batch queue named `GILDAS_BATCH`.

The `VECTOR\` language contains the following commands :

```
FITS            : A simple FITS -- Gildas conversion tool
RUN Program     : Activates a GILDAS task in a detached process
SPY [Task]      : Look at the status of one or all GILDAS tasks.
SUBMIT Program  : Submit a GILDAS task to GILDAS_BATCH queue
TRANSPOSE       : A command to transpose a Gildas data file.
```

3.1 Window Mode

The window mode is the default mode on X-Window systems with Motif interface. Let us assume in the following example we want to execute a task named `example`. To activate `example`, the user will type

```
     VECTOR> RUN example
or
     VECTOR> SUBMIT example
```

A separate input window is created: The user can then modify any of the parameters by clicking in the dialog areas. Help can be obtained by clicking on the HELP button, or on any parameter description.

Since SIC is used, parameter values can be variables or arithmetic expressions (e.g. `2*PI+EXP(X[3])` is a perfectly valid value for a real, provided the array `X[n]` with `n>3` has been previously defined).

Once all parameters are defined, the task can be launched by clicking the OK button, or aborted using the ABORT button. Parameter values are checked, and if all parameters are valid, the task is executed (or submitted). If one parameter is invalid, the `RUN` or `SUBMIT` command sends back a message :
`E-RUN, Missing GO command`
and returns an error.

3.2 Query Mode

When no window-mode is available, the user is prompted for the parameters. In this example, the dialog will be

```
     An integer value
     INTEGER     I$           1 <CR>
     A value between 0 and 1
     REAL        A$           0.1 <CR>
     Any character string
     CHARACTER   CHAIN$       ABCD <CR>
     4 Real values
```


EXAMPLE
GO
UPDATE
ABORT
HELP
A value between 0 and 1
0.23
A random text string
Salut Arthur
4 arbitrary values
1 2 3 4
A valid file name
myfile.gdf
An existing file name
thisone.exe

```
REAL        ARRAY$[4]    1 2 3 4 <CR>
A valid name
FILE        FILE$        TESTFILE.DAT <CR>
Any values
VALUES      OLD$         acos(-1) 1.234 <CR>
```

The prompting method is always the same: an explanatory first line indicating the meaning of the parameter, and a second line in the following format:

```
TYPE NAME[Dimensions]
```

where

- `TYPE` indicates the type of parameter `(CHARACTER, FILE, INTEGER, LOGICAL, REAL)`. A parameter of type `FILE` is a character string containing a valid file name. `VALUES` is intended to hold multiple values (which can be mathematical expression), without any prior on the number of values.
- `NAME` is the parameter name
- `[Dimensions]` are the parameter dimensions, in case it is an array. Only `REAL` and `INTEGER` parameters may be arrays.

Query mode is also used for missing parameters in Window-mode.

3.3 EDIT Mode

Commands `RUN` and `SUBMIT` execute two SIC command procedures, the *Initialization File* `Task.init`, which defines all parameters needed for `example`, and the *Checker File* `Task.check`, which verifies that all parameters are valid. In the example above, the `example.init` file is

```
TASK\INTEGER "An integer value"    I$
TASK\REAL "A value between 0 and 1" A$
TASK\CHARACTER "Any character string" CHAIN$
TASK\REAL "4 Real values" ARRAY$[4]
TASK\FILE "A valid file name" FILE$
TASK\VALUES "Any values"  OLD$
TASK\GO
```

This is a standard procedure containing commands of a SIC language named TASK\. Commands from this language are used to define the parameters required by the task, and cannot be called interactively. The command syntax is always the same :

```
TASK\Command "Prompt String" Parameter$[Dimensions] [Value [...]]
```

where

- `Command` indicates the type of parameter (`CHARACTER, FILE, INTEGER, LOGICAL, REAL`). A parameter of type `FILE` is a character string containing a valid file name. `VALUES` is intended to hold multiple values (which can be mathematical expression), without any prior on the number of values.

- "Prompt String" is a character string used as a prompt to ask for the parameter value(s)
- Parameter$ is the parameter name
- [Dimensions] are the parameter dimensions, in case it is an array. Only REAL and INTEGER parameters may be arrays.
- Value(s) are the parameter values, an array requiring as many values as array elements.

Once all parameters have been assigned values, RUN and SUBMIT commands execute the example.check file, writing the current parameter values in an auxiliary file which will be used by the task example. If a parameter is incorrect, an error is returned, and the task example not executed.

Instead of supplying the parameters in a query mode, the user can use a text editor to edit the .init file using command

```
     VECTOR> RUN example /EDIT
```

or

```
     VECTOR> SUBMIT example /EDIT
```

The parameter values can then be typed after the parameter names in the example.init file, using SIC continuation marks ("-" as the last character of a line) if needed for long command lines. example.init will be executed after exiting the editor. If a parameter value is missing, the user will nevertheless be prompted for it after exiting the editor.

The text editor called is user defined by the command SIC\SIC EDITOR or the logical name GAG_EDIT.

3.4 Specifying the .init File

By default, in Query mode RUN and SUBMIT commands use the .init file located in TASK#DIR: search path. In EDIT mode, the .init file located in the current default directory is used if it exists. To override this default behaviour, you can specify any .init file as the second argument to commands RUN and SUBMIT.

3.5 Errors and Aborting

If an error occurs in the .init or .check procedure, the erroneous command will be returned to the user, and the procedure execution is interrupted by a pause. You can then correct the error, execute the command, and type C or CONTINUE to resume the procedure execution. Or you can type QUIT (as in any SIC procedure indeed) to abort the execution, until the RUN or SUBMIT command returns an error.

You may also want to abort a RUN or SUBMIT command while you are in the editor: typing QUIT instead of EXIT to end the editing will do it.

3.6 Log Files

A log file is created by the RUN command in your GAG_LOG: directory with the task name as file name and the extension .gildas; this log is printed by the SUBMIT command. If the user is still running the main program (GreG or Mapping, etc...) when a task completes, he (or she) is warned of the completion with the return status. Log files are not purged automatically, so that

you should take care of that. They are intended essentially as a debugging aid if something goes wrong, but you can print them as archive of your data processing.

A command file is created in your `GAG_LOG:` directory to run or submit the programs. It is in principle deleted at task completion.

3.7 Synchronizing Jobs

The batch queue `GILDAS_BATCH` should have a job limit of 1, so that all tasks submitted by command `SUBMIT` execute in sequence. There may even be intervening jobs from other users.

Tasks activate by command `RUN` must complete before control is returned to the user.

Command `SPY` can be used to monitor the execution of tasks activated by command `RUN`.

3.8 Obtaining Explanations: `HELP RUN TaskName` Command

Help on each individual task can be obtained with the `HELP RUN` followed by the task name. Help on their parameters can be obtained by adding the parameter name, or * for all parameters.

4 SIC Programming Manual

The SIC programming manual has been moved to the GILDAS programming guide.

5 SIC Language Internal Help

5.1 Language

```
                    SIC\ Command Language Summary

ACCEPT         : Read variable in various format.
BEGIN          : Begin a sub-procedure, help file or data file.
BREAK          : Exit without error from a FOR-NEXT loop.
COMPUTE        : Execute non-arithmetic operations on variables.
CONTINUE       : Resume macro or loop execution after PAUSE.
DATETIME       : Convert date and/or time to/from various formats
DEFINE Type V  : Define new variables.
DELETE         : Delete variables or symbols.
DIFF Var1 Var2: List the differences between two variables or files.
EDIT [File]    : Edit a file or a dump of the Stack.
ELSE [IF Log]  : Alternate IF block directive.
END            : End IF block structure or a sub-procedure.
EXECUTE        : Execute a command line
EXAMINE Var    : Type the current value of the specified variable.
EXIT [Code]    : Exit from the program.
FOR            : Open a FOR-NEXT loop. Also FOR /WHILE or FOR /IN.
HELP XX[\]     : Give an explanation of command XX or language XX\.
IF Logical     : Start a conditional IF block.
IMPORT Package: Dynamically import another package in current one.
LET            : Assign value to variable.
MESSAGE        : Send a message to screen and/or message file.
MFIT A=F(B,p)  : Fit a formula into SIC variables.
MODIFY         : Modify the spectroscopic axis of a cube.
NEXT           : End FOR loop definition and activates the execution.
ON ERROR COMM  : Change the current error recovery action.
PAUSE          : Set a break point in a Loop or a Macro.
PYTHON         : Start/End intercommunication between SIC and PYTHON.
QUIT           : Abort an execution interrupted by PAUSE.
RECALL [Text]  : Recall line from stack, and edit it if possible.
RETURN         : Terminate procedure execution.
SAY "text"     : Type a text or variable or expression value.
SIC Arg        : List or Change some SIC internal status.
SORT Key Vars  : Sort variables according to another one.
SYMBOL         : Define, list and delete symbols.
SYSTEM         : Create or attach sub-processes, run system commands.
TIMER          : Check or customize the SIC timer.
TYPE [XX]      : List file XX or the stack.
@ XX [P1 ...]  : Read commands from macro XX and executes them.
```

5.2 ACCEPT

[SIC\]ACCEPT Var_Name /ARRAY File_Name [/options]
[SIC\]ACCEPT Var_Name /BINARY File_Name Offset
[SIC\]ACCEPT Var_1 [Var_2 [...]] /COLUMN File_Name [Separator] [/options]

Read SIC variables from formatted or binary files. This command has 3 major modes: /ARRAY to read in a FORMATTED way a SINGLE n-dimensional variable, /BINARY to read in BINARY form a SINGLE n-dimensional variable, and /COLUMN to read in a FORMATTED way SEVERAL 1-dimensional variables.

5.2.1 ACCEPT /ARRAY

[SIC\]ACCEPT Var_Name /ARRAY File_Name [/FORMAT String] [/LINE Begin End]

Read a N-Dimensional variable of name Var_Name from a formatted file File_Name, using list-directed (free) format, or a user specified format if /FORMAT option is present. The /LINE option can be used to skip some lines before starting reading. Not recommended for character arrays.

5.2.2 ACCEPT /BINARY

[SIC\]ACCEPT Var_Name /BINARY File_Name [Skip]

Read a N-Dimensional variable of name Var_Name from a binary file File_Name. The optional Skip argument indicates how many BYTEs to skip before starting reading.

The command COMPUTE LINES /BLANK can be used to count the number of (useful) lines in a file and to define the output arrays with the appropriate size.

5.2.3 ACCEPT /COLUMN

[SIC\]ACCEPT Var_1 [Var_2 [...] /COLUMN File_Name [Separator] [/FORMAT String] [/LINE L1 [L2]]

Read ONE or SEVERAL 1-Dimensional variables in a flexible format from a formatted file File_Name. Each variables can be of any length, from 0 (scalar) to N. They are filled by reading each associated column, starting from first line. Data is read in as many lines as required. If end of file is reached in the meantime, an error is raised.

Characters after a "!" are considered as comments and are ignored. Blank or full comment lines (starting with a "!") are skipped. The starting line number can be selected with /LINE option.

A * as an argument of the ACCEPT command indicates a dummy variable used to skip a column in the input file. For example, the command

```
ACCEPT A * B C /COLUMN TEST.DAT
```

reads A,B and C from columns 1,3 and 4 of file TEST.DAT, since the * indicates to skip the second column.

Option /FORMAT can also be used to specify a fixed Fortran-like format.

By default, the format is the equivalent of the Fortran list-directed (*) format. In this format, character strings must be included between quotes. The default can be changed by specifying a separator as a second argument of option /COLUMN. The separator can be " " (which gives a behaviour like the COLUMN command of GreG), or any other character. For example, specifying "\;" as a separator can allow to read CSV files (e.g. Excel data files).

Example of list-directed read (note the quotes are taken into account):

```
SIC> type test.dat
123 "Hello, world!" 3.14
SIC> define integer i
SIC> define character c*16
SIC> define real r
SIC> accept i c r /column test.dat
SIC> exa i c r
I               =          123           ! Integer GLOBAL
C               = Hello, world!          ! Character*16 GLOBAL
R               =    3.140000            ! Real    GLOBAL
```

In most cases, a list-directed is equivalent to use space as separator. Only when quotes or blanks in strings are involved makes a difference.

Example of blank-separated read (in this case, providing the blank separator or not makes no difference):

```
SIC> type test.dat
123 Hello 3.14
SIC> accept i c r /column test.dat " "
SIC> exa i c r
I               =          123           ! Integer GLOBAL
C               = Hello                  ! Character*16 GLOBAL
R               =    3.140000            ! Real    GLOBAL
```

Example of blank-separated read (in this case, the blank separator MUST be given):

```
SIC> type test.dat
123 1.23" 4.56"
SIC> define character l*12 m*12
```

```
SIC> accept i l m /column test.dat " "
SIC> exa i l m
I               =          123           ! Integer GLOBAL
L               = 1.23"                  ! Character*12 GLOBAL
M               = 4.56"                  ! Character*12 GLOBAL
```

5.2.4 ACCEPT /FORMAT

Specify a fortran format to read the input file for command ACCEPT /ARRAY or ACCEPT /COLUMN. Mixing CHARACTER and NUMERIC (REAL,...) arrays is not available yet, and only one character array can be read at a time. Blank or commented lines are not skipped.

Example:

```
SIC> TYPE cities.dat
  Toulouse          1.111    11.11111
  Bordeaux          2.222    22.22222
  Grenoble          3.333    33.33333
  Marseille         4.444    44.44444
(truncated)
SIC> DEFINE CHARACTER CITY*16[44]
SIC> DEFINE REAL X[44] Y[44]
SIC> ACCEPT X Y /COLUMN "cities.dat" /LINE 4 /FORMAT "20x,F8.3,1x,F8.3"
SIC> ACCEPT CITY /COLUMN "cities.dat" /LINE 4 /FORMAT "2x,A16"
```

This option is invalid with /BINARY.

5.2.5 ACCEPT /LINE

```
[SIC\]ACCEPT Var_Name /LINE L1
```

Indicate the first line to be read in the input file for command ACCEPT /ARRAY or ACCEPT /COLUMN. Blank and commented lines have to be taken into account in L1.

This option is invalid with /BINARY.

5.2.6 ACCEPT Excel

Excel (TM) datafile may be readable by command ACCEPT. They must be converted to CSV format (Comma Separated Values). A suitable use of the dummy variables in ACCEPT command allows to skip the non-numeric fields. The proper separator must then be specified using the second argument of option /COLUMN: e.g.
ACCEPT * Var1 Var2 /File Excel.csv ";" /LINES 2 20
will read Var1 and Var2 from lines 2 to 20 of the corresponding Excel file, from Excel columns "B" and "C" (second and third).

To be documented: Can character variables also be read with numeric ones provided some adequate /FORMAT option ?

5.3 BEGIN

```
[SIC\]BEGIN Procedure|Help|Data FileName
```

Begin a new Procedure, Help file, or ASCII Data file. All lines until the corresponding END Procedure|Help|Data FileName command is found are considered to be the body of the new file. Such files are located in the directory designated by the logical name GAG_PROC:.

5.4 BREAK

```
[SIC\]BREAK
```

Terminate a loop execution. The two uses of command BREAK are usually

```
    ON ERROR BREAK
    FOR I 1 TO 3 BY 0.5
      ...
    NEXT
```

or

```
    LET A = C               ! A is a known variable
    FOR I 1 TO 100 BY 1
      ...
      IF I+A.EQ.0  BREAK
    NEXT
```

BREAK differs from QUIT because it does not transmit any error.

5.5 COMPUTE

```
[SIC\]COMPUTE OutVar OPERATION InVar [Parameters] [/BLANKING Bval [Eval]]
```

Perform operations or transformations on variables that are not directly supported by the array capabilities of the SIC command interpretor. OutVar is the output variable, InVar the input variable. Outvar must be defined beforehand.

The following operations are available on INTEGER, REAL and DOUBLE PRECISION arrays, regardless of their nature:

- MAX
  MIN
  MEAN
  RMS (standard deviation)
  SUM
  PRODUCT

```
    MEDIAN
    for which InVar is an array of rank 1 to 7 (a 1-D to 7-D array), and
    OutVar must be an array of lesser rank (i.e., one or more  dimension
    less  than InVar, down to a number), AND of identical shape as InVar
    for the dimensions in common.
    Example (if A[4,12,2,8] and B[4,12]):
            COMPUTE B MAX A
    Sections, implicit transpositions, etc... permitted by SIC are  sup-
    ported.
  - HISTOGRAM: see subtopic HISTOGRAM for details.
Blanking  values Bval and Eval allow to ignore values of the Invar array
if at Eval from Bval, for these operations only. At least Bval  must  be
passed  to  option /BLANKING, and default is 0.0 for Eval.  SET BLANKING
has no effect here since it is a command of the GREG1\ language.
In case of no valid result, NaN is returned, or Bval if blanking is  en-
abled. This may occur in particular when all InVar values are blanked.

The following transformations are available:
  - GATHER
    Return the list of different input values (see subtopic for details)
  - RANKORDER
    Return the ordering of the input values.
  - INTEGRAL or DERIVATIVE
    Integral or derivative of the input variable
  - FFT+ or FFT-
    Direct  or  Inverse Fast Fourier Transform: see subtopic FFT for de-
    tails.
  - FOURT+ or FOURT-
    Direct or Inverse Fast Fourier Transform: see subtopic FOURT for de-
    tails.
  - COMPLEX
    Populates  the  Real and (optionally) Imaginary part of OutVar (com-
    plex variable) with InVar (real) and (optionally) the Imaginary part
    by the following (real) variable name.
  - REAL
    OutVar = REAL part of InVar. OutVar is Real, InVar is Complex .
  - IMAG
    OutVar = IMAGE part of InVar. OutVar is Real, InVar is Complex.
  - ABS
    OutVar = AMPLITUDE (InVar). OutVar is Real, InVar is Complex.
  - PHASE
    OutVar = PHASE (InVar). OutVar is Real, InVar is Complex.
  - CMP*  operations  as  in  "COMPUTE OutVar CMPMUL InVar1 InVar2". All
    variables must be complex arrays of identical size.
      CMPADD: OutVar = InVar1 + InVar2.
      CMPSUB: OutVar = InVar1 - InVar2.
      CMPMUL: OutVar = InVar1 * InVar2.
```

```
      CMPDIV: OutVar = InVar1 / InVar2.

Operations on files:
  - DATE: File last modification time
  - LINES: Number of lines in a formatted file

Miscellaneous operations (see subtopics for details):
  - DIMOF
  - GAG_DATE
  - LOCATION
  - BTEST Test bit values
  - IS_A_SIC_VAR
```

5.5.1 COMPUTE DATE

```
    [SIC\]COMPUTE Modification_Date DATE Filename
```

Return the last modification date of the file Filename into the integer variable Modification_Date. Used in procedures to check when file has last been changed.

If the output variable is a 4-bytes integer (Fortran I*4), returned value is in seconds since 01-jan-1970. If the variable is a 8-bytes integer (Fortran I*8), returned value is in nanoseconds since 01-jan-1970. In the latter case, your system (e.g. Linux >= 2.5.48) AND filesystem (e.g. ext4) must support timestamp granularity below the second. If not, value precision is limited to 1 second.

5.5.2 COMPUTE DIMOF

```
    [SIC\]COMPUTE OutVar DIMOF InVar
```

Return the shape of array InVar in OutVar[1:7], and its rank in OutVar[8]. For this purpose, OutVar must be an INTEGER array of dimension [8]. Undefined dimensions are set to 0.

See also HELP FUNCTION RANK and HELP FUNCTION SIZE.

5.5.3 COMPUTE FFT

```
    [SIC\]COMPUTE OutVar FFT+|FFT- InVar [REAL]
```

FFT+ performs Direct Fast Fourier Transform, while FFT- performs Inverse Fast Fourier Transform.

Command accepts REAL or COMPLEX variables. OutVar is a 2-D array with second dimension equals 2, storing respectively the Real and Imaginary part of the (complex) output Fourier transform. By default, InVar is like OutVar, but if parameter REAL is specified InVar is a 1-D array.

5.5.4 COMPUTE FOURT

```
[SIC\]COMPUTE OutVar FOURT+|FOURT- InVar
```

FOURT+ performs Direct Fast Fourier Transform, while FOURT- performs Inverse Fast Fourier Transform.

Operates on input and output on COMPLEX arrays of dimension [2,NX,NY].

5.5.5 COMPUTE GATHER

```
[SIC\]COMPUTE OutArray GATHER InArray
```

Returns in a NEW 1-D variable named OutArray the list of values found in the (existing) n-D variable InArray. The output array must not exists and is created by the program itself. It is of the same type as the input array.

NaN and Blanked values are ignored.

5.5.6 COMPUTE GAG_DATE

```
[SIC\]COMPUTE IntDate GAG_DATE "15-DEC-2035"
[SIC\]COMPUTE StrDate GAG_DATE 4127
```

GAG_DATE keyword converts a string date into a "radio Julian date" (integer value), or vice-versa. The kind of the output variable (resp. integer or character) rules the expected kind for input variable (resp. character or integer). The output variable must be scalar and writeable, and the date string should be 11 characters at least.

A "radio Julian date" (or "Jansky Julian date") starts as -2^15 on the date of the first radio observation by Karl Jansky. It is thus the Modified Julian date minus 60549. That choice was made to maximize the time interval over which radio astronomical data could be usefully stored in an integer*2, back when 2 bytes of header space per spectrum were a significant consideration. This date has little meaning outside the rather sparse community of souls gathered around the CLASS program, however.

5.5.7 COMPUTE HISTOGRAM

```
[SIC\]COMPUTE OutVar HISTOGRAM InVar [Hmin] [Hmax] [/BLANKING Bval [Eval]]
```

Put in variable OutVar (dimension [n,2]) the histogram of values of n-D variable InVar, eventually between the cuts Hmin and Hmax, and with blanking values Bval and Eval (that is, values of the Invar array are not taken in account for the histogram if at Eval from Bval). These pa-

rameters can be absent. One can use '*' to omit any of them.

The number of bins is dictated by the first dimension of the array OutVar. In the absence of said cuts, the cuts used are the maximum and minimum values of the InVar array. The OutVar variable contains the histogram in its first column (OutVar[1]) and the corresponding bin value in the second column (OutVar[2]).

5.5.8 COMPUTE IS_A_SIC_VAR

[SIC\]COMPUTE OutVar IS_A_SIC_VAR InVar

Determines whether InVar (Character string) is the name of an existing SIC Variable or not. OutVar must be a scalar Logical variable.

The functionality is similar to the build-in SIC function EXIST, but EXIST would complain about invalid filenames as arguments. Here, InVar can be any arbitraray string.

This command is used in some scripts to determine silently whether its arguments are File names, variable names or other expressions (see e.g. p_load.greg for an application.)

5.5.9 COMPUTE LINES

[SIC\]COMPUTE OutVar LINES FileName [/BLANK]

Count the number of lines in a formatted file. More precisely, it counts the number of carriage returns (same as Linux shell "wc -l"). Be careful of missing carriage return at the end of your files.

The option /BLANK can be used to ignore blank and comment (starting with "!") lines, in the same way the command ACCEPT does.

5.5.10 COMPUTE LOCATION

[SIC\]COMPUTE OutVar LOCATION InVar Value

Search for Value in the array InVar, and puts in OutVar the 2 nearest indices of InVar for which its values surrounds Value. These variables have some requirements:

- InVar must be a 1D-array of REAL or DOUBLE values,
- Value must be scalar numeric,
- OutVar must be a 1D-array of 2 INTEGER values.

InVar is also intended to be ordered (ascending or descending), or else the result is unpredictable. Use SIC\SORT to sort arrays.

5.5.11 COMPUTE RANKORDER

```
[SIC\]COMPUTE OutVar RANKORDER InVar
```

Compute the order (sorting array) corresponding to the values in InVar, e.g.

```
SIC> define real a[4] b[4]
SIC> let a 10 15 12 9
SIC> compute b rankorder a
SIC> exa b
B               is a real Array     of dimensions  4
   4.000000       1.000000       3.000000       2.000000
```

InVar and OutVar are R*4 or R*8 variables, assumed to be 1-D variables. This can be used e.g. for statistical non-parametric tests like the Spearman test.

5.5.12 COMPUTE INTEGRAL

```
[SIC\]COMPUTE OutVar INTEGRAL InVar
```

Compute the integral of the InVar variable, in the classical sense, i.e.

```
OutVar[i] = InVar[1]+Invar[2]+...+Invar[i]
```

InVar and OutVar are assumed to be 1-D variables

5.5.13 COMPUTE DERIVATIVE

```
[SIC\]COMPUTE OutVar DERIVATIVE InVar
```

Compute the derivative of the InVar variable, in the classical sense, i.e.

```
OutVar[i] = InVar[i+1]-Invar[i]
```

InVar and OutVar are assumed to be 1-D variables, of size N. OutVar[N] is linearly extrapolated from the N-1 and N-2 values.

5.5.14 COMPUTE BTEST

```
[SIC\]COMPUTE OutVar BTEST InVar [Ibit]
```

Bit-test one or several bit values of the input variable, and return true for each bit equal to 1.

The optional integer argument Ibit indicate which bit should be tested. The default is to check all the bits.

If all the bits of (a single element of) InVar are tested, OutVar must be an array with its first dimension equal to this number of bits, e.g. 32 for INTEGERs, 64 for DOUBLEs, etc. InVar may be a multi-dimensional

```
array, in which case OutVar must have the same extra-dimensions.

In  details,  this tool is an overlay to the BTEST Fortran function (ex-
cept that bit numbering starts at 1 here). Note that this hides the  ma-
chine dependencies (IEEE, EEEI, etc) by using a unique integer model re-
gardless of its physical layout. For more information, please  refer  to
the Fortran documentation.

Examples:

SIC> define integer i
SIC> let i 2
SIC> define logical one
SIC> compute one btest i 1  ! First bit value
SIC> exa one
ONE             = F                          ! Logical GLOBAL

SIC> define logical all[32]
SIC> compute all btest i  ! All bit values
SIC> exa all
ALL             is a logical Array     of dimensions  32
 F T F F F F F F F F F F F F F F F F F F F F F F F F F F F F
 F F F F F F F F

SIC> define integer i2[3]
SIC> let i2 1 2 3
SIC> define logical one2[3]
SIC> compute one2 btest i2 2  ! Second bit values of the 3 elements
SIC> exa one2
ONE2            is a logical Array     of dimensions  3
 F T T

SIC> define logical all2[32,3]
SIC> compute all2 btest i2  ! All bit values of all elements
SIC> exa all2[1] all2[2] all2[3]
ALL2[1]         is a logical Sub-Array of dimensions  32
 T F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
 F F F F F F F F
ALL2[2]         is a logical Sub-Array of dimensions  32
 F T F F F F F F F F F F F F F F F F F F F F F F F F F F F F
 F F F F F F F F
ALL2[3]         is a logical Sub-Array of dimensions  32
 T T F F F F F F F F F F F F F F F F F F F F F F F F F F F F
 F F F F F F F F
```

5.6 CONTINUE

```
   [SIC\]CONTINUE     or     C
```

Resume loop or macro execution after a PAUSE (explicit or caused by an error or a <^C>). Typing C instead of CONTINUE will always do exactly the same thing. This is the only superior abbreviation installed in SIC.

5.7 DATETIME

[SIC\]DATETIME /FROM Date-Time-Spec /TO Outvar1 Outformat1 [... OutvarN OutformatN]

Convert a date and/or time between various formats, storing result in one or more output variables. The options /FROM and /TO must be present.

5.7.1 DATETIME /FROM

```
[SIC\]DATETIME /FROM NOW
[SIC\]DATETIME /FROM Val1 Form1 ... ValN FormN
```

The option /FROM is used to define a single and non-ambiguous date-time. The date-time is fully specified as the combination of a year, month, day, hour, minute, and seconds. They can be described with the following pairs:

```
  Iye YEAR (integer, default 1970)
  Imo MONTH (integer, default 1)
  Ida DAY (integer, default 1)
  Iho HOUR (integer, default 0)
  Imi MINUTE (integer, default 0)
  Nse SECONDS (real, default 0.0)
```

The pairs can be combined in any order, but each field must be defined once at most. If a field is not specified, its default value is used. The day, hour, minute, and seconds fields can go beyond their usual ranges: the resulting date-time is shifted accordingly. For example:

```
  DATETIME /FROM 2015 YEAR                 ! 2015-01-01 00:00:00.000
  DATETIME /FROM 2015 YEAR 11 MONTH 16 DAY ! 2015-11-16 00:00:00.000
  DATETIME /FROM 2015 YEAR 100 DAY         ! 100-th day of year 2015
```

The keyword NOW can be used alone to specify the current UTC date-time. The values PREVIOUS or NEXT can also be used for each field: they resolve as the previous or next quantity with respect to the current UTC date-time. These syntaxes define all the fields at once, i.e. they can not be combined with other pairs of values. For example:

```
  DATETIME /FROM NOW           ! Now (UTC)
  DATETIME /FROM PREVIOUS DAY  ! Yesterday, same time
  DATETIME /FROM NEXT HOUR     ! Today or tomorrow, in 1 hour from now
```

```
The option /FROM also recognizes specific date and/or time formats:
  JULIAN       Julian date with fractional day (e.g. 2457343.227256944)
  MJD          Modified Julian Date (e.g. 57342.72725694445)
  ISO          ISO date (e.g. 2015-11-16T17:27:15.000)
  GAG_DATE     GAG date integer format (e.g. -3207)
  YYYYMMDD     date with all numbers (e.g. 20151116)
  DD-MMM-YYYY  date where MMM is the month in letters (e.g. 16-NOV-2015)
  RADIAN       time of day in radians, usually betwen 0 and 2*pi
  SEXAGESIMAL  time of day using sexagesimal notation (e.g. 17:27:15.000)

JULIAN, MJD and ISO define the 6 date and time fields: they can not be
combined with anything else. GAG_DATE, YYYYMMDD, and DD-MMM-YYYY define
the date: they can be combined with a time specifier. RADIAN and SEXA-
GESIMAL define the time: they can be combined with a date specifier. For
example:
  DATETIME /FROM 2015-11-16T17:27:15.000 ISO
  DATETIME /FROM 57342.72725694445 MJD
  DATETIME /FROM -3207 GAG_DATE 4.569490147 RADIAN

The inputs can be either explicit scalar values, scalar variables, or
array variables. They can be mixed as long as arrays have equal size.
For example
  DATETIME /FROM "16-NOV-2015" DD-MMM-YYYY ArrayVar RADIAN
results in an array specification, where all the specified values share
the same date but each time is taken in ArrayVar.
```

5.7.2 DATETIME /TO

```
[SIC\]DATETIME /TO Outvar1|* Outformat1 [... OutvarN|* OutformatN]
```

```
The date-time(s) defined by the /FROM option is converted and transfered
to the output targets. A target can be a variable (given its name) or
the terminal (if * is given). There is no restriction on their kinds and
numbers, i.e. you can convert the date-time to all the possible formats,
or repeat the same conversion for several targets. Supported formats
are:

  YEAR         year value as integer
  MONTH        month value as integer
  DAY          day value as integer
  HOUR         hour value as integer
  MINUTE       minute value as integer
  SECONDS      seconds value as real value
  JULIAN       Julian date with fractional day (e.g. 2457343.227256944)
  MJD          Modified Julian Date (e.g. 57342.72725694445)
  ISO          ISO date (e.g. 2015-11-16T17:27:15.000)
  GAG_DATE     GAG date integer format (e.g. -3207)
  YYYYMMDD     date with all numbers (e.g. 20151116)
```

```
DD-MMM-YYYY  date where MMM is the month in letters (e.g. 16-NOV-2015)
RADIAN       time of day in radians (betwen 0 and 2*pi)
SEXAGESIMAL  time of day using sexagesimal notation (e.g. 17:27:15.000)
```

If input the date-time specification is an array, the output variable(s) must be arrays of the same size.

5.8 DEFINE

```
[SIC\]DEFINE Type Var1 [Keys...] [Var2 [Keys...] [...]] [/GLOBAL]
[/LIKE VarLike] [/TRIM [Rank]]
```

If Type is FUNCTION, define a user-function. If Type is COMMAND or LANGUAGE, define a new user command or language respectively. Otherwise, define new variables of the specified type. Type can then be REAL, INTEGER, DOUBLE (for double precision real values), LOGICAL, CHARACTER, or TABLE, HEADER, IMAGE, UVTABLE, STRUCTURE, FITS or ALIAS. By default, new variables are LOCAL, i.e. valid only within the current macro and all loops or interactive levels called by this macro. When the /GLOBAL switch is specified, the variables are valid at all levels. Local variables are examined before global variables.

5.8.1 DEFINE ALIAS

```
[SIC\]DEFINE ALIAS AliasName TargetVar [/GLOBAL]
```

Define a new variable AliasName pointing towards (a subset of) an existing variable. The new alias shares all the properties of its target: type, Local or Global status, Readonly attribute. The TargetVar can be a sub-array, but implicit transposition is not allowed.

If the /GLOBAL option is present, the TargetVar must be a Global variable.

Aliases can be deleted using the standard DELETE /VAR command: this does not delete the TargetVar variable. On the contrary, when a standard variable is deleted, all aliases attached to it are also deleted, since the variable content disappears.

Structures cannot have aliases, but structure members can.

5.8.2 DEFINE CHARACTER

```
[SIC\]DEFINE CHARACTER Var1*Lvar1[DIM1] [...]
[SIC\]DEFINE CHARACTER*Length Var1[DIM1] [...]
```

CHARACTER variables can be scalar or multi-dimensional. The length of CHARACTER variable is specified after an * either after CHARACTER keyword or after the name of the variable. When both are present, the per-

variable length has precedence. The dimension field (same syntax as for other arrays) should follow the length declaration. The CHARACTER keyword can be abbreviated, i.e. this is a valid statement

```
DEFIN CHAR*12 Var1  Var2*36 Var3[16]
```

Character substrings can be accessed for reading and writing with the [charmin:charmax] specification, e.g.

```
DEFINE CHARACTER C*128
LET C[3:4] "AB"
EXAMINE C[3:4]
IF C[3:4].EQ."AB"  SAY "Yes"
```

The substring specification MUST be provided with the column ":", even for a single character, e.g. C[1:1]. If the character variable has one or more dimensions, the substring specification can be applied to a scalar subarray, e.g.

```
DEFINE CHARACTER*128 D[3,4]
EXAMINE D[1,2][3:4]
```

It is an error to apply a substring specification to an array, e.g.

```
EXAMINE D[1,][3:4]  ! Invalid
```

5.8.3 DEFINE COMMAND

```
[SIC\]DEFINE COMMAND NAME "Equivalent command" [Help_File]
```

Define a new user-defined SIC command.

By default, the new command will be part of the USER\ language. USER\ language is implicitely created if required without invoking DEFINE LANGUAGE. If the command is prefixed by another user language, it will be part of it. Attempting to add a command of a program language is an error.

Help_File is an optional argument indicating the name of the associated help text to be used by the command HELP. The standard rules for help syntax applies (see SIC documentation for details). If no help file is provided, the one of the parent language will be used.

User defined commands are translated into their equivalent command at execution time. They appear in the list of command displayed by HELP, and can be abbreviated as normal "program-defined" commands. For example

```
DEFINE COMMAND INPUT "@ input.greg" pr:input_greg.hlp
```

define a new command USER\INPUT, which will execute procedure input.greg.

5.8.4 DEFINE DOUBLE

```
[SIC\]DEFINE DOUBLE Var1[DIM1] [Var2[DIM2] [...]] [/GLOBAL]
```

DOUBLE variables can be multi-dimensional. Up to 7 dimensions can be specified in the optional DIM field, with the following syntax:

```
    Var[n1] or Var[n1,n2] etc... up to Var[n1,n2,n3,n4,n5,n6,n7]
```

where Var is the variable name and n1 to n7 are integer constants or variables.

5.8.5 DEFINE FITS

```
[SIC\]DEFINE FITS Var File [HEADER] [ [index] [T] [B] ]  [/GLOBAL]
```

Define variables associated to the keywords and data of a FITS file. The defined variables depend on the FITS file content. Both "basic" FITS files and FITS extension (XTENSION) are handled.

The HEADER keyword indicates to read only the FITS Headers (which includes any Extension), not the main data array.

An 'index' value may be added to read only the 'index'-th extension.

With 'B' code (stands for Basic): define only the basic keywords, not proprietary keywords. In particular suppress HIERARCH keyword structure present in ESO FITS headers.

The 'T' code may be added to get all multidimensional arrays transposed (depending on how the FITS file was written, sometimes the dimensionality of the SIC variables created from the FITS structure is not handy. Using the transposition code can be a solution then).

The 'index', 'T' and 'B' codes can appear in any order.

For basic FITS data, the following variables are defined

VAR%NDIM	Integer	Number of dimensions
VAR%DIM	Integer[4]	Dimensions
VAR%CONVERT	Double[3,4]	Conversion formulae for the 4 axes: Reference pixel, Value at reference pixel, Increment
VAR%DATA	Real	FITS data array

The variables are defined as part of a structure. GreG command LIMITS /RGDATA A recognizes such a basic FITS structure in much the same way as GILDAS images.

The DEFINE FITS command tries to define a SIC structure which contains all FITS keywords, as well as all binary and ASCII tables located in FITS extensions.

Support for Random groups (although this is an obsolescent FITS structure, it is still widely used) is also available.

DEFINE FITS only works to read FITS files, but not to create them. See VECTOR\FITS to write FITS files (and also read them, but only the main array so far).

5.8.6 DEFINE FUNCTION

```
[SIC\]DEFINE FUNCTION NAME(X,Y,Z) Expression(X,Y,Z)
```

Define an arithmetic user function of several variables. The maximum number of variables is 8. The function definition can reference any of the known mathematic operators and intrinsic or program defined functions, but not previously defined user functions (i.e. user function definition is not recursive).

5.8.7 DEFINE HEADER

```
[SIC\]DEFINE HEADER Var1 File1 Key1 [Var2 File2 Key2 [...]]
            [/GLOBAL] [/TRIM [Rank]] [/LIKE OtherVar]
```

Define variables associated to the HEADER of the GILDAS images located in the files specified by File1, File2, etc... The keywords Key1, Key2, etc... must be either READ or WRITE.

FileJ can be * to define a virtual header. In this case, KeyJ must be IMAGE or UVDATA. VarJ must include the dimensions, or the /LIKE option should be present to derive them from an existing image.

The following variables are defined:

VAR%GENE	Integer	Length of general section
VAR%NDIM	Integer	Number of dimensions (ReadOnly)
VAR%DIM	Integer[4]	Dimensions (ReadOnly)
VAR%CONVERT	Double[3,4]	Conversion formulae for the 4 axes: Reference pixel, Value at reference pixel, Increment
VAR%BLAN	Integer	Length of blanking section
VAR%BLANK	Real[2]	Blanking and tolerance
VAR%EXTREMA	Integer	Length of extrema section
VAR%MAX	Real	Maximum
VAR%MIN	Real	Minimum
VAR%WHERE	Integer[4,2]	Position of max and min
VAR%DESC	Integer	Length of units and coordinate system secti
VAR%UNIT	Char*12	Image unit
VAR%UNIT1	Char*12	First axis type
VAR%UNIT2	Char*12	Second axis type
VAR%UNIT3	Char*12	Third axis type
VAR%UNIT4	Char*12	Fourth axis type

VAR%SYSTEM	Char*12	Coordinate system
VAR%POSI	Integer	Length of position section
VAR%SOURCE	Char*12	Source name
VAR%RA	Double	Right Ascension
VAR%DEC	Double	Declination
VAR%LII	Double	Galactic longitude
VAR%BII	Double	Galactic latitude
VAR%EQUINOX	Real	Equinox of coordinates
VAR%PROJ	Integer	Length of projection section
VAR%PTYPE	Integer	Projection type (code)
VAR%A0	Double	First coordinate of projection center
VAR%D0	Double	Second coordinate of projection center
VAR%ANGLE	Double	Position angle of projection
VAR%X_AXIS	Integer	First projected axis
VAR%Y_AXIS	Integer	Second projected axis
VAR%SPEC	Integer	Length of spectroscopy section
VAR%LINE	Char*12	Line name
VAR%FREQRES	Double	Frequency resolution
VAR%IMAGFRE	Double	Image Frequency
VAR%RESTFRE	Double	Rest Frequency
VAR%VELRES	Real	Velocity resolution
VAR%VELOFF	Real	Velocity offset
VAR%F_AXIS	Integer	Frequency/Velocity axis
VAR%BEAM	Integer	Length of beam section
VAR%MAJOR	Real	Major axis of beam
VAR%MINOR	Real	Minor axis of beam
VAR%PA	Real	Position angle of beam

where VAR is the specified variable name for the header. VAR becomes a dummy variable of type header, which can only be referenced in a further DELETE /VARIABLE command, in structure assignments through command LET, or in the HEADER command. The VAR%item variables are ReadOnly if the keyword Key is READ, ReadWrite otherwise (including of course virtual headers), except for the dimension variables which cannot be modified.

5.8.8 DEFINE IMAGE

```
[SIC\]DEFINE IMAGE Var1 File1 Key1 [Var2 File2 Key2 [...]]
          [/GLOBAL] [/TRIM [Rank]]
```

Define variables associated to both the content and the header of the GILDAS images located in the files, if specified. This command acts as a combination of DEFINE HEADER and DEFINE TABLE. It accepts the same keywords (KeyN) as DEFINE TABLE. If the keyword is READ, the header variables are mapped ReadOnly, otherwise they are mapped ReadWrite.

See HELP DEFINE HEADER for a description of the individual header variables.

FITS cubes can also be loaded as a Gildas image variable, using the FITS file name in place of the file name. There will be no disk-to-disk translation: at load time, the FITS header will be translated on-the-fly to the GDF header elements, and the data will also be converted on-the-fly if needed.
This feature is only available for cube-like data in Primary HDU. To load other HDUs, use disk-to-disk conversion with the command V\FITS /HDU.

5.8.9 DEFINE INTEGER

```
[SIC\]DEFINE INTEGER Var1[DIM1] [Var2[DIM2] [...]] [/GLOBAL]
```

INTEGER stands for SHORT (I*4) or LONG (I*8) integers depending on the current SIC INTEGER rule.

INTEGER variables can be multi-dimensional. Up to 7 dimensions can be specified in the optional DIM field, with the following syntax:
Var[n1] or Var[n1,n2] etc... up to Var[n1,n2,n3,n4,n5,n6,n7]
where Var is the variable name and n1 to n7 are integer constants or variables.

5.8.10 DEFINE LANGUAGE

```
[SIC\]DEFINE LANGUAGE Name [Help_File]
```

Define a new user language. User languages can be filled with user commands with DEFINE COMMAND. Their precedence against other languages is set by default to Automatic, use SIC PRIORITY afterwards to change it.

If an help file is provided, the command HELP will look into this file for the language help (see SIC documentation for details). Else it will display a summary of all the language commands and their translations.

5.8.11 DEFINE LOGICAL

```
[SIC\]DEFINE LOGICAL Var1[DIM1] Var2[DIM2] [...] [/GLOBAL]
```

LOGICAL variables can be multi-dimensional. Up to 7 dimensions can be specified in the optional DIM field, with the following syntax:
Var[n1] or Var[n1,n2] etc... up to Var[n1,n2,n3,n4,n5,n6,n7]
where Var is the variable name and n1 to n7 are integer constants or variables.

5.8.12 DEFINE REAL

```
[SIC\]DEFINE REAL Var1[DIM1] [Var2[DIM2] [...]] [/GLOBAL]
```

REAL variables can be multi-dimensional. Up to 7 dimensions can be specified in the optional DIM field, with the following syntax:

    Var[n1] or Var[n1,n2] etc... up to Var[n1,n2,n3,n4,n5,n6,n7]

where Var is the variable name and n1 to n7 are integer constants or variables.

5.8.13 DEFINE STRUCTURE

    [SIC\]DEFINE STRUCTURE Str [/GLOBAL] [/LIKE MoldStr]

Define a new structure name. Structure naming follows the Fortran-90 convention, i.e. Str%SubStr%SubStrElement. Structure elements (including sub-structures if needed) can be defined using command

    DEFINE REAL Str%Element

etc. The %DATA element field is reserved to handle the N-Dimensional data area associated to the structure, allowing Structure to mimick Images if needed.

An entire structure and all its associated members is deleted by a single command DELETE Str /VARIABLE. A Structure can be defined like another one using the /LIKE option.

5.8.14 DEFINE TABLE

    [SIC\]DEFINE TABLE Var1 File1 Key1 [Var2 File2 Key2 [...]] [/GLOBAL]

Define variables associated to GILDAS images located in the files, if specified. The variable type and dimensions are derived according to the value of the keyword Key and the file content:

- READ or WRITE: use the type and dimensions from the file, and connect the image in Readonly or ReadWrite access. The variable name must not include any dimension field.
- DOUBLE, INTEGER or REAL: create a new table of the specified type. The dimensions must then be specified in the dimension field of the variable name, as for a standard variable. The associated file is created, unless a star ('*') is used as file name.
- EXTEND: take type and first dimensions from the file, but extend the last dimension to the value specified in the dimension field. The full syntax in this case is thus

        DEFINE TABLE Var[Ldim] File EXTEND

  where Ldim is the new value for the last dimension.

DEFINE TABLE does not create any additional variables for the image header. See DEFINE IMAGE and DEFINE HEADER for this information.

5.8.15 DEFINE UVTABLE

    [SIC\]DEFINE UVTABLE Var1 File1 Key1 [Var2 File2 Key2 [...]] [/GLOB-

```
AL]

Define variables associated to both the content and the  header  of  the
GILDAS UV Table located in the specified files. The files must be UV Ta-
bles, a version of 2-D tables with some conventions for the  interpreta-
tion of columns and an image-like header.

When creating a new file, if the file name has the extension ".tuv", the
command creates a TUV table  (transposed  UVT)  with  the  corresponding
header elements transposed.

See  HELP DEFINE HEADER for a description of the individual header vari-
ables.
```

5.8.16 DEFINE /GLOBAL

```
[SIC\]DEFINE Type Var1 [Var2 [...]] /GLOBAL

The /GLOBAL option specifies that the variables are global,  instead  of
being  local  to  the current procedure. Local variables have precedence
over global variables.

The /GLOBAL option is incompatible with DEFINE FUNCTION and DEFINE  COM-
MAND.
```

5.8.17 DEFINE /LIKE

```
[SIC\]DEFINE Type Var1 [Var2 [...]] /LIKE VarLike
[SIC\]DEFINE IMAGE Var1 File1 Key1 [...]] /LIKE VarLike
[SIC\]DEFINE STRUCTURE Struct1 [Struct2 [...]] /LIKE StructLike

The  /LIKE option allows definition of Type REAL, LOGICAL, DOUBLE or IN-
TEGER arrays with dimensions identical to those of the  (existing)  Var-
Like  array.  IMAGEs and TABLEs can also be defined in this way. The di-
mension field must not be specified in such a case.

/LIKE can also be used in structure context: the new structure  will  be
filled with a tree identical to the StructLike one (which can be a HEAD-
ER or an IMAGE). However, the %DATA area of Structures and Images is ig-
nored in this process, and should be defined separately if needed.

The  /LIKE  option is incompatible with DEFINE FUNCTION, DEFINE COMMAND,
as well as DEFINE HEADER and DEFINE FITS.
```

5.8.18 DEFINE /TRIM

```
[SIC\]DEFINE IMAGE|HEADER Var1 File1 Key1 [Var2 File2 Key2 [...]]
          /TRIM [Rank] [/GLOBAL]
```

Attempt to trim trailing degenerate dimensions up to the specified Rank. If no Rank is given, trim all possible trailing degenerate dimensions.

If Rank is positive, the defined header or image will have exactly Rank dimensions. An error occur if the corresponding file has non-degenerate dimensions beyond this rank. If the file is of lower dimensionality than the specified Rank, its dimensionality is extended by adding degenerate dimensions up to Rank.

If Rank is negative, trim all possible trailing degenerate dimensions, and return an error if the final dimensionality remains larger than Rank.

Examples:

```
  Assume file.gdf contains a 3-D array of dimensions [Nx,Ny,1]
  DEFINE IMAG A file.gdf READ
    will return the 3-D array A[Nx,Ny,1]
  DEFINE IMAG A file.gdf READ /TRIM 4
    will return the 4-D array A[Nx,Ny,1,1]
  DEFINE IMAG A file.gdf READ /TRIM 1
    will produce an error (same with /TRIM -1)
  DEFINE IMAG A file.gdf READ /TRIM -4
    will return the 2-D array A[Nx,Ny]
  DEFINE IMAG A file.gdf READ /TRIM
    will also return the 2-D array A[Nx,Ny]
```

5.9 DELETE

```
[SIC\]DELETE /VARIABLE|/SYMBOL|/FUNCTION Name1 [Name2 [...]]
```

Delete specified variables, symbols or functions. For IMAGE variables, also frees the corresponding file.

5.10 DIFF

```
[SIC\]DIFF Var1|File1 Var2|File2
```

List the differences between two entities. Each entity can be either a Sic variable name or a GDF file name (images, cubes, UV tables,...). Comparison is made on their headers (if both variables provide a header) and/or their data (if both variables provide a data array).

The result of the last comparison (true/false) is saved in the logical variable SIC%DIFF.

5.11 EDIT

```
[SIC\]EDIT [File_Name]
```

Without argument, EDIT dumps the Stack on a file named STACK.DEFEXT where DEFEXT is the default macro extension specified by the program or by the user using command SIC\SIC EXTENSION, and then calls a text editor to edit this file. If a file name is given, the specified file is edited.

The editor to be chosen is defined by the logical name GAG_EDIT, which you can define in your $HOME/.gag.dico file. Command SIC EDIT NewEditor can also be used to re-define the choice of the editor.

5.12 ELSE

[SIC\]ELSE [IF Logical_expression [THEN]]

Conditional directive in an IF-END IF block. Similar to Fortran ELSE and ELSEIF statements, but note that here the space between ELSE and IF is compulsory, while the THEN keyword is optional. If the compulsory space bothers you, just define the following symbol

SYMBOL ELSEIF "SIC\ELSE IF"

5.13 END

[SIC\]END Procedure|Help|Data|If

[SIC\]END IF

Mark the end of an IF block. Normal execution resumes. The symbol ENDIF is defined as an abbreviation of SIC\END IF.

[SIC\]END Procedure|Help|Data

Terminate the definition of a new Procedure, Help file, or Data file. See command BEGIN for details.

5.14 EXECUTE

[SIC\]EXECUTE CommandString

Execute the command line given as argument. It can be an explicit character string, a string variable contents, or a mixture of both. For example:

```
! Simple command
SIC> SIC\EXECUTE EXAMINE

! Command with several words
SIC> SIC\EXECUTE "EXAMINE /GLOBAL"

! Command stored in variable
```

```
SIC> DEFINE CHARACTER*32 COMMAND
SIC> LET COMMAND "EXAMINE /GLOBAL"
SIC> SIC\EXECUTE 'COMMAND'

! Command stored in several variables
SIC> DEFINE CHARACTER*32 COMMAND OPTION
SIC> LET COMMAND "EXAMINE"
SIC> LET OPTION "/GLOBAL"
SIC> SIC\EXECUTE 'COMMAND'" "'OPTION'
```

The error status of the executed command is propagated to the EXECUTE command, except for @ (errors raised by @ are lost).

5.15 EXAMINE

[SIC\]EXAMINE [Name1] [Name2] [...] [NameN] [/GLOBAL] [/HEADER] [/ADDRESS] [/ALIAS] [/PAGE] [/SAVE]

List variables.

EXAMINE without argument gives the list of known variables with their respective types. Each variable name is followed by its definition and the keywords "GBL" for GLOBAL (interpreter level 0) variables, LCL(lev) for LOCAL variables at interpretor level "lev" (viz., in a procedure), WR for READ/WRITE variables, RD for READONLY variables. The option /PAGE can be used to examine long outputs page by page.

With one or more arguments, EXAMINE will usually type the content of the specified variables (local variables have priority). But:

- If the argument ends by the % character, it is assumed to indicate a structure or an image header, and all associated header variables are listed.
- Wildcarding is permitted: if the argument contains one or more "*" character, all compatible variables names are listed.
- In any case, typing an ambiguous variable name will generate a list of choices.

With the /GLOBAL option, lists only the GLOBAL variables, as opposed to LOCAL variables (i.e, in a loop or a macro)

With the /HEADER option, lists only the variables having an associated header (i.e., the known images).

With the /ALIAS option, lists only the Aliases and their association.

The /ADDRESS option is used for debugging. It gives the internal variable descriptor.

5.15.1 EXAMINE /SAVE

```
[SIC\]EXAMINE Variable /SAVE OutputFile
```

THIS OPTION IS EXPERIMENTAL AND MIGHT BE REMOVED OR RENAMED IN THE FUTURE WITHOUT ADVISE.

The output of the command EXAMINE is written in the named file instead of terminal. The output is specially formatted under the form

```
  Variable = Value
or
  Array = Val1 Val2 Val3  ! i.e. flatened array
```

This way, the output file can be replayed in Sic as a procedure in a save-and-reload process. This makes most sense when examining structures with the syntax EXA MYSTRUCT% /SAVE MYOUTPUT.

5.16 EXIT

```
[SIC\]EXIT [Code] [/NOPROMPT]
```

EXIT always ends SIC execution, at any level. An error status can be set on return to the shell by passing an integer code (default 0). In interactive sessions, <^D> has the same behaviour as EXIT.

On the opposite <^Z> only ends the current level of execution, so that when typed at the top level, <^Z> also ends the session. Note that <^Z> may be trapped when interactive editing is enabled.

The option /NOPROMPT will disable any prompt at exit time, for programs (or their dependencies) which prompt for an answer before leaving.

Note that end of SIC execution does not necessarily means end of program execution: SIC may be entered again later by the calling program.

5.17 FOR

```
[SIC\]FOR Loop_Variable n1 n2 n3 TO n4 BY n5 n6 TO n7
[SIC\]FOR Loop_Variable /IN List_Variable
[SIC\]FOR /WHILE Logical_Expression
```

Open a FOR-NEXT loop to be executed for real values in the list (FOR Variable), or any values (including strings) given in the /IN option,
or until a logical expression becomes false (FOR /WHILE).

Up to 9 loops may be nested in any way. The loop can be exited at any time with the command BREAK (see HELP BREAK for details).

If the loop is entered interactively at the prompt, the last character of the prompt becomes a ':' instead of '>', and all subsequent commands typed in until NEXT are the body of the loop.

* Indexed loops: FOR Loop_Variable n1 n2 n3 TO n4 BY n5 n6 TO n7
  See HELP FOR Indexed

* Generalized loops: FOR Loop_Variable /IN List
  See HELP FOR /IN

* Conditional loops: FOR /WHILE Logical_Expression
  See HELP FOR /WHILE

The Loopriable is the name of a SIC variable created by the FOR command and must not be an existing variable name.

5.17.1 FOR Indexed

```
[SIC\]FOR Loop_Variable n1 n2 n3 TO n4 BY n5 n6 TO n7
```

The loop specified in this example will execute for the following values of the index :

```
n1
n2
n3, n3 + n5, n3 + 2 n5, n3 + 3 n5, ... , n4
n6, n6 + 1, n6 + 2, ... , n7
```

(assuming that n4-n3 is a multiple of n5, n7-n6 an integer).

Loop variables are undefined outside the loop, and must not be previously defined. Here, the loop variables are Double Precision numbers, and non-integer start, end, and step can be used. Invalid ranges are ignored at execution time: e.g. in the previous example, the loop would not be executed for n3 TO n4 BY n5 if n5*(n4-n3) < 0, but no error is signaled. Increments of 0 produce an error. The index of the loop will be substituted to the loop variable Loop_Variable during loop execution.

5.17.2 FOR /IN

```
[SIC\]FOR Loop_Variable /IN List_Variable
```

Execute a generalized loop, for all the values in List_Variable. List_Variable is a 1-D (or scalar) variable of any type, i.e. numerics, logicals, or character strings. The loop variable is undefined outside the loop, and must not be previously defined. It will be implicitly defined as a scalar with the same type and kind as the List_Variable. For example:

```
DEFINE CHARACTER*8 files[3]
LET files "a.txt" "b.txt" "c.txt"
FOR f /IN files
  SAY 'f'
NEXT f
```

The loop executes with the Loop_Variable taking the value of each element of the List_Variable, in order.

5.17.3 FOR /WHILE

```
[SIC\]FOR /WHILE Logical_Expression
```

Conditional loops execute until the logical expression becomes false. For example FOR /WHILE .TRUE. will execute forever.

Such loops have no associated loop variable.

5.18 HELP

```
[SIC\]HELP [FUNCTION|TASK|RUN|GO] [Topic [Subtopic|*]]
```

Display the help of various Gildas tools. There may be two levels of help available. In this case, a wildcard will display all the subtopic helps at once. You can use the switch SIC HELP to customize the way long helps are displayed (see HELP SIC HELP).

HELP
  Without an argument, HELP gives the list of available languages and commands.

HELP LangName\
  With a language name followed by a backslash (e.g. HELP SIC\), HELP gives a one line description of all commands specific to this language.

HELP Command [Subtopic]
  With a command name, the help for this command is displayed. Ambiguities can be fixed by prefixing the command by its parent language (e.g. HELP SIC\DEFINE). A subtopic (e.g. an option name) can be given as second argument.

  Subtopics which appear in capital letters when listed by HELP Command are not case-sensitive, while Subtopics appearing in mixed-cases are case-sensitive.

HELP FUNCTION [Name]

With the keyword FUNCTION alone, it gives a list of all functions available. With a function name added, it gives a detailed help for this function. There are 3 kind of functions: 1) functions with native support in the SIC interpreter, 2) program-defined functions (specific to each program), 3) user-defined functions (defined with DEFINE FUNCTION).

HELP TASK [Group|?]
With the keyword TASK, it gives a one line description of all available tasks. A task group name can be given as second to restrict the list to this group. Type HELP TASK ? for the list of available groups.

HELP RUN TaskName [VarName]
With the keyword RUN followed by a task name, the help for this task is displayed.

HELP GO ProcName [Subtopic]
With the keyword GO followed by a procedure name (lower case sensitive), the help for this procedure is displayed.

HELP SIC SYNTAX
Get some basic help on the SIC interpreter syntax.

5.19 IF

```
[SIC\]IF Logical_Expression [THEN]
[SIC\]IF Logical_Expression Command [Arguments]
```

The first form starts a conditional IF block. The behaviour of IF blocks is similar to structured Fortran. The THEN keyword is optional.
The second form defines a logical IF statement, i.e. it uses a single line with no possibility of alternate execution (no ELSE choice). When the argument following the logical expression is not THEN, it is assumed to be a command which will be interpreted, together with the next arguments, if the logical expression evaluates as true.

This command can only be used within a procedure: interactive users are expected to be able to make their decisions themselves.

The logical expression must be a single argument (no blanks); composite expressions using .AND. or .OR. operators are supported. Logical functions like EXIST, FILE, or FUNCTION can also be used (see HELP FUNCTION for details).

5.20 IMPORT

```
[SIC\]IMPORT Package
```

Dynamically import a package (i.e. load its languages and perform its initialization) in current scope. This allows to load another package without closing (and loosing) the current session.

This feature is currently only available for: ASTRO, CLASS and MAPPING.

5.21 LET

```
[SIC\]LET Variable [=] [Expression]
[SIC\]LET Variable [=] Expression /CHOICE Value_1 ... Value_n
[SIC\]LET Variable [=] Expression /FILE Filter
[SIC\]LET Variable [=] Expression /FORMAT format_string
[SIC\]LET Variable [=] Expression /FORMULA
[SIC\]LET Variable [=] Expression /INDEX Value_1 ... Value_n
[SIC\]LET Variable [=] Expression /LOWER
[SIC\]LET Variable [=] Expression /NEW Type [Attr]
[SIC\]LET Variable [=] Expression /PROMPT "Explanatory text"
[SIC\]LET Variable [=] Expression /RANGE Min Max
[SIC\]LET Variable [=] OldVariable /REPLACE
[SIC\]LET Variable [=] Value_1 ... Value_n /RESIZE
[SIC\]LET Variable [=] Expression /SEXAGESIMAL
[SIC\]LET Variable [=] Val1 ... ValN /RESIZE
[SIC\]LET Variable /STATUS Read|Write
[SIC\]LET Variable [=] Expression /UPPER
[SIC\]LET Variable [=] Expression /WHERE Condition_mask
```

Assign a value to a variable. The variable must already be defined (see HELP DEFINE) unless the /NEW option is present. Logical expression results cannot be assigned to non logical variables, and vice versa. All numerical expression evaluations are done in double precision, and automatically converted to the type of (numerical) variable assigned.

The equal sign may be always omitted EXCEPT when using Free Syntax (see HELP LET Free_Syntax).

If no value is assigned to the Variable, the user will be prompted for the variable value. The Prompt text can be defined with the /PROMPT Option.

VECTOR OPERATION:

Operations are vectorial, i.e. a full array is computed at the same time. The variable name can define a subset of an known array, such as:

```
DEFINE REAL A[4,5,6] B[4]
LET A[,,3]  = 1.0  ! or equivalently  LET A[3] = 1.0
LET A[,2,2] = B    ! or equivalently  LET A[2,2] = B
```

These commands assign the value 1.0 to A[i,j,3], with i running from 1 to 4 and j from 1 to 5, and B[k] to A[k,2,2] for k from 1 to 4. Implic-

it transposition is now allowed, though still somewhat experimental: both A[,2,3] and A[2,,3] are valid. A range of indexes can be specified rather than one index: for the above example A[3:5] is a valid 4x5x3 array, A[2:4,,] is a 3x5x6 array.

IMPLICIT LOOPS:

In addition to vector computing, it is possible to assign an array using "implicit loops", i.e. functions of the array indices such as

```
    DEFINE REAL A[4,5]
    LET A[I,J] = (I-J)**2
```

Implicit loops cannot be mixed with variable index values.

CONDITION MASK (/WHERE option):

Vector assignment can be done only where a specified logical array (or logical array expression) is true, using the /WHERE option. See HELP LET /WHERE.

5.21.1 LET Free_Syntax

The LET command may be omitted if no option is present, and if the SIC syntax is set to FREE. In this case, the = sign is mandatory. In FIXED syntax, the LET command is compulsory, but the = sign optional. In FREE syntax, commands lines like

```
       A[I,J] = SIN((2*I+J)/PI)
```

where A is a known variable, are recognized as assignement and automatically expanded to the equivalent FIXED syntax

```
       LET A[I,J] SIN((2*I+J)/PI)
```

In case of conflict between a variable name and a (complete) command name, an error message is issued. Free syntax cannot be used for the LET /WHERE command. Command SIC SYNTAX [Free|Fixed] can be used to toggle between Fixed and Free syntax if needed.

5.21.2 LET GUI_Widget

GUI (Graphic-User-Interface) input mode:

If command GUI\PANEL has been issued before, the LET command defines widgets in the master window defined by GUI\PANEL. The widget is a simple prompt when option /PROMPT is set, a slider if option /RANGE is present, a list of choices if option /CHOICE or /INDEX is given, and a selection of files with the specified filter when option /FILE is specified. If neither of these options is present, the LET command works in the usual way.

The widgets are created and activated by command GUI\GO. Standard input can be used, and pressing button GO will define all the variables as specified. If button ABORT is pressed instead, none of the variables are modified and an error is returned. Error handling is available.

Options related to the GUI Widget mode are: /CHOICE, /FILE, /FORMULA, /INDEX, /PROMPT and /RANGE

5.21.3 LET Structure

LET OutStructure% [=] InStructure%

Assign, in as much as possible, a structure to another one. The structures do not need to be identical: only matching structure elements are copied.

In addition, the DATA area of Structures are ignored in this process: if needed, they must be copied separately, e.g.
OutStructure%DATA = InStructure%DATA

5.21.4 LET /CHOICE

[SIC\]LET Variable [= Expression] /CHOICE Val1 ... ValN [*] [/PROMPT "Prompt Text]
[SIC\]LET Variable [= Expression] /CHOICE List [/PROMPT "Prompt Text]
[SIC\]LET Variable = Expression /CHOICE [/PROMPT "Prompt Text]
(GUI input mode)

Activate a choice widget which will return a value among the specified list. The widget will be created by command GUI\GO. This option is usually combined with a /PROMPT option. If the last specified value is an asterisk, the choice widget will be editable and any value will be valid.

The choice list can also be given as a character array variable.

If the /CHOICE has no argument, the Variable will be displayed, but cannot be modified. This mode can be useful to display control parameters in a menu, or variables which result from operations with other variables.

5.21.5 LET /FILE

[SIC\]LET Variable [= Expression] /FILE Filter [/PROMPT "Prompt

```
Text]
            (GUI input mode)
```

Activate a "file" widget to select a file according to the specified filter. The widget will be created by command GUI\GO. This option is usually combined with a /PROMPT option.

5.21.6 LET /FORMAT

```
[SIC\]LET Variable [=] Expression /FORMAT format_string
```

Use the specified format to convert the expression into a character string and assign it to the requested (character) variable. For example:

```
define character c*6
LET C acos(-1.0) /FORMAT F6.3
exa c
c = 3.142
```

5.21.7 LET /FORMULA

```
[SIC\]LET Variable [=] Expression /FORMULA [/other_option]
        (GUI input mode)
```

Use Expression as a formula in GUI input mode, rather than only the value of the expression. This option can be combined with any other GUI related option, in particular /PROMPT.

5.21.8 LET /INDEX

```
[SIC\]LET Variable [= Expression] /INDEX Text_1 Text_2 ... Text_n
        (GUI input mode)
```

Activate a choice widget which will return an integer value corresponding to the index of the selected text among the specified list. The widget will be created by command GUI\GO.

5.21.9 LET /LOWER

```
[SIC\]LET Variable [=] Expression /LOWER
```

Convert the output to lower case prior to assignment. To convert a variable, use e.g. for variable C

```
let c 'c' /lower
```

5.21.10 LET /NEW

```
[SIC\]LET Variable [=] Expression /NEW Type [Attr]
```

Define the variable and assign it at the same time. Type is the type of

the variable (INTEGER, REAL, DOUBLE, LOGICAL, or CHARACTER). INTEGER stands for SHORT or LONG integer depending on the current SIC INTEGER rule.

The optional Attr argument is used to specify whether the newly created variable is LOCAL (default) or GLOBAL. See also the SIC\DEFINE command.

5.21.11 LET /PROMPT

```
[SIC\]LET Variable [=] [Expression] /PROMPT "Explanatory text"
        (GUI input mode and Command line mode)
```

Create a text widget, with a specified prompt, to return a value or valid expression for the variable. The text widget can be pre-loaded with an expression or value(s). Standard input will be available once the widget is created by command GUI\GO.

5.21.12 LET /RANGE

```
[SIC\]LET Variable [=] [Expression] /RANGE Min Max
        (GUI input mode)
```

Activate a slider widget to return a value within the specified range. The widget will be created by command GUI\GO.

5.21.13 LET /REPLACE

```
[SIC\]LET NewVariable [=] OldVariable /REPLACE
```

Rename OldVariable into NewVariable. The content and location of the variable remains unchanged: only the name is modified. If the variable is a structure, its elements are renamed accordingly.

5.21.14 LET /RESIZE

```
[SIC\]LET Variable [=] Val1 ... ValN /RESIZE
```

Automatically resize the variable according to the number of values to be stored in the output variable. For example:

```
SIC> define integer a[2]
SIC> let a 1 2 3 /resize
W-LET,  Resizing array A to length 3
SIC> exa a
A               is an integer Array     of dimensions  3
            1           2           3
```

This feature makes sense when the array size is not known at the time the procedure in written, but have to be decided instead from unpredictable inputs at run time (e.g. user input in widgets). It applies

(only) to 1D arrays of any kind (numeric, logical, character). The type, kind, rank and level (global/local) of the output variable is not modified during the process. Read-only and program-defined variables can not be resized.

5.21.15 LET /SEXAGESIMAL

```
    [SIC\]LET Numeric [=] [+|-]DD[:MM[:SS.SS]] /SEXAGESIMAL [D|H|R]
[D|H]
    [SIC\]LET String [=] Value /SEXAGESIMAL [D|H] [D|H|R]
```

The /SEXAGESIMAL option allows to use sexagesimal expressions in assignments. This option is only valid for scalar variables. Units of the operands can be provided by optional characters chosen in D)egrees, H)ours or R)adian. The first unit is for the output, the second for the input. Radian is an invalid unit for the sexagesimal string. Degrees are assumed by default for both operands.

The first form converts the sexagesimal expression into a numerical value stored in the assigned variable. Mathematical expressions can be used for each elements, provided they are separated by a semicolon.

The second form converts a numerical value into the sexagesimal notation stored in the assigned character variable.

5.21.16 LET /STATUS

```
    [SIC\]LET Variable /STATUS Read|Write
```

Modify a variable status to/from ReadOnly from/to Writeable.

5.21.17 LET /UPPER

```
    [SIC\]LET Variable [=] Expression /UPPER
```

Convert the output to upper case prior to assignment. To convert a variable, use e.g. for variable C

```
   let c 'c' /upper
```

5.21.18 LET /WHERE

```
    [SIC\]LET Variable = Expression /WHERE Condition_mask
```

The LET command allows setting variables only where a given condition mask is .TRUE.. The condition mask can be a logical array or a logical expression (of same dimension as the result variable). Implicit loops can be used in conjunction to the /WHERE option.

For example

```
   DEFINE REAL A[4,5] B[4,5]
   LET B[I,J] = I+J
   LET A[I,J] = (I-J)**2 /WHERE COS(I).GT.SIN(J)  ! 1
   LET A[I,J] = SIN(I+J) /WHERE B.GT.5            ! 2
```

is equivalent (in terms of results, but about 500 times faster) to the loops

```
   DEFINE REAL A[4,5] B[4,5]
   LET B[I,J] = I+J
! 1
   FOR J 1 to 5
     FOR I 1 to 4
       IF (COS(I).GT.SIN(J)) THEN
         LET A[I,J] = (I-J)**2
       ENDIF
     NEXT
   NEXT
! 2
   FOR J 1 to 5
     FOR I 1 to 4
       IF (B[I,J].GT.5) THEN
         LET A[I,J] = SIN(I+J)
       ENDIF
     NEXT
   NEXT
```

Note in the example above that implicit variables can be used. However, the following syntax is non valid,

```
   LET A[I,J] = SIN(I+J) /WHERE B[I,J].GT.5
```

because implicit variables cannot appear as indexes to an operand array (B), but only as indexes to the result array (A) or as variables as in SIN(I+J). The correct syntax would be

```
   LET A[I,J] = SIN(I+J) /WHERE B.GT.5
```

The following syntax is also non valid

```
   LET A = B /WHERE B[I,J].GT.LOG(I+J)
```

because implicit variables cannot be defined by an operand array (B), but only by the result array. The correct syntax would be

```
   LET A[I,J] = B /WHERE B.GT.LOG(I+J)
```

5.22 MESSAGE

```
    [SIC\]MESSAGE Severity ProcedureName "Message"
    [SIC\]MESSAGE Severity ProcedureName Arg1 Arg2 ... ArgN [/FORMAT
Fmt1 Fmt2 ... FmtN]
```

Print a message using the same mechanism as the one used by Gildas programs. In particular, it will be printed or not, to the terminal and/or to the message file, depending on the filters that currently apply to messages.

- The severity must be a single letter from one of the following: F)atal, E)rror, W)arning, R)esult, I)nfo, D)ebug, T)race, C)ommand, U)nknown.
- The procedure name is a free character string.
- The message must have at least one component. Multiple remaining arguments will be printed out spaced by a single space. Enclose with double-quote a character string if you want to use more than one space.
- If /FORMAT option is invoked, each argument is displayed using its associated format. Formats are Fortran ones (so may be slightly machine-dependent) like a10, i2, f5.2 and so on.

Example:
```
SIC> MESSAGE I FOO "Hello world!"
I-FOO,  Hello world!
```

For more information on message filters, see the HELP for the SIC MESSAGE command.

5.23 MFIT

[SIC\]MFIT Yvar=Func(Xvar,&A,&B,...) [/START A1 B1 ...] [/STEP A2 B2 ...] [/EPSILON e] [/WEIGHTS w] [/METHOD m] [/QUIET]
or
[SIC\]MFIT Filename [/START A1 B1 ...] [/STEP A2 B2 ...] [/EPSILON e] [/WEIGHTS w] [/METHOD m] [/QUIET]

Perform a least squares fit using the specified Method. The least square fit tries to adjust function Func of variable Xvar and parameters &A, &B, ... to match variable Yvar.

Any function or combination of functions known by SIC may be used. Blanks are supported in the formula. The '=' sign is required.

Fit parameter DUMMY names are &A...&Z . Already defined SIC variables may be used in the formula as long as they don't match the variables used by MFIT. The resulting parameters will be stored in the structure variable MFIT%. An array with weights may be given by option /WEIGHTS. It must match the dimension of the formula result (defaulted to uniform weight).

The formula can also be provided by a disk file (default extension .GRF), where lines beginning with an exclamation mark are treated as comments. Inside the file, blanks can be used in the formula, which may be spread over several lines.

The command MFIT defines the following global SIC variables to hold its results:

MFIT%PAR : Fit parameters

```
    MFIT%ERRORS : The parameter errors, if they have been computed
    MFIT%MATH   : Text string containing the formula with replaced dummies
    MFIT%FIT    : Found approximation (the application of MFIT%MATH)
    MFIT%RES    : Residuals  (i.e. Yvar-MFIT%FIT)
    MFIT%STATUS : .FALSE. on successful completion.
```

Suppose you read X,Y=f(X) and Z (errorbar on Y) in the 3 arrays X, Y and Z. You could change Z into weights (for example SIC\LET Z 1|Z^2), then fit a cubic polynomial regression in variables X,Y by typing:

```
    MFIT Y=(&A*X^3+&B*X^2+&C*X+&D) /WEIGHTS Z
```

The MFIT command will print the used formula:

```
    I-MFIT,  Formula (stored in variable MFIT%MATH) :
    (MFIT%PAR[01]*X^3+MFIT%PAR[02]*X^2+MFIT%PAR[03]*X+MFIT%PAR[04])
```

The array MFIT%PAR will contain MFIT%PAR[1]=value of '&A', and so on... to be used afterwards.

5.23.1 MFIT /EPSILON

This option is used to specify the desired tolerance. Its interpretation is method dependent. For SIMPLEX and POWELL, it means the relative deviation of the mean squared difference of two fit iterations. For SLATEC, which uses non-reduced chi-2, it is the absolute difference between two iterations. For ANNEAL, its interpretation is totally different.

Use default value 0 to let the program guess. In subtle cases, or to gain speed in case a coarse result is desired, use values around 1D-5 for SIMPLEX and POWELL, and of order 1 for SLATEC. For ANNEAL, see details in the /METHOD option.

Note that this is not the absolute error of the fit parameters.

5.23.2 MFIT /METHOD

ANNEAL Simulated annealing technique. This may require a very large number of function evaluation.

POWELL Gradient using the Powell method (see Press et al. for details).

SIMPLEX Classical Simplex amoeba search

ROBUST A combination of Simplex + Slatec, to be used with large initial steps.

SLATEC Modified Levenberg Marquardt method with adaptive steps, as implement in the Slatec library.

All methods require a proper choice of initial values and steps. The ANNEAL method is much more robust against poor guesses, but may require 10 to 100 times more function evaluations than any other.

5.23.3 MFIT /QUIET

Require MFIT to be silent (useful to avoid too many messages in loops).

5.23.4 MFIT /START

```
[SIC\]MFIT Y=f(X,&a,...) /START A1 B1 ...
```

A1 B1 ... are used to pass starting guesses for the parameters &A &B (defaulted to 1.0). The starting values should not be too far from a potential solution, otherwise the convergence may not be possible.

5.23.5 MFIT /STEP

```
[SIC\]MFIT Y=f(X,&a,...) /STEP A2 B2 ...
```

A2 B2 ... are used to pass the unity vectors (steps) for the iteration on parameters &A &B (defaulted to 0.25). Poor choice of initial steps may lead to non convergence. Too small steps will not help converging when one starts too far from the solution. Too large steps may lead to incorrect evaluation of the parameter errors. The optimal step for the determination of the errors is about the error bar, so that all parameters become dimensionless.

5.24 MODIFY

[SIC\]MODIFY VarName|GDFName [/FREQUENCY Linename RestFreq] [/VELOCITY Velocity] [/SPECUNIT FREQUENCY|VELOCITY]

Modify consistently the spectroscopic axis of a lmv cube or a Sic structure of type IMAGE, HEADER, or UVTABLE.

The spectroscopic axis can be changed to the alternate unit with the option /SPECUNIT. By default, the unit is unchanged.

5.25 NEXT

```
[SIC\]NEXT
```

Depending on the context, NEXT has one of two possible interpretations:

- If typed during loop definition, NEXT indicates the end of the definition and starts the execution.
- If encountered during loop execution, NEXT will skip all instructions remaining in the Loop Buffer and resumes the Loop execution at its first line for the next value of the index. This typically occurs as an error recovery (ON ERROR NEXT).

The commands will be echoed to the terminal if the VERIFY switch is ON.

5.26 ON

```
[SIC\]ON Status [Command]
```

Perform the Command when Status in encoutered. Only the ON ERROR case is available. See subtopic ERROR for details.

5.26.1 ON ERROR

```
[SIC\]ON ERROR [Command]
```

This command changes the current error recovery action.

By default, a PAUSE is automatically generated by any error during execution. Without argument, ON ERROR resets to this default. Use the SIC command (without argument) to display the current ON ERROR status.

The error recovery can be changed and any other valid command line Command can be activated instead of PAUSE when an error occurs, using the command ON ERROR Command. A very useful error recovery command for loops is NEXT, which enables skipping the remaining commands of the loop when something went wrong, and resumes the loop at for the following index value.

Note that the ON ERROR command does not affect the behaviour on <^C> trapping, which always returns a PAUSE. The ON ERROR command is only valid within the procedure which declared it.

5.27 PARSE

```
[SIC\]PARSE [/OptionName1 [Nargmin [Nargmax]]] ... [/OptionNameN ...]
```

Parse the array PRO%ARG in order to mimic the options mechanism provided by a standard command.

When a procedure is executed, the array PRO%ARG is filled with the arguments passed to the procedure. If you invoke the command PARSE with some option names and required number of arguments, it will parse the arguments to gather them by options and option arguments into the structure PRO%PARSE%. The command PARSE will check that the caller has provided known options and correct number of arguments, else an error will be raised.

For example, if you invoke your procedure like this:

```
SIC> @ myprocedure 123 /MYOPTION "ABC"
```

the PRO%ARG array will contain

```
SIC_3> EXA PRO%ARG
PRO%ARG    is a character*256 Array of dimensions 3
123
/MYOPTION
ABC
```

Invoking the command PARSE in your procedure e.g.

```
SIC_3> PARSE /MYOPTION /OTHEROPTION
```

will gather the command and the 2 option arguments like this:

```
SIC_3> EXA PRO%PARSE%
PRO%PARSE%                                    ! Structure GLOBAL
PRO%PARSE%COMMAND                             ! Structure GLOBAL
PRO%PARSE%COMMAND%NARG =            1         ! Integer GLOBAL RO
PRO%PARSE%COMMAND%ARG is a character*256 Array of dimensions 1
PRO%PARSE%MYOPTION                            ! Structure GLOBAL
PRO%PARSE%MYOPTION%NARG =            1        ! Integer GLOBAL RO
PRO%PARSE%MYOPTION%ARG is a character*256 Array of dimensions 1
PRO%PARSE%OTHEROPTION                         ! Structure GLOBAL
PRO%PARSE%OTHEROPTION%NARG =           -1     ! Integer GLOBAL RO
```

Namely: the command was passed 1 argument (available in the associated ARG array), the option MYOPTION was passed 1 argument, and the option OTHEROPTION was absent (NARG=-1). An option is present if NARG>=0.

You can control the number of required of arguments to the command and the options. Syntax is "/OptionName [Nargmin [Nargmax]]". If Nargmax is absent, it defaults to Nargmin. If Nargmin is absent, it means no specific constraint. For example:

```
SIC_3> PARSE 1 /MYOPTION 2 3 /OTHEROPTION
```

means that exactly 1 argument must be passed to the command (before the options), that at least 2 and at most 3 arguments are required for /MYOPTION, and that there is no check for /OTHEROPTION.

5.28 PAUSE

```
[SIC\]PAUSE "Message"
```

Set a break point in a Loop or a Macro. PAUSE returns control to user when executed in a macro. The prompt is changed to indicate the execution level. Any valid command can be executed while in interrupt mode. The execution can be resumed by typing CONTINUE, aborted by QUIT (or RETURN, depending whether an error status is desired). NEXT may also be used for the Loop.

If PAUSE is followed by an argument, this argument is typed when the

pause occurs. When typed in interactive, PAUSE generates a traceback of the levels of execution in SIC.

A pause is generated after completion of the current command if you type <^C> on the keyboard. This allows you to interrupt any sequence of commands (even if ON ERROR CONTINUE has been typed).

5.29 PYTHON

```
[SIC\]PYTHON
[SIC\]PYTHON PythonCommandLine
[SIC\]PYTHON PythonScript.py
```

Any form of the PYTHON command starts the Python interpreter and intercommunication between SIC and Python at first call. Subsequent call do not repeat this initialization step.

Three main interaction schemes are available:

- Without argument, the user is exposed to the PYTHON prompt.

- With a Python command line as argument (any string following the command), the command is transparently executed and the user recover the SIC prompt. Arguments starting with a slash (/) should be protected by double-quotes: this avoids SIC to interpret it as a command option. Remember also that Python is case sensitive and the string should take care of this. For example:

```
SIC> PYTHON print "Hello world!"
Hello world!
SIC> PYTHON def f(x): return x*x
SIC> PYTHON print f(2)
4
SIC>
```

- With a PythonScript.py as first argument (i.e. ending with ".py"), the Python script is executed and then the user recover the SIC prompt. Arguments following the script name can be recovered in the sys.argv Python list. These arguments are interpreted by SIC before being sent to the script. For example:

```
SIC> TYPE show_args.py
import sys
for i in sys.argv:
  print i, type(i)
SIC>
SIC> PYTHON show_args.py PI 'PI' 1.234
showrgs.py <type 'str'>
PI <type 'str'>
3.1415926535898 <type 'str'>
1.234 <type 'str'>
```

```
SIC>
```

5.30 QUIT

```
[SIC\]QUIT [ALL]
```

When typed after a pause, QUIT aborts the execution of the interrupted level. QUIT returns an error to the previous level to signal this anomalous end. The previous level is then also interrupted to allow a complete error recovery. Note that this behaviour depends on the ON ERROR command. QUIT can also be used to abort FOR-NEXT loop compilation.

If argument ALL is present, all nested procedures are aborted, and SIC goes back to base level.

In interactive sessions, QUIT should be used instead of BREAK and RETURN because it allows extra decision at run time. QUIT used as the error recovery command is equivalent to RETURN ERROR.

5.31 RECALL

```
[SIC\]RECALL [Arg]
```

This command retrieves command lines from the stack. The retrieved line is edited if possible or immediately executed if not. If no argument is present, the last command line is retrieved. If the argument is a number, the specified command is retrieved. If it is a character string, the first (most recently entered) command which begins by this string is recalled (in this case, the string may include a full language name or no language at all).

When line editing is possible, commands can also be retrieved using the Up arrow (or ^P) to recover the previous command, and Down arrow (or ^N) to recover the next command.

If no line editing is available, you are prompted whether to execute or not the recalled command.

5.32 RETURN

```
[SIC\]RETURN [BASE|ERROR]
```

End a procedure execution, transmitting or not an error if argument ERROR is present.

If argument BASE is present, RETURN will end all nested procedures, and give back control to base level, without transmitting any error.

5.33 SAY

[SIC\]SAY ["Text"] ['Expression'] ['Variable'] [/FORMAT ...] [/NOADVANCE]

Display strings or character variable or implicitly formatted arithmetic (or logical) expressions in the shortest possible format, unless the /FORMAT option is used.

Simple text:

Simple text can be displayed by passing double-quoted strings to the command, e.g.

```
SIC> say "Hello, world!"
Hello, world!
```

An unquoted argument will also be displayed "as is", like a simple string.

Variable:

The contents of a scalar variable can be displayed by evaluating it with single-quotes, e.g.

```
SIC> say 'pi'
3.1415926535898
```

Single quotes tell Sic to evaluate the variable and build a representating string showing its contents. Then this string is passed to SAY for display. Though Sic does its best to format the result, you can customize the format with the option /FORMAT. Array variables can not be displayed all at once using single quotes (but scalar subsets can be evaluated e.g. SAY 'A[1]'), use SAY /FORMAT to display arrays.

Expression:

Expressions can be evaluated using single quotes, e.g.

```
SIC> say '2*pi'
6.2831853071796
```

Like for variables, Sic evaluates the result and builds a representative string passed to SAY for display. Here also, the result must be scalar.

Multiple arguments:

SAY can display as many strings as there are arguments to the command. Remember that in Sic, arguments are separated by one or more spaces. Each argument can be of any kind explained above. For example:

```
SIC> say "PI ="   'pi'
PI = 3.1415926535898
```

Note that the number spaces separating the input arguments are not significant. In return, SAY will implicitely separate each string by one blank.

```
Concatenated arguments:
    Arguments can be concatenated by gluing any of the 3 basic kinds to-
    gether. For example:
        SIC> say "("'pi'")"
        (3.1415926535898)
    Note that there is no space between the components.  In  this  case,
    Sic  does  its  best to evaluate each component and produce a single
    temporary string concatenating all the  individual  representations.
    Then this formatted string is passed to the command SAY for display.
    Note that SAY sees only one argument (no blank = no separator => one
    argument).
    This syntax is not specific to SAY: it is a Sic feature which can be
    used in other commands. Be careful you have no control on the format
    choosen by Sic at concatenation time: /FORMAT is useless as it would
    affect only the single temporary string seen by SAY after formatting
    by Sic.

Output:
    By  default, SAY writes its output text on the terminal. However, it
    can be redirected to a file thanks to  the  command  SIC\SIC  OUTPUT
    (see HELP for details)..

GUI mode:
    In GUI mode (see command GUI\PANEL), SAY writes the text in the cur-
    rent window. SAY without parameters inserts a separator.
```

5.33.1 SAY /FORMAT

```
    [SIC\]SAY Arg1 Arg2 [... ArgN] /FORMAT fmt1 fmt2 [...fmtn]

where fmt1 applies to Arg1, and so on. The format used is a Fortran for-
mat (so it may be slightly machine-dependent). Use formats like a10, i2,
f5.2, and so on. For example:
    SIC> say '2*pi' /format F4.2
    6.28
Be careful that each argument must be separated by blanks (see "Multiple
arguments" vs "Concatenated arguments" in the main HELP SAY).

Formatting a scalar variable:
    Note also that the /FORMAT context, SAY is able to format a variable
    without a pre-evaluation by Sic. For example:
        SIC> say pi
        pi
        SIC> say 'pi'
        3.1415926535898
        SIC> say pi /format F4.2
        3.14
```

```
SIC> say 'pi' /format F4.2
3.14
```

1) In the first case, SAY considers unquoted arguments as simple strings to be displayed "as is".
2) In the second case, using single-quotes, Sic translates the variable contents to a text representation which is then given to SAY (SAY is not aware that a floating point variable is involved here).
3) In the third case, a floating point format is requested: this is an indication to SAY that the argument must be a numeric variable which has to be converted in a specific way. Here SAY knows it is dealing with a variable.
4) And finally, the last form should be avoided! The single-quotes are a request to Sic to translate the variable to a string using its best format (same as second case). Then this string is passed to SAY. As the /FORMAT option is present, SAY re-interprets the ASCII string into a floating-point value, and then this value is finally formatted to its final representation. In other words, the following sequence occured: read variable name -> get its floating-point value -> format as ASCII string -> read as floating-point value -> format as ASCII string. There are two useless steps, even if the result is correct as Sic avoids precision loss when representing numeric values.

Formatting an array variable:

If the argument is an array variable, the associated format is applied repeatedly to all its individual values:

```
SIC> define real a[4]
SIC> let a 1.2 3.4 5.6 7.8
SIC> say a /format f6.2
  1.20  3.40  5.60  7.80
SIC> say a /format 4(f6.2)  ! Equivalent
  1.20  3.40  5.60  7.80
```

Note however that the format can be used to display array values grouped per line, e.g.

```
SIC> say a /format 2(f6.2)  ! Group 2 values per line
  1.20  3.40
  5.60  7.80
```

Mixture of explicit values, evaluated expressions, scalar, and array variables:

```
SIC> define integer n
SIC> let n 4
SIC> define integer id[n]
SIC> let id[i] i
SIC> say "My interferometer has " n " antennas numbered" -
SIC-? id " providing " 'n*(n-1)/2' " baselines." -
```

```
SIC-? /format a i0 a i3 a i0 a
My interferometer has 4 antennas numbered  1  2  3  4 providing 6 ba
```

5.33.2 SAY /NOADVANCE

[SIC\]SAY Arg1 Arg2 [... ArgN] /FORMAT fmt1 fmt2 [...fmtn] /NOADVANCE

Do not put a Carriage Return at end. This allows to add several variables on the same output line in subsequent SAY commands. For example

```
SAY Arg1 /Format fmt1 /NOADVANCE
SAY Arg2 /Format fmt2
```

is equivalent to SAY Arg1 Arg2 /Format fmt1 fmt2

The /NOADVANCE option is only active with the /FORMAT option, and if the output (defined by SIC OUTPUT) is not the terminal.

5.34 SIC

[SIC\]SIC Arg1 Arg2 [... ArgN]

The SIC command has 5 categories of actions:

(1) File System Handling actions: (HELP SIC FileSystem)
These are used to manipulate files in a system independent ways from SIC procedures.
APPEND COPY DELETE DIRECTORY FIND GREP MKDIR MODIFIED PARSE RENAME

(2) Procedure related operations (HELP SIC Procedure)
EXPAND EXTENSION MACRO OUTPUT SAVE VERIFY WHICH

(3) Customization and status (HELP SIC Customize)
These are used to toggle some mode of the SIC monitor, or control some SIC monitor parameter. When called with no further argument, the status of the item will be displayed.
EDIT ERROR HELP INTEGER LOGICAL MESSAGE PRECISION SYSTEM TIMER WINDOW

(4) Command interpretation (HELP SIC Command)
Language\ PRIORITY SYNTAX

(5) Miscellaneous actions (HELP SIC Miscellaneous)
BEEP CPU DATE DEBUG DELAY FLUSH LOCK RANDOM_SEED USER UV WAIT

5.34.1 SIC FileSystem

SIC File System handling commands

```
SIC APPEND File FileAppended
SIC COPY FileOld FileNew
SIC DELETE File
SIC DIRECTORY Directory
```

```
SIC FIND FileFilter
SIC GREP String File
SIC MKDIR New_Directory
SIC MODIFIED FileName StrName
SIC PARSE Filename Name [Ext [Dir]]
SIC RENAME FileOld FileNew
```

5.34.2 SIC Procedure

```
                SIC Procedure related operations

SIC EXPAND Infilename Outfilename
SIC EXTENSION MacroExtension
SIC MACRO [ProcedurePath]
SIC OUTPUT [File [NEW|APPEND]]
SIC SAVE FileNew Command
SIC VERIFY [ON|OFF|MACRO] [ON|OFF|STEP]
SIC WHICH Procedure
```

5.34.3 SIC Customize

```
                  SIC Customization commands

SIC EDIT [Editor[]]
SIC ERROR
SIC HELP [HelpMode]
SIC INTEGER [SHORT|LONG]
SIC LOGICAL [LogName [Translation]]
SIC MESSAGE [Code]
SIC PRECISION [Single|Double]
SIC SYSTEM
SIC TIMER [Value]
SIC WINDOW [On|Off]
```

5.34.4 SIC Command

```
                  Command interpretation rules

SIC Language\ [Status]
SIC PRIORITY [Level Language [...]]
SIC SYNTAX [Fixed|Free]
```

5.34.5 SIC Miscellaneous

```
              SIC Miscellaneous operations and controls
```

```
SIC BEEP [N]
SIC CPU
SIC DATE
SIC DEBUG Item
SIC FLUSH
SIC LOCK LockFile
SIC RANDOM_SEED Value
SIC USER
SIC UV
SIC WAIT Seconds
```

5.34.6 SIC APPEND

```
[SIC\]SIC APPEND FirstFile FileAppended
```

Append file "FirstFile" to file "FileAppended". It is equivalent to Unix command

```
cat FirstFile >> FileAppended
```

but system independent.

The error status on return is ruled by the switch SIC SYSTEM (see HELP for details). The default is to raise an error if an error occurs during the concatenation.

5.34.7 SIC BEEP

```
[SIC\]SIC BEEP [N]
```

Will beep 1 or N times.

5.34.8 SIC CPU

```
[SIC\]SIC CPU [VERBOSE]
```

Will return the User CPU and System CPU times in real variables USER and SYSTEM resp. These variables are holded by two structures SIC%CPU%RAW% and SIC%CPU%CUMUL% which record times since last and first calls of SIC CPU resp. These structures also provide elapsed time in variables ELAPSED.

If the extra argument VERBOSE is present, the CPU times are evaluated, the Sic structures are filled (same as above), and additionally a summary is printed to the terminal.

5.34.9 SIC DATE

```
[SIC\]SIC DATE
```

Will return the current date and time in symbol SYS_DATE.

5.34.10 SIC DEBUG

```
[SIC\]SIC DEBUG LUN [Number]
[SIC\]SIC DEBUG IMAGE [Number]
[SIC\]SIC DEBUG MEMALIGN
[SIC\]SIC DEBUG MESSAGE
[SIC\]SIC DEBUG GFORTRAN
[SIC\]SIC DEBUG RESOURCES
[SIC\]SIC DEBUG PYTHON
[SIC\]SIC DEBUG VARIABLES
[SIC\]SIC DEBUG LOCALE [Value]
[SIC\]SIC DEBUG SYSTEM ON|OFF
```

Debugging command.

The first two keywords LUN and IMAGE print the status of reserved Logical UNit numbers (LUN), or of allocated image slots (IMAGE).

The MEMALIGN keyword will check if dynamically allocated buffers are aligned on 4, 8, 16 and 32 bytes. Some applications (e.g. FFTW3) are more or less faster dependending on the buffer alignment in memory.

The MESSAGE keyword swaps ON or OFF the debugging mode for messages. When turning it ON, all messages to both screen and message files are enabled. In this case it is a shortcut of the commands:

```
SIC MESSAGE GLOBAL ON
SIC MESSAGE GLOBAL A=FEWRIDTCU
```

When turning it OFF, standard message filters (maybe customized by user) are re-enabled. It is a shortcut of the command:

```
SIC MESSAGE GLOBAL OFF
```

Default is debug mode turned OFF, unless a '-d' option has been provided when invoking the Gildas executable.

The GFORTRAN keyword will check if the program is able to read and write correctly Gildas binary files. Some versions of the Gfortran compiler have an issue on this point, this command is intended to give a clear status to the user.

The RESOURCES keyword will display current resource usage and limits (memory, files, etc). See man pages for getrusage and getrlimit for details (Linux only).

The PYTHON keyword will display the Python version which was used to compile Gildas against the one used at run time. If they are different, this probably means troubles.

The VARIABLES keyword will display a technical list of the variables defined in Sic.

The LOCALE keyword, with no more argument, will display the current LC_ALL locale and some of its details. The current locale can be changed with an extra argument. Take care that Gildas is English-speaking, e.g. decimal point should be ".". If not, you are in trouble! It is wise to use the standard default, i.e. the "C" locale.

The SYSTEM keyword enables (argument ON) or disables (argument OFF) a debugging message displayed when a system() call is performed. system() calls are convenient and flexible to execute shell commands, but under Linux they need to fork a sub-process, duplicating temporarily the memory consumed by the current process. This may be a limiting factor on many cases. Default is OFF, i.e. message are disabled.

5.34.11 SIC COPY

```
[SIC\]SIC COPY FileOld FileNew
[SIC\]SIC COPY File Directory
```

Copy FileOld to FileNew, or copy File to the named Directory. It is equivalent to Unix command

```
cp FileOld FileNew
```

but system independent.

The error status on return is ruled by the switch SIC SYSTEM (see HELP for details). The default is to raise an error if an error occurs during the copy.

5.34.12 SIC DELAY

```
[SIC\]SIC DELAY Seconds
[SIC\]SIC DELAY
```

Delay the execution of commands. SIC DELAY is used in two times. The first call is given a duration as argument. Added to the current date and time, this defines a virtual milestone in the future. The next call without argument will wait for this milestone if it has not been reached yet, or will return without waiting if the limit is already past. During the waiting time, CTRL-C can be used to exit the command.

An application of SIC DELAY is to slow down the execution of loops which display or draw useful informations to the user. In the example below, we ensure that the loops are executed every second (or more), instead of depending on the CPU speed which varies over machines and epochs:

```
FOR I 1 to 10
  SIC DELAY 1.0
```

```
    Do something
    SIC DELAY
  NEXT
```

5.34.13 SIC DELETE

```
[SIC\]SIC DELETE File
```

Delete the specified file. It is equivalent to Unix command

```
    rm File
```

but system independent.

5.34.14 SIC DIRECTORY

```
[SIC\]SIC DIRECTORY Directory
```

Change the working directory to the specified name. It is equivalent to Unix command

```
    cd Directory
```

but system independent.

5.34.15 SIC EDIT

```
[SIC\]SIC EDIT [ON|OFF|EditorName]
```

List or change the status of the command line editing mode, or the default text editor used by command EDIT.

5.34.16 SIC ERROR

```
[SIC\]SIC ERROR
```

List the current error recovery command, defined by command ON ERROR.

5.34.17 SIC EXPAND

```
[SIC\]SIC EXPAND InFileName OutFileName
```

Add the language name before each command of the macro file InFileName and write the results into the new macro file OutFileName. This command should be used in a program where the InFileName macro does not produce any command name ambiguity. The resulting macro file will then be usable in any program.

The symbols are not touched because some procedures may need to redefine symbols on the fly.

Right now the SIC commands are not transformed because it would cause trouble to the procedure structure if some commands like FOR, IF, etc... would be changed.

5.34.18 SIC EXTENSION

```
[SIC\]SIC EXTENSION [Extension1 ... ExtensionN]
```

Without arguments, print the list of extensions used by commands @ and EDIT, with precedence given from the left to the right. Defaults are program dependent: for all Gildas softwares, it is the program name, while many user created applications use the generic default extension ".pro".

With one or more arguments, add the input extension(s) at the beginning of the list, i.e. with highest precedence. If extension was already known, SIC EXTENSION also brings it back at the beginning.

5.34.19 SIC FIND

```
[SIC\]SIC FIND [FileFilter [Directory]]
```

Search for all files matching the specified FileFilter in the specified Directory. Matching files are returned in a SIC structure named DIR%. DIR%NFILE is the number of found files, and DIR%FILE[1:DIR%NFILE] a character array containing the filenames.

The FileFilter can contain a directory name. In this case the returned file names will include it. On the contrary, if the FileFilter does not contain a directory name, and a "Directory" argument is specified, the returned names will not contain the Directory name.

The default file filter is *.*

5.34.20 SIC FLUSH

```
[SIC\]SIC FLUSH
```

Flush message and log files buffers onto disk.

5.34.21 SIC GREP

```
[SIC\]SIC GREP String File
```

Parse line by line the named (ASCII) file and search if the string appears in each line. The number of matching lines is saved in the output variable GREP%N. The matching lines themselves are saved in the array GREP%LINES; this array is defined only if GREP%N is non-zero.

5.34.22 SIC HELP

```
[SIC\]SIC HELP [PAGE|SCROLL|HTML]
```

With no argument, print how the HELP command behaves. Otherwise, controls whether it outputs the text page by page (PAGE), or continuously (SCROLL).

- SIC HELP PAGE indicates the HELP command should display page per page.
- SIC HELP SCROLL indicates the HELP command should display the the help in continuuous scrolling mode.
- SIC HELP HTML indicates the HELP command should refer the HTML version of the documentation and use the appropriate browser to display it.

5.34.23 SIC INTEGER

[SIC\]SIC INTEGER [SHORT|LONG]

Define the default SIC integer kind (i.e. used by DEFINE INTEGER). SHORT refers to Fortran's INTEGER*4 (with a limit value of 2**31-1) while LONG refers to INTEGER*8 (with a limit value of 2**63-1). Default is SHORT.

Without argument, the command displays the current default SIC integer kind.

5.34.24 SIC LANGUAGE

[SIC\]SIC Language\ [ON|OFF]

Place a language in the active scope, or removes it, or list its status. Languages not in active scope are not searched for their commands, unless the language name is explicitly given.

5.34.25 SIC LOCK

[SIC\]SIC LOCK [LockFile]

With no argument, display the list of lock files owned by the current program session. With a file name as argument, create this file as a lock file. Attempting to create a lock file which already exists is a fatal error. Lock files owned by the current program session are implicitely deleted when exiting.

This command is intended to protect some user's resources, e.g. invoking it in a procedure ensures that this procedure can not be run twice in the same or in another session.

5.34.26 SIC LOGICAL

[SIC\]SIC LOGICAL LogName Translation
[SIC\]SIC LOGICAL LogName

```
[SIC\]SIC LOGICAL [Pattern]
```

The first syntax sets or replaces a logical name.

SIC LOG LogName will give the translation of that precise logical name.

Finally, SIC LOG Pattern will search for all logical names matching the pattern. A wilcard '*' means 0 or more characters. Without second argument the pattern '*' is used, i.e. it will list all the logical names.

5.34.27 SIC MACRO

```
[SIC\]SIC MACRO MacroPath
```

Change the search path for the procedures. Procedures executed by @ are searched in a path specified by the special logical name MACRO#DIR: . This command allows to list or change the content of MACRO#DIR: .

5.34.28 SIC MEMORY

```
[SIC\]SIC MEMORY [ON|OFF]
```

Enable or disable automatic insertion of successful commands into the stack.

5.34.29 SIC MESSAGE

```
[SIC\]SIC MESSAGE
[SIC\]SIC MESSAGE Pack1 [[[S|L|A]-|=|+]F|E|W|R|I|D|T|C|U] [PackN...]
[SIC\]SIC MESSAGE Global [[[S|L|A]-|=|+]F|E|W|R|I|D|T|C|U] [ON|OFF]
[SIC\]SIC MESSAGE * [[[S|L|A]-|=|+]F|E|W|R|I|D|T|C|U] [PackN...]
[SIC\]SIC MESSAGE /COLOR [Args]
```

The command SIC MESSAGE allows to customize the general verbosity of all the GILDAS commands, or to customize the color of the messages depending on their kinds (see HELP SIC /COLOR for details).

Modify and display the messaging filters for one or more packages. Messages are usually printed on screen and into a message file in the GAG_LOG: directory. Depending on its kind (from trace to fatal errors), a message may be printed or not to one of these outputs. Messaging filters allow the user to fine tune the kinds of messages he wants to see. Filters are either global (i.e. all the messages are filtered whathever the package they belong to) or package-dependent (i.e. the messages are filtered depending on the package they belong to).

The command SIC MESSAGE,

- without argument, outputs the messaging filters for all active packages;

- with one or more package name as arguments, displays the associated filters, e.g. :

```
GREG> SIC MESSAGE SIC GLOBAL
sic    on-screen  active   filter: FEWRI---U
sic    to-mesfile active   filter: FEWRIDTCU
global on-screen  inactive filter: FE-------
global to-mesfile inactive filter: FEWRIDTCU
```

  First column shows the package name, second column the output device, third column the filter status, last column the associated filter;
- with a package name followed by a messaging rule (see below), updates its the filter value and displays it. The filters of several packages can be changed on the same command line, e.g. SIC MESSAGE SIC S+D GREG L-W

The package name can additionally have two special values:
- * : all the package filters are modified according to a single input rule;
- GLOBAL : set a global filter which can override all the package filters without loosing them. This global behavior is activated and deactivated by the ON or OFF keyword.

A rule to change a filter is a single string composed of three parts, from left to right:
- A, S and/or L as first argument to modify All, Screen, or Logfile filters. This must be unique but is optional, default is Screen only.
- +, - or = as second argument to add to, remove from, or redefine filter values. This operator must be unique, and it is optional: default is +.
- F, E, W, R, I, D, T, C and/or U as last argument(s) to modify the filters on Fatal, Error, Warning, Result, Info, Debug, Trace, Command and Unknown messages. This argument is mandatory, and the letters may be associated.

The message kinds are:
- F)atal: Program will cleanly end now because a fatal error occured, e.g. a required initialization of the program could not occur or an error can not be safely recovered or the program is in an unstable status and can not go on.
- E)rror: Command or action could not be executed to its end. Such errors can be recovered, and program can continue to run safely.
  - It denotes an attempt to do something not allowed or not implemented in the command.
  - Command can not run to the end as it will not produce the expected result.
- W)arning: Command or action will go on executing without an error, but

```
    - the user must be warned about a strange behavior;
    - a result is produced, but user must be careful with its  interpre-
      tation.
  R)esult:  Information  directly requested by the user. It concerns all
    the results returned by active commands. This should  be  understand
    as:
      - "You asked for this, as a result, that happened"
      - "This was created/changed accordingly to your request"
      - "Here is what you asked for (some values,...) "
  I)nfo:  Additional  information not directly expected by the user when
    he runs the command. It gives secondary informations on current  ac-
    tions,  or  it concerns more or less the result and the processes to
    obtain it, but it is not the result itself.
  D)ebug: High level debugging, e.g. follow the steps of complex  compu-
    tation.  By default, it will not be printed. User can activate it if
    strange non-fatal behavior occurs.
  T)race: Low-level debugging, e.g. track the program execution.  By de-
    fault,  it  will  not be printed. User must use it wisely because it
    may produce thousands of messages in a row (for example, in  loops).
    User can activate it in case of unforeseen fatal behavior.
  C)ommand:  Each  command  typed  on  the  terminal is resolved and can
    printed back to terminal and to logfile. This is equivalent  to  the
    obsolescent SIC VERIFY ON behavior.
  U)nknow: It is reserved for the migration from the old message facili-
    ties to the new one, and to  handle  internal  errors  in  messaging
    process.

Simple examples:
    SIC> SIC MESSAGE
    sic     on-screen  active   filter: FEWRI---U
    sic     to-mesfile active   filter: FEWRIDTCU
    SIC> SIC MESSAGE SIC GLOBAL
    sic     on-screen  active   filter: FEWRI---U
    sic     to-mesfile active   filter: FEWRIDTCU
    global  on-screen  inactive filter: FE-------
    global  to-mesfile inactive filter: FEWRIDTCU
    SIC> SIC MESSAGE SIC S+D
    sic     on-screen  active   filter: FEWRID--U
    sic     to-mesfile active   filter: FEWRIDTCU
    SIC> SIC MESSAGE SIC L-DT
    sic     on-screen  active   filter: FEWRID--U
    sic     to-mesfile active   filter: FEWRI--CU
    SIC> SIC MESSAGE SIC A=FEWRI
    sic     on-screen  active   filter: FEWRI----
    sic     to-mesfile active   filter: FEWRI----
    SIC> SIC MESSAGE GLOBAL ON
    Turning ON global filtering rules
```

```
SIC>
```

Q: I've lost all messages, SIC MESSAGE is silent, what happens?
A: Many message kinds are certainly disabled, in particular Results printed to Screen. This is why SIC MESSAGE is also silent. Consider typing SIC MESSAGE SIC S+R and you should see back messaging filters.

5.34.30 SIC /COLOR

```
[SIC\]SIC MESSAGE /COLOR
[SIC\]SIC MESSAGE /COLOR ON|OFF
[SIC\]SIC MESSAGE /COLOR [kind1=color1] [kind2=color2] ...
```

Customize the message colors in the terminal depending on the message kind.

Without argument, show the current color (if relevant) of all the message kinds, and the available colors.

With ON or OFF as argument, enable or disable the message coloring.

With one or more arguments of the form "kind=color", customize the color of the given message kind. Type SIC MESSAGE /COLOR without argument to know the various message kinds and available colors. Use color "NONE" to revert to the default terminal color. For example, the default at start-up is:

```
SIC> SIC MESSAGE /COLOR F=RED E=RED W=ORANGE
```

5.34.31 SIC MKDIR

```
[SIC\]SIC MKDIR New_Directory
```

Create a new directory. It is equivalent to Unix command

```
mkdir -p New_Directory
```

but system independent. No error is raised if the directory already exists.

5.34.32 SIC MODIFIED

```
[SIC\]SIC MODIFIED FileName StrName
```

Check if a "file" was modified since last call, and fill the Sic structure named "StrName" accordingly. The file can be a regular file or a directory. If file is a symbolic link, evaluation is made on its target. On first call, the structure is set up and the file is assumed modified. On subsequent calls, file and calling dates are compared and modification status is evaluated. In case of doubt (e.g. modification time unchanged at the file system precision), a modification is assumed.

The output structure has the following (scalar) components:
- StrName%FILE: the file name (character string),
- StrName%MTIME: the file modification time (long integer, in nanoseconds since 01-jan-1970),
- StrName%PTIME: last time a modification was proved (long integer, in nanoseconds),
- StrName%MODIF: file was (or may have been) modified or not since last call (logical)

The structure must be a user-defined, global, variable. If the structure or its elements are missing, they will be created at first call.

5.34.33 SIC OUTPUT

```
[SIC\]SIC OUTPUT [?|FileName [NEW|APPEND]]
```

Redirect the output of the SIC\SAY command to the specified file.

If NEW is passed after the file name, the file name is suffixed (renamed) with a ~ if it was existing and a new empty file is created. This is the default if this keyword is omitted.

If APPEND is passed after the file name, the file is reopened for appending text at the end if it was existing before. Else, a new empty file is created.

If ? is passed as file name, the command prints the current output name (terminal or file name).

If no argument is present, any currently opened output file will be closed, and the output of command SAY is re-directed to the terminal.

5.34.34 SIC PARALLEL

```
[SIC\]SIC PARALLEL [Nthread]
```

Check or set the number of threads for parallel executions. Nthread is either an integer indicating the desired number of threads, or "*" to use all the cores of the current machine.

The current and maximum number of threads are visible in variables SIC%OPENMP%NTHREADS and SIC%OPENMP%MTHREADS respectively.

Only active when compiled with Open-MP mode.

5.34.35 SIC PARSE

```
SIC PARSE Filename Name [Ext [Dir]]
```

Parse an input filename, and return the short Name, and optionally the

Extension and Directory into pre-defined SIC character variables.

5.34.36 SIC PRECISION

```
[SIC\]SIC PRECISION [SINGLE|REAL|DOUBLE|AUTO]
```

Select the precision of all computations using SIC variables. Automatic precision will use the highest precision present in an expression.

5.34.37 SIC PRIORITY

```
[SIC\]SIC PRIORITY [Level1 Lang1 ... [LevelI LangN ...]]
```

SIC PRIORITY redefines the languages precedence when an ambiguous command (same name in several languages) is encountered: the lower the precedence level, the higher the priority. By default, they are all the same (level is set to 1), i.e. there is no automatic ambiguity resolution.

The priority levels can be changed for one, some, or all languages. LevelI is an integer, which must be followed by one or more languages:

- > 0: the level value is used directly, the lower level the higher priority,
- = 0: the level value is Automatic, i.e. the priority comes after the languages with an explicit positive value and before the ones with a negative value,
- < 0: set the priority from the end of the list, e.g. -1 means that the language must have the lowest priority.

Example:

Level	Priority
+1	1
+2	2
+4	3
0	4
-2	5
-1	6

Note that the priority list is compressed if the values are discontinued.

A call to SIC PRIORITY without arguments prints the current levels for all languages.

5.34.38 SIC RANDOM_SEED

```
[SIC\]SIC RANDOM_SEED [DATETIME|URANDOM|Value]
```

This command resets the Fortran random seed used to initialize the sequence of pseudorandom numbers returned by the functions RANDOM and

NOISE. Several arguments are allowed:
- DATETIME: compute a seed based on current date and time. This is portable but not highly random, e.g. 2 processes running at the same time can end with the same seed.
- URANDOM: use the operating system special file /dev/urandom as a random number generator, if available. This is the best choice but it is not fully portable.
- Value: the user can specify its own seed (integer value). Reusing the same value will ensure the same sequence of pseudorandom numbers later on, which can be useful depending on the context.

Note that this command affects the seed, i.e. the starting point of the sequence of pseudo-random numbers. It does not affect the randomness of those numbers (i.e. you can safely use the Value 1 if you want).

The default at startup is DATETIME, i.e. the sequence will always be different from one session to another. This default can be overriden by setting the Sic logical GILDAS_RANDOM_SEED to DATETIME, URANDOM, or an integer value in the file $HOME/.gag.dico

Without argument, the command displays the current Fortran random seed in use. For debugging purpose.

5.34.39 SIC RENAME

```
[SIC\]SIC RENAME FileOld FileNew
```

Rename an existing file. It is equivalent to Unix command
```
        mv FileOld FileNew
```
but system independent.

5.34.40 SIC SAVE

```
[SIC\]SIC SAVE [FileName Symbol]
```

This is a specific command (currently) used by the ALMA simulator to copy into an output file (a procedure) the name of any newly defined SIC variable, prefixed by whatever Symbol is given here.

Without argument, just closes the current "save" file.

Can be used whenever you need to apply different actions to the same set of variables, or get two sets of variables with similar names (one prefixed, the other not) e.g. to store default values or last values. Redefining the Symbol and executing the created procedure will do the desired job on the list of variables defined when SIC SAVE was active...

```
sic save define_all TOTO
```

```
define a Bunch Of Variables Here /global ! Only global stuff, though...
sic save
!
symbol TOTO "EXA &1 "   ! Examine the Variable name
@ gag_proc:define_all   ! for all variables in this list...
!
sic output init_all.sic
symbol TOTO "@ sicvar_init"   ! Look into gag_pro:clone_var.sic
@ gag_proc:define_all         ! for further explanation
sic output
@ init_all.sic                ! Initialize them
!
define structure CLONE% /global
sic output clone_all.sic
symbol TOTO "@ sicvar_clone CLONE% "   ! Will actually create a
@ gag_proc:define_all         ! copy of all variables into structure CLON
sic output
symbol TOTO continue
@ clone_all.sic                ! Duplicate them into CLONE% structure
```

5.34.41 SIC SEARCH

```
[SIC\]SIC SEARCH FileName
```

(Obsolescent).
Search for the specified file, and set the logical SIC%EXIST to YES if the file exists, NO if not.

This is an obsolescent feature. The same functionality is available through the logical function FILE("FileName").

5.34.42 SIC SYNTAX

```
SYNTAX of SIC commands
```

and

```
[SIC\]SIC SYNTAX [FIXED|FREE]
```

The SIC syntax is the following :

```
[LANG\]COMM [Arg1 [Arg2 [...]]] [/OPT1 [p1 [...]] [/OPT2 [...]]
```

where [] indicates optional fields.

Language, command and options can be abbreviated. The language field (LANG\) is optional but may help resolve ambiguities.

First, the line is stripped of duplicate separators (spaces or tabs). Character strings (entities included between double quotes) are not affected by this formatting.

Then symbols (entities included between simple quotes, plus the language-command field) and tokens (macros parameters 1 2 etc...) are translated, even within character strings.

Finally, the line is analysed for ambiguities and the language, command and option names are expanded. An option is a word beginning by a slash (e.g. /OPT1 is an option in the above example).

All arguments can be character or mathematical variables or expressions, depending on the type required by the program. Character expressions can be concatenated with implicit formatting of variables and mathematical expressions, such as in

"The real number PI is equal to "'ACOS(-1.0)'

Variable and expressions are not evaluated during the parsing, but only during the execution. Character variables must be included between quotes for translation, e.g.

```
DEFINE CHARACTER C*6
LET C 3.14159
DEFINE REAL A
LET A 'C'/PI
EXAMINE A
  A       =    0.9999705
```

[SIC\]SIC SYNTAX [FIXED|FREE]

controls the syntax for mathematic operations. In FIXED syntax, the LET command is compulsory. In FREE syntax, commands lines like

```
A[I,J] = SIN((2*I+J)/PI)
```

where A is a known variable, are recognized as assignement and automatically expanded to the equivalent FIXED syntax

```
LET A[I,J] SIN((2*I+J)/PI)
```

Free syntax cannot be used for the LET /WHERE command.

5.34.43 SIC SYSTEM

[SIC\]SIC SYSTEM [ERROR|NOERROR]

Indicate if the commands SIC\SYSTEM, SIC\SIC COPY, and SIC\SIC APPEND can raise errors or not, i.e. if any error occuring when executing the string command (SIC\SYSTEM), or occuring when copying/appending a file should stop the Sic execution flow. Default is true. The current status is also available in the logical variable SIC%SYSTEMERROR.

5.34.44 SIC TIMER

[SIC\]SIC TIMER [Time [HOURS|MINUTES|SECONDS]]

All Gildas programs are automatically closed (normal exit) after a defined period of inactivity at the prompt level. This period is defined by the Sic timer.

With a Time argument, set the timer to the input value. Unit is a keyword which can be Hours, Minutes, or Seconds (default Hours). A null or negative value disables the timer.

Without argument, display the current value of the timer.

The timer can be customized in a Sic logical named SIC_TIMER (integer value, in hour unit). This value is evaluated once at startup from one of the Sic logical dictionaries. Later changes are ignored. Default is 3 hours.

5.34.45 SIC USER

[SIC\]SIC USER

THIS COMMAND IS OBSOLESCENT. Use instead variables SIC%USER and SIC%HOST defined at startup.

SIC USER will return the user name (usually with host name) in symbol SYS_INFO.

5.34.46 SIC UVT_VERSION

[SIC\]SIC UVT_VERSION Version

[EXPERIMENTAL -- Reserved for developpers so far]

Select the UV Table version being used by SIC. The Version can be FREQUENCY, DOPPLER or SYSTEM. It affects the way the spectral axis is interpreted to define spectral and spatial resolution.

FREQUENCY is the "old" historical behaviour, which is only suitable for small field of views and number of channels.

DOPPPLER is suitable for large number of channels, but may be limited to small field of views in case of data spanning several observation dates

SYSTEM is more accurate, but requires an extra column in the UV Tables to hold time dependent Doppler tracking.

5.34.47 SIC VERIFY

[SIC\]SIC VERIFY
[SIC\]SIC VERIFY ON|OFF

```
[SIC\]SIC VERIFY MACRO ON|OFF|STEP
```

Control the listing of macros, loop or stack during execution. Without argument, SIC VERIFY shows its current status.

SIC VERIFY ON enables echoing each command in the terminal before execution. This is mostly useful for debugging. Note that numerous outputs in the terminal slow down considerably the procedure execution. Default is OFF.

SIC VERIFY MACRO ON adds feedback in the terminal when entering and leaving a procedure (procedure file name, number of arguments, arguments). It also waits for the user to type <Return> before starting the procedure execution. SIC VERIFY MACRO STEP adds feedback in the terminal before executing each command (similar to SIC VERIFY ON), but it also waits for the user to type <Return> before executing the command. Default is OFF, i.e. no feedback is shown, and no need to type <Return>.

5.34.48 SIC VERSION

```
[SIC\]SIC VERSION
```

Return the version number and credits or copyrights for the languages or subprograms currently used.

5.34.49 SIC WAIT

```
[SIC\]SIC WAIT Seconds
```

Will wait for the required number of seconds. Fraction of seconds can be specified.

5.34.50 SIC WHICH

```
[SIC\]SIC WHICH Procedure
```

Return the full path of the procedure or macro which will be executed by "@ Procedure" or "TYPE Procedure" calls.

See HELP @ for details on which directories procedures and macros are searched in.

5.34.51 SIC WINDOW

```
[SIC\]SIC WINDOW [ON|OFF]
```

Allow or disallow use of GUI mode.

5.35 SORT

```
[SIC\]SORT KeyVar [Var1 [... VarN]]
```

Sort an ensemble of 1-D or 2-D SIC variables by increasing values of the (1-D only) KeyVar variable. A standard use is to sort a complete GILDAS table, using

```
DEFINE TABLE A MyFile WRITE
SORT A[3] A
DELETE /VAR A
```

As shown in the example, the command handles the case where KeyVar is a subset of any of the variables to be sorted. However, if there is aliasing between any of the Var1 ... VarN variables, the result is unpredictable.

The command can be useful for further use of the COMPUTE LOCATION command, or plots using GreG. To mimic the behaviour of the (obsolete) GREG\SORT command, use

```
SIC\SORT Key Var2 Var3
```

where Key is any of X Y Z, and Var2 and Var3 the two other ones.

5.36 SYMBOL

```
[SIC\]SYMBOL [X ["Translation"]] [/INQUIRE "Prompt Text"]
```

This commands defines, lists and deletes symbols.

SYMBOL
Lists the Symbol table

SYMBOL X
Gives the translation of the symbol X

SYMBOL X "Translation"
Defines a new symbol or update the precedent symbol definition.

SYMBOL X /INQUIRE "Prompt Text"
Inquires the definition of a symbol with the specified prompt (for interactive session only).

A symbol is an abbreviation of any character string. The symbol translations are substituted to the corresponding symbols when found in a command line. In a command line, symbol TOTO must appear inside simple quotes (like 'TOTO') to be translated, except for the line beginning where SIC assumes the first word might be a symbol. A symbol definition may refer to another already defined symbol. The substitution occurs everywhere, even within the character strings.

Symbols in FOR-NEXT loops are substituted at compilation time (i.e. when the command line is written but not yet executed), using their current value. In a loop, redefining a symbol which is already defined out of the loop has no effect since its occurences are substituted before the execution of all the commands.

Symbols can be deleted when they are no longer usefull, using command DELETE /SYMBOL.

5.37 SYSTEM

```
[SIC\]SYSTEM ["Command"]
$ Command
```

Execute a command from the operating system, or create a subshell.

If no argument is given, start a subshell based on the SIC logical GAG\PROCESS. If it is not defined, the user $SHELL environment variable is used. If it is also not defined, the last fallback is /bin/sh. The subshell can be terminated by typing 'exit' or 'bye' or case.

If an argument is given, execute the command in a /bin/sh subshell (note that /bin/sh is system-dependent and might behave differently from one system to another). It must be a single argument: use double-quotes to enclose multiple arguments separated by blanks. Note that an error is raised if the shell command returns a non-zero status. This behavior can be modified in 2 manners:
- use dedicated command options, if any, to ensure it does not return a non-zero status (for example -q option for grep),
- catch the error in the shell way, e.g. terminate the command line with " || true" which executes the "true" command if an error occurs. Obviously ignoring errors is at your own risk.

Note that Unix environment variables cannot be defined in such a way, since it is a subshell (i.e. the environment modifications are lost when the subshell ends). In particular, use command SIC DIRECTORY to change your working directory.

$ Command:
At the interactive prompt, system commands can also be executed directly from the SIC level using the $ token. "Command" must be a valid operating system command in the default shell of the user. The $ token is invalid in procedures.

5.38 TIMER

```
[SIC\]TIMER
[SIC\]TIMER ON|OFF
```

```
[SIC\]TIMER [Time [Unit]] [/COMMAND String]
```

All Gildas programs are automatically closed (normal exit) after a defined period of inactivity at the prompt level. This period is defined by the Sic timer. The timer starts as soon as the program prompt is available (i.e. inactivity). The timer is stopped when a command starts running. It restarts from the beginning when the prompt is back.

Without argument, display the timer status.

With the keyword ON or OFF as argument, enable or disable the timer. The timer is enabled by default.

With a Time argument, set the timer to the input value. Unit is a keyword which can be HOURS, MINUTES, or SECONDS (default HOURS). Default value at startup is 3 hours.

The command executed once the timer is done can be customized with the option /COMMAND. The command string must be a single argument, possibly double-quoted if blanks are needed, e.g.

```
  SIC> TIMER /COMMAND EXIT
  SIC> TIMER /COMMAND "EXAMINE PI"
  SIC> TIMER /COMMAND "SAY ""Hello from timer"""
```

Default command at startup is EXIT.

The timer duration can be customized in a Sic logical named SIC_TIMER (integer value, in hour unit). This value is evaluated once at startup from one of the Sic logical dictionaries. Later changes are ignored. Default is 3 hours.

5.39 TYPE

```
[SIC\]TYPE [Macro_Name] [/OUTPUT OutFile]
```

TYPE lists the commands of the named macro. The same search rules as for command SIC\@ apply (see HELP @). The command SIC WHICH can be used if you have doubts on the TYPE'd macro.

If no argument is given, the Stack is listed.

The output of the command can be redirected to a file instead of the standard output (STDOUT) thanks to the option /OUTPUT.

5.40 @

```
[SIC\]@ Macro_Name [Arg1 [Arg2 [...]]]
```

Read commands from macro (or procedure) Macro_Name and execute them.

Default file extension is program dependent (usually the program name, or .pro), and can be changed using command SIC EXTENSION. Procedures are searched for according to the following rules:
- first in the directory designated by the logical name GAG_PROC:
- then, in order, in all the directories in the path specified by the logical name MACRO#DIR: (see command SIC MACRO).

One can use the command SIC WHICH to be sure of the macro which will be executed.

All commands will be echoed to the terminal when executed if the VERIFY switch is ON (see command SIC VERIFY).

Macros (as any other text files) can be edited using a standard text editor by typing command EDIT with the macro file name as argument (see EDIT and SIC EDIT).

5.40.1 @ ARGUMENTS

```
[SIC\]@ Macro_Name [Arg1 [Arg2 [...]]]
```

The arguments passed to the macro are described by the structure PRO%:
- PRO%NAME: calling procedure name,
- PRO%NARG: number of arguments passed,
- PRO%ARG: a character array (size PRO%NARG) providing all the arguments. This array is not defined if PRO%NARG is 0.

The structure is updated when entering each macro. Up to 32 parameters can be given.

Note that the arguments are all saved as character strings. A typical use is:

```
SIC> LET MYVAR 'PRO%ARG[1]'
```

to substitute the character string by its content. See also HELP PARSE for advanced argument parsing.

The old-fashion way to access the arguments is to use the tokens &1, &2, ..., &9. They are substituted to the corresponding arguments as they are found in the body of the procedure, even within character strings. For example

```
SIC> say "a&1b"
```

will display the first argument value surrounded by "a" and "b". Note that these tokens can be used to access the 9 first arguments only.

6 GUI Language Internal Help

6.1 Language

```
BUTTON : Associate a command with a button
END    : Read parameters from window and set variables accordingly
GO     : Activate the currently defined window
MENU   : Create a pulldown menu for next buttons
SUBMENU: Create a submenu in the current menu
PANEL  : Define or delete an input window or menubar
WAIT   : Wait for GO button in the current window
URI    : Open any kind of URI, e.g. html link or file name.
```

6.2 BUTTON

GUI\BUTTON "Command" Button ["Title" HelpFile [OptionTitle]]
(Graphic-User-Interface mode only)

Creates a button widget to execute the specified "Command".

If no "Title" is given, the button will have no associated variables, and will appear with other similar buttons at the top of the window.

If a "Title" argument is present, a "secondary parameters" window is created. In the main window, the "title" appears followed by 3 buttons: one with the button name, one pointing to the secondary parameters window, and a "HELP" button. All subsequent LET commands will create widgets in this secondary window, until a new GUI\BUTTON command is typed. "HelpFile" specifies a text file where the help for the variables can be found, and "OptionTitle" is a title for the secondary window (and associated button).

6.3 END

GUI\END

Reads all parameters from the current window(s) and set the modified variables accordingly. Normally reserved for programming applications.

6.4 GO

GUI\GO ["Command"]

Map the windows defined by the previous GUI\PANEL command and its associated GUI\BUTTON and LET commands. "Command" is an optional command to be executed when button "GO" is pressed.

6.5 MENU

GUI\MENU "Title" [/CLOSE] [/CHOICES]

Creates a pulldown menu to group a set of buttons without associated pa-

rameters. Subsequent GUI\BUTTON or GUI\URI commands will add buttons in the pulldown menu.

When option /CLOSE is present, closes the current pulldown menu. Subsequent GUI\BUTTON or GUI\URI commands will create buttons on the main menubar.

This command is valid only when creating a detached menubar, i.e. after a GUI\PANEL/DETACH command has been typed.

6.6 SUBMENU

```
GUI\SUBMENU "Title" [/CLOSE]
```

Create a new submenu in the current menu. Subsequent GUI\BUTTON or GUI\URI commands will add buttons in this submenu.

When option /CLOSE is invoked, closes the current submenu. Subsequent GUI\BUTTON or GUI\URI commands will create buttons in the parent menu.

6.7 PANEL

```
GUI\PANEL "Title" HelpFile [/DETACH] [/LOG LogFile]
GUI\PANEL [HelpFile] /CLOSE
```

Activate the Graphic-User-Interface input mode for variables. A window with the specified title is created, but not mapped to the screen. Successive LET commands will create widgets in this window to allow to modify variables by entering values in the widgets. Command GUI\GO "Command" will map the window to the screen. Once the proper input has been defined, clicking on the "GO" button will setup all the related variables in the main program, and execute the associated command. Clicking on button "UPDATE" will only set the variables. Clicking on button "ABORT" will return without modifying the variables.

Help is available in the window through the "HELP" button, but also clicking in any prompt area.

See LET command for details.

If option /DETACH is present, a menubar is created instead of a normal window. Several buttons can be attached to this menubar using the GUI\BUTTON and GUI\MENU commands, but no variables can be set in this mode. The menubar is mapped when command GUI\GO is typed.

If option /CLOSE is present, the last detached menubar, or the specified one, is deleted.

/LOG option stores variable definitions in specified log file.

6.8 WAIT

```
GUI\WAIT
```

Wait for button "GO" "UPDATE" or "ABORT" to be pressed. The command monitor stays in hold state until one of these buttons in the main window are pressed.

6.9 URI

```
GUI\URI
```

Open any kind of URI (Uniform Resource Identifier), e.g. html link or file names, for example:

```
GUI\URI "http://www.iram.fr/IRAMFR/GILDAS/" "GILDAS Web Page"
GUI\URI gag_doc:pdf/gildas-intro.pdf "GILDAS Introduction"
```

The choice of the software used to open the URI is let as much as possible to the system (which should itself honor the user's preferences).

7 VECTOR Language Internal Help

7.1 Language

```
VECTOR\ Language Summary

FITS        : Convert between FITS files and Gildas images
HEADER      : List the header of a Gildas image
RUN Task    : Activate a GILDAS task
SPY [Task]  : Look at current status of detached Tasks
SUBMIT Task : Submit a GILDAS Task in batch queue GILDAS_BATCH
TRANSPOSE   : Transpose data cubes or SIC Variables
```

7.2 FITS

```
[VECTOR\]FITS FitsFile
[VECTOR\]FITS FitsFile TO GildasFile [/STYLE Style] [/HDU Number]
[/BLC Position...] [/TRC Position...] [/OVERWRITE]
[VECTOR\]FITS FitsFile FROM GildasFile [/STYLE Style] [/BITS Nbit]
[/OVERWRITE]
```

With a single FITS file name as argument, give a summary of all HDUs found in the file. With more arguments and options, convert between FITS files and Gildas images or tables.

With the option /OVERWRITE, the command will silently overwrite the output file if it already exists.

7.2.1 FITS FROM

[VECTOR\]FITS OutputFile.fits FROM InputFile.gdf [/STYLE Style] [/BITS Nbit]

Create a FITS file from a Gildas image or table. The layout of the FITS file depends on the specified "Style", although default styles will be used depending on the Gildas image (e.g. UVFITS is used for UV Tables, STANDARD is used for images).

The number of bits is controlled by "Nbit". It defaults to -32 for images, and +16 for UV data. See subtopic for details.

Note that, in order to conform Calabretta & Greisen 2002 formalism and recommendations (which define the FITS description of the spatial projections of the celestial sphere), radio-projected maps are exported to FITS under the Sanson-Flamsteed (SFL) description, by properly redefining the reference point on the Equator with no need for reprojection of the data itself.

Options /BLC and /TRC are not available in this direction.

7.2.2 FITS TO

[VECTOR\]FITS InputFile.fits TO GildasFile.gdf [/STYLE Style] [/HDU Number] [/BLC Min1 ... MinN] [/TRC Max1 ... MaxN]

Convert the InputFile.fits FITS file into a Gildas image in GildasFile.gdf, using the appropriate "Style". A specific HDU can be specified instead of the Primary FITS header (default is 1, i.e. Primary).

A subset of the input file data can be extracted by specifying the bottom left corner (/BLC) and the top right corner (/TRC) coordinates. For example

```
FITS ... /BLC 10 100 /TRC 20 200
```

will extract the range [10:20] in the first dimension, and [100:200] in the second dimension. Dimensions omitted default to 0, which stands for the largest possible corner position in the corresponding direction.

7.2.3 FITS /BITS

[VECTOR\]FITS OutputFile.fits FROM InputFile.gdf /BITS Nbit

The number of bits offers two controls: the data size and the encoding scheme. This value will be found in the resulting header under the card BITPIX.

Encoding scheme:

It is controled by the Nbit sign.

Positive sign means indexed encoding: the values are scaled as integers distributed among the 2**abs(Nbit) values. The forward and back-conversion from actual values to indexed values are based on the data extrema. Beware they must be up-to-date in the GDF header before converting a BITPIX>0 FITS (see command V\HEADER /EXTREMA)! Blank values are identified as the last integer of the indexed scale.
Negative sign means the actual values are saved "as is" (IEEE floating point format). Blank values are saved as IEEE NaN.

Data size:
It is controled by the Nbit absolute value: 16, 32, or 64 will consume respectively 2, 4 and 8 bytes per value, with a direct impact of the file size.

Nbits=+16 is worth only if the data dynamic (range of values) is low, otherwise value precision loss will happen (2**16 values are not enough to cover properly a large range of values). In the other cases, Nbit<0 is more modern and should be prefered. The command supports Nbits +16, +32, -32, -64.

7.2.4 FITS /STYLE

[VECTOR\]FITS ... /STYLE Name

From FITS to GILDAS, the command automatically chooses between an IMAGE FITS (STANDARD style) and a UVFITS (UVFITS style). /STYLE option can be used to customize the UVFITS import:

STANDARD
for IMAGE FITS (images, cubes,...),
UVFITS
for UV FITS,
CASA_UVFITS
same as UVFITS, but keep all Stokes parameters if relevant.

From GILDAS to FITS, support styles are:

STANDARD
automatic choice between an IMAGE FITS (image, cube,...) and regular UVFITS for UV data,
UVFITS
default for UV data,
CASA_UVFITS
same as AIPSFITS,
AIPSFITS
enables writing the antenna table extension, a 7th "IF" axis, and proper velocity description,
SORTED_AIPSFITS: same as AIPSFITS, with time-sorting of the visibilities enabled.

7.3 HEADER

[VECTOR\]HEADER GildasFile|FITSFile|HeaderVariable [/EXTREMA] [/TELESCOPE Args]

Interact with file headers or header variables. Some actions are possible or not depending on the header kind or properties.

If no option is present, display the named header.

The /EXTREMA option computes (or recomputes) and updates the extrema section in the header.

The /TELESCOPE option creates or updates the telescope section. See subtopic for details.

7.3.1 HEADER /TELESCOPE

[VECTOR\]HEADER GildasFile|HeaderVariable /TELESCOPE [+-]Tel1

This option adds (leading +) or removes (leading -) one or more telescope from the telescope section. If the + or - character is omitted, + is assumed.

For example:
```
  SIC> HEADER myfile.gdf /TELESCOPE 30M
```
adds the 30M to the telescope section.
```
  SIC> HEADER myfile.gdf /TELESCOPE -ALMA +ACA
```
removes the ALMA telescope and adds ACA.

It is not an error to add a telescope already present, and it is not an error to remove a telescope already absent.

7.4 RUN

[VECTOR\]RUN TaskName [Parameter_File] [/EDIT] [/NOWINDOW] [/WINDOW]

Execute a GILDAS task as a detached process.

See also:
  HELP TASK for a summary of all available tasks.
  HELP TASK GroupName for a summary of tasks in a group.
  HELP RUN TaskName for details on a specific task.

Input file (*.init):
  The input parameters are read from the file Parameter_File. It is a SIC procedure with commands from the TASK\ language. The default name of the parameter file is TaskName.init in the current working directory. If it does not exists yet, a template duplicated from the

```
  GAG_TASK: location is used. If this template file does not  exist,  it
  may  be that the Task you want to run does not exist either, or is not
  yet debugged at all.

  There are 3 ways to set up the input file:
  - Window mode: if the variable RUN_WINDOW is YES (default on  X-Window
        systems), or if the option /WINDOW is specified, a panel appears
        to enter all parameters. Help if available by  clicking  on  the
        prompt  string  for  each parameter, or on the HELP button. Once
        all parameters have been adequately specified, the task  can  be
        activated by clicking OK, or aborted by clicking ABORT.
  - Editor  mode:  if  the option /EDIT is specified, your favorite text
        editor (see SIC EDIT) is invoked to edit the parameter file.
  - No window mode: if the variable RUN_WINDOW is NO, or if  the  option
        /NOWINDOW is specified, you won't be asked to edit the parameter
        file. This is useful for non-interactive data  processing.  This
        assumes you have created the parameter file previously, e.g. "by
        hand" or from a previous call.

  Once the parameter file has been prepared in Edit-mode or Window-mode,
  the  RUN  command  will  prompt  you for all missing parameters in the
  .init file.

Parameter file (*.par):
  A temporary parameter file derived from the init file  is  created  by
  the RUN command in the GAG_SCRATCH: directory. It may be deleted after
  Task execution. See also HELP SPY.
  At this stage a second SIC procedure is executed before  task  submis-
  sion  to  check  the validity of input parameters. If any parameter is
  invalid, an error is returned and the Task not submitted.

Log file:
  The  output  messages  of  the   task   are   logged   in   the   file
  GAG_LOG:TaskName.gildas.  The Sic variable SIC%TEE controls wheter the
  messages should be printed in real time (YES) or after the task execu-
  tion  (NO).  Default  is  NO. Beware the 'tee' program does not always
  propagate task execution errors (this behavior  is  system-dependent),
  which  means  the  command flow may continue even if a task fails, re-
  sulting in later troubles.

Task location:
  If no directory is specified in the task name, the task is assumed  to
  be in the GILDAS_LOCAL: (if defined). Else it is searched in TASK#DIR:
  default locations.

Asynchronous execution:
  The task execution may be synchronous (the main  program  waiting  for
```

task completion) or asynchronous (control returns to the main program immediately). Note that an error status issued by the task does not stop the main program in the case of asynchronous execution. If the task terminates before you exit from the activating program, a termination message will be typed on the terminal, giving the termination status.

Remote execution:
Task may execute on a remote node rather than on the local machine. The node name is controlled by logical name GILDAS_NODE. If GILDAS_NODE = LOCAL, local execution is performed. If not, GILDAS_NODE must be the node name of the computer on which execution will be performed. No synchronisation is offered for remote execution.

7.5 SPY

[VECTOR\]SPY [Task_Name]

Displays the status of all active GILDAS tasks, or lists the last output from the specified task.

7.6 SUBMIT

[VECTOR\]SUBMIT Task_Name [Parameter_File] [/EDIT] [/NOWINDOW] [/WINDOW]

The SUBMIT command is similar to the RUN command, except that the Task is submitted to a batch queue (named GILDAS_BATCH) instead of being executed as a detached process. See RUN command for details.

7.7 TRANSPOSE

[VECTOR\]TRANSPOSE Input Output Order

Perform tranposition of disk data files (GDF or FITS) or SIC variables.

Disk data files:

GDF or FITS files storing tables or N-dimensional cubes are transposed to GDF files according to the transposition order specified by Order (21, 312, 213, etc...). For example:
VECTOR\TRANSPOSE TEST.VLM TEST.LMV 231
The transposition takes place in RAM memory if the sic logical SPACE_GILDAS (in MegaBytes unit) is big enough, else it takes place on disk with more computation and disk access costs.

SIC variables:

For example

```
    DEFINE REAL A[10,2] B[2,10]
    VECTOR\TRANSPOSE A B 21
```

will transpose the A SIC variable to the B SIC variable. SIC variables must be of the same type. If A and B are IMAGE variables, the header is also transposed accordingly.

8 TASK Language Internal Help

8.1 Language

TASK\ Command Language Summary

Define and specify the input parameters of tasks. Only available in a Library mode, not to be used interactively.

```
CHARACTER :  Define a character string parameter
FILE      :  Define a filename
GO        :  Activate the task
HELP      :  Show the help
INTEGER   :  Define an integer parameter
LOGICAL   :  Define a logical parameter
MORE      :  Show a separator in a widget
REAL      :  Define a real parameter
VALUES    :  Define a real parameter through a variable length expression
WRITE     :  Write parameter value to task parameter file.
```

8.2 CHARACTER

```
TASK\CHARACTER "Prompt text" Name = [Value] [/CHOICE List]
```

Define the parameter Name of Character type (C*256), and assign it a value if specified. Otherwise, prompt with the text for a value.

A list of predefined values can be provided with the option /CHOICE. If a wildcard (*) is present in the list, a custom value can also be set by the user.

8.3 FILE

```
TASK\FILE "Prompt text" Name = [Filename] [/OLD] [/NEW]
```

Define the parameter Name of Character type, and assign it a filename. Otherwise, prompt with the text for a content. In Windows mode, launch a file browser to get the filename.

NOT IMPLEMENTED: if option /OLD (resp. /NEW) is present, the file must (resp. must not) already exist.

8.4 GO

```
TASK\GO
```

In a .init file, activate the Widget if in Windows mode, or finish parameter entry and activate the task checker.

In a .check file, launch the Task.

```
 .\"=====================================================================
```

8.5 INTEGER

```
   TASK\INTEGER  "Prompt text"  Name[Ndim]  =  [Value  ..  Value_Ndim]
[/RANGE]
  [/INDEX String1 ... StringN [*]] [/CHOICE Value1 ... ValueN [*]]
```

Define the parameter Name of Integer type and dimension Ndim (or scale), and assign it a value if specified. Otherwise, prompt with the text for value(s). Both I*4 and I*8 parameters are supported by this command.

The /RANGE option restrict the valid range for the values. In Windows mode, it activates a slider.

A list of character strings can be provided with the option /INDEX. This /INDEX option means to return the rank of the given string among the available choices. A * as the last choice means any other value is also valid.

A list of predefined values can be provided with the option /CHOICE. If a wildcard (*) is present in the list, a custom value can also be set by the user.

8.6 LOGICAL

```
TASK\LOGICAL "Prompt text" Name = [YES|NO]
```

Define the parameter Name of Logical type, and assign it a value if specified. Otherwise, prompt with the text for a content.

8.7 MORE

```
TASK\MORE
```

8.8 REAL

```
TASK\REAL "Prompt text" Name[Ndim] = [Value .. Value_Ndim] [/RANGE]
[/SEXAGESIMAL] [/CHOICE Value1 ... ValueN [*]]
```

Define the parameter Name of Real type and dimension Ndim, and assign it a value if specified. Otherwise, prompt with the text for value(s). Both R*4 and R*8 parameters are supported by this command.

The /RANGE option restrict the valid range for the values. In Windows mode, it activates a slider.

If option /SEXAGESIMAL is present, a sexagesimal string can be provided: it will be converted to a floating point value.

A list of predefined values can be provided with the option /CHOICE. If a wildcard (*) is present in the list, a custom value can also be set by the user.

8.9 VALUES

```
TASK\VALUES "Prompt text" Name = [Value1 ... ValueN]
```

Define the parameter Name of Character type, and assign it a string holding all values specified as further arguments. Values can be math expression and will be converted to numerics before assignment. This command allows to specify variable dimension numeric arrays as task parameters.

8.10 WRITE

```
TASK\WRITE Name
```

This command is only valid in a .check file. It writes the parameter Name on one line in the .par file, followed by its content in the next lines.

9 ADJUST Language Internal Help

9.1 Language

```
                    ADJUST\ Language Summary

ADJUST : Allows fitting arbitrary data with up to 26 parameters.
EMCEE  : Compute an EMCEE Markov Chain
ESHOW  : Show results from an EMCEE Markov Chain
```

9.2 ADJUST

```
[ADJUST\]ADJUST My_Data "My Command" [/START G1 G2 ... Gn] [/STEP S1 S2 .. Sn] [/EPSILON e] [/WEIGHTS w] [/METHOD Powell|Robust|Simplex|Slatec|Anneal] [/ITER Niter] [/QUIET] [/ROOT_NAME Name] /PARAMETERS V1 V2 ... Vn [/BOUNDS Va Lowa Higha [Vb Lowb Highb ...]]
```

```
Allows fitting arbitrary data with (almost) arbitrary number of  parame-
ters (well, less than 26 so far, I believe).

The  command  ADJUST  defines the following global SIC variables to hold
its results:
       Root%PAR    : Fit parameters
       Root%ERRORS : The parameter errors, if they have been computed
       Root%RES    : Residuals (i.e. My_Data-Root%FIT)
       Root%STATUS : .FALSE. on successful completion.
where Root is the name given in the /ROOT_NAME option (default is ADJ or
MFIT).

My_Data is a SIC variable containing your data to be fitted. It can be a
variable of any rank, but (currently) it must be of type DOUBLE.

"My Command" is a character string specifying the command to be executed
to  compute  the difference between My_Data and the model.  This differ-
ence must be returned in SIC variable Root%RES. The command is typically
        "@ my_difference"
where "my_difference" is a procedure containing the code to compute  the
model and a line to compute Root%RES.  Here is a simple example to fit a
Gaussian (see HELP ADJUST EXAMPLE for more details)
!
begin procedure my_difference
   let ADJ%res my_data-gauss(xx,amp,pos,width)
end procedure my_difference
```

9.2.1 ADJUST Example

```
    Here is a complete procedure to generate a noisy Gaussian and fit it
with ADJUST. The call sequence is

  @ gagemo:demo-adjust Amp Pos Width Noise

where  Amp  is  the desired amplitude, Pos the position, Width the width
and Noise the noise on the data.

! Procedure gag_demo:demo-adjust
!
! An example of ADJUST usage
! For simplicity, let us define the GAUSSIAN function
 define function gauss(x,a,b,c) a*exp(-0.5*((x-b)/c)^2)
!
! The Gaussian parameters, Amplitude, Position and Width
 define double amp pos width /global
! Now the data...
```

```
 define integer nx
 let nx 512
 define double xx[nx] my_data[nx] /global
! Define the Measurement points XX
let xx[i] 10*(i-nx/2)/nx
!
! Fill the data with some noisy values. In reality, this may come from
! an experiment or observation
let my_data gauss(xx,&1,&2,&3)
let my_data[i] my_data+&4*noise(i/i)
!
begin procedure my_difference
   let ADJ%res my_data-gauss(xx,amp,pos,width)
end procedure my_difference
!
! Minimize
adjust my_data "@ my_difference" /start 3 2 1 /par amp pos width /root ADJ
examine ADJ%PAR
examine ADJ%ERRORS
```

9.2.2 ADJUST /EPSILON

This option is used to specify the desired tolerance. Its interpretation is method dependent. For SIMPLEX and POWELL, it means the relative deviation of the mean squared difference of two fit iterations. For SLATEC, which uses non-reduced chi-2, it is the absolute difference between two iterations. For ANNEAL, its interpretation is totally different.

Use default value 0 to let the program guess. In subtle cases, or to gain speed in case a coarse result is desired, use values around 1D-5 for SIMPLEX and POWELL, and of order 1 for SLATEC (provided your weights are really 1/sigma^2...). For ANNEAL, see details in the /METHOD option.

Note that this is not the absolute error of the fit parameters.

9.2.3 ADJUST /METHOD

ANNEAL Simulated annealing technique. This may require a very large number of function evaluation.

POWELL Gradient using the Powell method with the Davidon-Fletcher variable metric.

SIMPLEX Classical Simplex amoeba search

ROBUST A combination of Simplex + Slatec, to be used with large enough initial steps.

SLATEC Modified Levenberg Marquardt method with adaptive steps, as implemented in the Slatec (LINPACK) library

All methods require a proper choice of initial values and steps. The ANNEAL method is much more robust against poor guesses, but may require 10 to 100 times more function evaluations than any other. It is also difficult to control.

9.2.4 ADJUST /START

[ADJUST\]ADJUST My_Data "My Command" /START G1 G2 ... Gn

G1 G2 .. Qn are used to pass starting guesses for the parameters P1 to Pn. The default is 1.0. The starting values should not be too far from a potential solution, otherwise the convergence may not be possible.

9.2.5 ADJUST /STEP

[ADJUST\]ADJUST My_Data "My Command" /STEP S1 S2 .. Sn

S1 S2 .. Sn are used to pass the unity vectors (steps) for the iteration on parameters P1 to. Pn. The default depends on the starting guesses Gi: Si is equal to abs(0.1 x Gi) or 0.1 if Gi is 0.0 . Poor choice of initial steps may lead to non convergence. Too small steps will not help converging when one starts too far from the solution. Too large steps may lead to incorrect evaluation of the parameter errors. The optimal step for the determination of the errors is about the error bar, so that all parameters become dimensionless.

9.2.6 ADJUST /PARAMETER

[ADJUST\]ADJUST My_Data "My Command" /PARAMETER Aname Bname Cname [...]

Defines the parameter names. The parameter names are Global SIC variables which are then used in the user supplied command to compute the model. The variables must exist: they are not created by the ADJUST command.

9.2.7 ADJUST /QUIET

Requires ADJUST to be silent (useful to avoid too many messages in loops).

9.2.8 ADJUST /WEIGHTS

[ADJUST\]ADJUST My_Data "My Command" /WEIGHT My_weights

Define the weights of each data. The array My_Weights must be real, and of same size as the array My_Data.

9.2.9 ADJUST /BOUNDS

[ADJUST\]ADJUST My_Data "My Command" /PARAMETER Aname Bname Cname [...] /BOUNDS Name1 Low1 High1 [Name2 Low2 High2 ...]]

Specify the search boundaries for some parameters. This can be useful to prevent the minimization to enter non physical regions. The code does not (yet) support one sided boundaries such as [Low, +Inf[

Caution:
/BOUNDS is still experimental and may affect the values of the error bars. Please check the result using unbounded minimization once a minimum has been found with a bounded search.

9.3 EMCEE

[ADJUST\]EMCEE [Data Command] [/BEGIN] [/BOUNDS Par Low Up [...]] [/LENGTH Length] [/PARAMETERS Va Vb Vc ...] [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk] [/ROOT_NAME Root]

****
Foreword: this command is a wrapper around the "emcee" Python module. Gildas must have been compiled with the Gildas-Python binding enabled, and the modules "emcee" and "pickle" must be installed in your Python version.
****

Prepare, begin or continue a Monte Carlo Markov Chain using the EMCEE method.

Command is the command line which computes a Chi^2 function on your data. Typically, it is "@ my_chi2" where my_chi2 is a script gathering all necessary computations. The resulting chi^2 must be put into the global variable EMCEE%CHI2

Argument Data is unused so far.

The command has 3 forms. Without argument, it can accept only options /BEGIN and /LENGTH. It then begins or continues a previously defined MCMC chain. With arguments, the /PARAMETER option must be present. The EMCEE command then defines all parameters for the MCMC chain, and optionally starts it if /BEGIN is present.

9.3.1 EMCEE Caution

- Contrary to MFIT or ADJUST, EMCEE does ***NOT*** return best fit val-

ues. This can be done a posteriori using the ESHOW command.

- EMCEE cannot be interrupted by ^C. While EMCEE will react to ^C, it will not stop properly. You may no longer be able to continue afterwards, and you may have to kill the whole process. This is because it uses an intrincate nesting of SIC and Python commands and macros.

9.3.2 EMCEE Credits

The EMCEE method uses the affine-invariant ensemble sampler proposed by Goodman & Weare (2010). The implementation is based on the "emcee" Python implementation developped by Foreman Mac-Key et al 2012, and is only available with the Python binding of SIC.

9.3.3 EMCEE Example

The following demo will fit a Gaussian. Amp must be in range [5,15], Pos in range [-1,1] and Width in range [0,3].

```
!
! @ gag_demo:demo-emcee  Amp Pos Width Noise
!
! An example of EMCEE usage
! For simplicity, let us define the GAUSSIAN function
define function gauss(x,a,b,c) a*exp(-0.5*((x-b)/c)^2)
!
if .not.exist(my_data) then
  ! The Gaussian parameters, Amplitude, Position and Width
  define double amp pos width /global
  ! Now the data...
  define integer nx
  let nx 512
  define double my_xx[nx] my_data[nx] my_noise /global
  ! Define the Measurement points XX
  let my_xx[i] 10*(i-nx/2)/nx
endif
!
! Fill the data with some noisy values. In reality, this may come from
! an experiment or observation
let my_noise &4
let my_data gauss(my_xx,&1,&2,&3)
let my_data[i] my_data+my_noise*noise(i/i)
!
! Start with some random value
let amp &1*(1+noise(my_noise))
let width &3*(1+noise(my_noise))
let pos &2+noise(my_noise)
```

```
!
begin procedure my_chi2
  define double res /like my_data
  let res my_data-gauss(my_xx,amp,pos,width)
  let res (res/my_noise)^2
  compute emcee%chi2 sum res ! The Chi^2 must be returned here
  @ emcee-timer ! Optional: report advances in the chain
end procedure my_chi2
!
emcee my_data "@ my_chi2" /start 'amp' 'pos' 'width' -
      /par amp pos width /bound amp 5 15 pos -1 1 width 0 10 -
      /step 0.02 0.02 0.02 /length 100 /walk 4 /begin
for i 1 to 4
  emcee
next
!
```

9.3.4 EMCEE /BEGIN

[ADJUST\]EMCEE Data Command /BEGIN [/BOUNDS Par Low Up [...]] [/LENGTH Length] /PARAMETERS Va Vb Vc ... [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk] [/ROOT_NAME Root]

or

[ADJUST\]EMCEE /BEGIN [/LENGTH Length]

Start a Monte Carlo Markov Chain using the EMCEE method. The short form uses parameters defined in a previous EMCEE command.

9.3.5 EMCEE /BOUNDS

[ADJUST\]EMCEE Data Command [/BEGIN] /BOUNDS Par Low Up [...] [/LENGTH Length] /PARAMETERS Va Vb Vc ... [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk]

Specify the parameter bounds for some or all parameters. If not specified, the bounds are derived from +/- 10 times the parameter step.

9.3.6 EMCEE /LENGTH

[ADJUST\]EMCEE [Data Command] [/BEGIN] [/BOUNDS Par Low Up [...]] /LENGTH Length [/PARAMETERS Va Vb Vc ...] [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk]

Specify the length of the MCMC chain. The total number of executions will be the Length times the number of parameters times the number of walkers per parameter. Length is the only control allowed to change when restarting a MCMC chain.

9.3.7 EMCEE /PARAMETERS

[ADJUST\]EMCEE Data Command [/BEGIN] [/BOUNDS Par Low Up [...]] [/LENGTH Length] /PARAMETERS Va Vb Vc ... [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk]

Specify the parameter names. These must be global SIC variables, so that they can be used the the "Command" computing the Chi^2.

9.3.8 EMCEE /ROOT_NAME

[ADJUST\]EMCEE Data Command [/BEGIN] [/BOUNDS Par Low Up [...]] [/LENGTH Length] /PARAMETERS Va Vb Vc ... [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk] /ROOT_NAME RootName

where Root is the name given in the /ROOT_NAME option (default is ADJ or MFIT).

Specify the root name for the results. Results are ***NOT*** computed by the EMCEEE command, but by the ESHOW command (ESHOW Errors or ESHOW Results). They will be returned in RootName%PAR and RootName%ERRORS, like for commands MFIT or ADJUST.

9.3.9 EMCEE /START

[ADJUST\]EMCEE Data Command [/BEGIN] [/BOUNDS Par Low Up [...]] [/LENGTH Length] /PARAMETERS Va Vb Vc ... /START A1 B1 C1 ... [/STEP A2 B2 C2 ...] [/WALKERS NWalk]

Specify the starting values for the parameters (in the same order as the parameter names).

9.3.10 EMCEE /STEP

[ADJUST\]EMCEE Data Command [/BEGIN] [/BOUNDS Par Low Up [...]] [/LENGTH Length] /PARAMETERS Va Vb Vc ... [/START A1 B1 C1 ...] /STEP A2 B2 C2 ... [/WALKERS NWalk]

Specify the random steps for the parameters (in the same order as the parameter names). If no bound is specified, the Lower and Upper bounds are +/- 10 times the step away from the starting value.

9.3.11 EMCEE /WALKERS

[ADJUST\]EMCEE Data Command [/BEGIN] [/BOUNDS Par Low Up [...]] [/LENGTH Length] /PARAMETERS Va Vb Vc ... [/START A1 B1 C1 ...] [/STEP A2 B2 C2 ...] [/WALKERS NWalk]

Specify the number of walkers per parameter.

9.4 ESHOW

[ADJUST\]ESHOW AUTOCORR|CHAINS|ERRORS|TRIANGLES [Args ...] [BURN Burn] [/SPLIT]

Show some results of an EMCEE Markov Chain. Except for ESHOW RESULTS and ESHOW ERRORS, GreG must be available for this action. (Trick: if it is not, just use command IMPORT MAPPING, and it will be...)

9.4.1 ESHOW AUTOCORR

[ADJUST\]ESHOW AUTOCORR [Length] [/SPLIT] [/BURN Burn]

Plot the time correlation for the parameters, and compute the autocorrelation length up to the specified Length (default 100). "Burn" is the number of steps which must be ignored at the beginning of the EMCEE chain before performing the action. Default is 50. /SPLIT option will compute this for each chain instead of globally.

9.4.2 ESHOW CHAINS

[ADJUST\]ESHOW CHAINS [Last]

Plot the MCMC chains. Number indicates the last iteration to be displayed.

9.4.3 ESHOW ERRORS

[ADJUST\]ESHOW ERRORS [/BURN Burn]

Print on screen the most likely values and their (asymmetric) errors. "Burn" is the number of steps which must be ignored at the beginning of the EMCEE chain before performing the action. Default is 50.

Results (median of the distribution) and Errors (68 percentiles) are also returned by this command, in variables RootName%PAR and RootName%ERRORS (where RootName is the name specified in option /ROOT_NAME of command EMCEE), like for commands MFIT or ADJUST. The original variables specified in the /PARAMETER option of command EMCEE are also set to their median values.

9.4.4 ESHOW RESULTS

[ADJUST\]ESHOW Results [Filename] [/BURN Burn]

Print the results and errors (symmetric and asymmetric) errors in a file. "Burn" is the number of steps which must be ignored at the beginning of the EMCEE chain before performing the action. Default is 50.

Results (median of the distribution) and Errors (68 percentiles) are also returned by this command, in variables RootName%PAR and RootName%ERRORS (where RootName is the name specified in option /ROOT_NAME of command EMCEE), like for commands MFIT or ADJUST. The original variables specified in the /PARAMETER option of command EMCEE are also set to their median values.

The created script is called 'Filename'.sic (default results.sic) and when executed, will print the results on screen in the same format as the ESHOW Errors command. The script can be customized by re-defining symbol emcee_result (which by default executes "@ emcee-error").

9.4.5 ESHOW TRIANGLES

```
[ADJUST\]ESHOW TRIANGLES [Parameter] [/BURN Burn]
```

Plot the 2x2 correlation surfaces for all parameters. If a Parameter name is given, only plot the correlation with this parameter.

"Burn" is the number of steps which must be ignored at the beginning of the EMCEE chain before performing the action. Default is 50.

9.4.6 ESHOW /BURN

```
[ADJUST\]ESHOW Action [Parameter] /BURN Burn
```

Indicate the size of the "burn-out" region, i.e. how many steps must be ignored at the beginning of the chains. Default is 50.

9.4.7 ESHOW /SPLIT

```
[ADJUST\]ESHOW AUTO [length] [/BURN Burn] /SPLIT
```

Plot the time correlation for the parameters, and compute the autocorrelation length for each chain.

10 SIC Error Messages and Recovery Procedures

SIC may output a number of error messages. These are usually self explanatory, and most of them refer to typing errors, or to an unanticipated degree of complexity reached during the program execution (too many loops, macros, complex mathematic formulas, etc...). More severe errors may appear, usually due to internal logic errors in the calling program. SIC is a relatively safe program. However, its very flexible possibilities, and in particular the possibility of calling it as a command monitor in multi-language application, possibly written independently by different programmers, make it very difficult to be error free. This section list all the error messages written by SIC, and some (but not all) information messages. The format of a SIC message is the following

```
C-FACILITY, Explanation text
```

C is a letter indicating the severity of the message, and may be :

`I` for Information

`W` for Warning. Normal execution can proceed, but the operation was not completed.

`E` for Error. Something really went wrong, and a corrective action should (usually) take place. Suggested actions are mentioned.

`F` for Fatal Error. This is a programming error, either in SIC, or in the calling program, or an unrecoverable error causing a program abort (such as a `PAUSE` in batch sessions).

`FACILITY` is a mnemonic of the subroutine or of the command where the error occured.
`"Explanation text"` is a concise but usually self explanatory message.

In case of Fatal errors, the "recovery procedure" usually indicates to "Submit an SPR". An SPR is a *Software Performance Report*, and it should be sent to the SIC authors, by E-Mail at

```
gildas@iram.fr
```

(`GILDAS` is a reserved account for all GILDAS software).

List of Error Messages

10.1 A through C

`E-COMPUTE, TRANSPOSE not implemented`
`SIC, COMPUTE` command. Self explanatory...
User action : If your variables are images, use the `VECTOR\TRANSPOSE` command instead.

`E-COMPUTE, Variable cannot be written`
`SIC, COMPUTE` command. The output variable is defined ReadOnly.
User action : Probably a user error. Check your variable name. If it is what you wanted, it means you are trying to overwrite a protected program defined variable, and this is forbidden of course.

`E-COMPUTE, Variable must be Real`
`SIC, COMPUTE` command. The required actions operate only on Real variables.
User action : Define intermediate variables if necessary.

`E-COMPUTE, Invalid OUTPUT variable dimensions`
`SIC, COMPUTE` command on Fast Fourier Transform action.
User action : See Help.

10.2 D

`E-DECODE, Invalid arithmetic expression`
`SIC,` Argument decoding routines. An arithmetic expression used as argument is invalid. The message is usually preceded by more detailed explanation.
User action : correct the expression. If a message "Internal logic error" appeared, submit an SPR.

`E-DECODE, Invalid logical expression`
`SIC,` Argument decoding routines. A logical expression used as argument is invalid. The message is usually preceded by more detailed explanation.
User action : correct the expression. If a message "Internal logic error" appeared, submit an SPR.

`E-DECODE, Error computing <String>`
`SIC`, Argument decoding routines. Some error occured during evaluation of a valid arithmetic expression. A more detailed text precedes this message. This is usually due to undefined variables, or arithmetic errors like square root of negative values.
User action : correct any typing error. Check values of variables if an arithmetic error occured.

`E-DECODE, Option <Integer> or argument <Integer> out of bounds`
`SIC`, Argument decoding routines. This is a programming error : a command required too many arguments or options.
User action : Notify the programmer who should consult the SIC programming manual.

`E-DECODE, You have overwritten the command line pointers.`
`SIC`, Argument decoding routines. This is a programming error: the program tries retrieving an argument after another command line has been analysed. This frequently occurs when

GREG is called in library mode by another program. This should only be done AFTER all arguments have been retrieved.
User action : Notify the programmer who should retrieve all needed arguments before he starts analysing another line.

`E-DECODE, Missing argument number <Integer> of Command <String>`

`E-DECODE, Missing argument number <Integer> of Option <String>`
`SIC`, Argument decoding routines. The specified argument is mandatory for the command or option.
User action : specify the missing argument.

`E-DEFINE, Cannot specify dimension for existing images`
`SIC, DEFINE or LET /NEW` commands. Dimensions can only be specified when creating an image.
User action : Don't specify a dimension for existing images.

`E-DEFINE, Invalid variable name <String>`
`SIC, DEFINE or LET /NEW` commands. Variable names must be less than 15 characters and begin with a letter.
User action : choose a valid name.

`E-DEFINE, Invalid status <String>`
`SIC, DEFINE HEADER` command. The header status can only be Read or Write.
User action : May be you confused `DEFINE IMAGE` and DEFINE HEADER. Correct your typing.

`E-DEFINE, Memory allocation failure`
`SIC, DEFINE or LET /NEW` commands. The memory needed to create the variable could not be obtained from the operating system, due to a shortage of system resources or quota. This message is may be preceded by an operating system error message. On a typical site, this error will only occur if you are using (very) big arrays.
User action : delete any unused variable, clear the plot if any, and then retry. Try to use *images* instead of *arrays*. If this does not work, exit the program, reenter it and retry. If this is not sufficient, consider whether you *really* need such big arrays. If the answer is yes, you might consider asking your system manager to increase the relevant quota.

`E-DEFINE, Missing dimension of new image`
`SIC, DEFINE` command. The dimension of a new image must be specified
User action : specify the dimension.

`E-DEFINE, Only last dimension can be extended`
`SIC, DEFINE` command. The `EXTEND` request is not acceptable.
User action : see `DEFINE IMAGE` internal help.

`E-DEFINE, Syntax error`
`SIC, DEFINE FUNCTION` command. The function definition is invalid.

`E-DEFINE, Too many variables`
`SIC, DEFINE or LET /NEW` commands.
User action : delete some existing variables, or use them instead of defining a new one. If you cannot, ask your system manager about increas SIC workspace.

`E-DEFINE, Too many arguments`
`SIC, DEFINE FUNCTION` command. Invalid syntax: only one function definition at a time is possible
User action : use several `DEFINE FUNCTION` commands if necessary.

`E-DEFINE, Variable <String> already exists`
`SIC, DEFINE or LET /NEW` commands. The specified name already is a known variable.
User action : use another name, or use (with command `LET` the already defined variable if you believe you may do so.

`E-DELETE, Incompatible options`
`SIC, DELETE` command. The options `/FUNCTION`, `/SYMBOL` and `/VARIABLE` are mutually exclusive.

`E-DELETE, Missing option`
`SIC, DELETE` command. One of the options `/FUNCTION /SYMBOL` and `/VARIABLE` must be present.

`E-DELETE, No such variable <String>` `SIC, DELETE` command. The specified variable cannot be deleted because it does not exist.

`E-DELETE, Variable <String> not deleted` `SIC, DELETE` command. The variable could not be deleted, because it is program defined
User action : May be a typing error. Check the variable name.

`E-DIMENSION, Invalid dimension <string>`
Any command with a numerical argument. In the present version of SIC indexes of arrays can only be constants or scalar numerical variables. Complex numerical expressions are not allowed.
User action : use an intermediate variable.

`E-DIMENSION, Invalid mixture of implicit and explicit dimensions`
`SIC, LET` command. Implicit loops on arrays cannot be mixed with explicit indexes for other dimensions.
User action : either use an explicit `FOR-NEXT` loop, or rearrange your expression to use the implicit loop (which is *much* faster).

`E-DIMENSION, Invalid string length <Number>`
`SIC, DEFINE or LET /NEW` commands. Character variable is not positive.

`E-DIMENSION, Invalid variable name <string>`
`SIC, DEFINE or LET /NEW` commands. Variable names must be less than 15 characters and begin with a letter.
User action : choose a valid name.

`E-DIMENSION, Missing character size`
`SIC, DEFINE or LET /NEW` commands. The size of character string must be specified.
User action : specify a length.

`E-DIMENSION, Missing closing bracket`
Any command with a numerical argument. An opening bracket is not matched with the corresponding closing bracket.
User action : add the appropriate closing bracket.

`E-DIMENSION, Too many dimensions`
`SIC, DEFINE or LET /NEW` commands. Only 4 dimensions are supported.
User action : decrease the number of dimensions.

10.3 E

`I-EDIT, Using <String> editor`
`SIC, EDIT` command with argument, information message. The specified editor is called to edit the file specified as argument. Control will return to SIC after the editing session.

`I-EDIT, Writing stack content on STACK.Ext`
`SIC, EDIT` command without argument, information message. `STACK.Ext`, where `Ext` is the current macro extension, will be edited using current editor.

`E-EDIT, File name too long`
`SIC, EDIT` command. The corresponding file cannot be edited.

`E-ELSE, Invalid argument <String>`
`SIC, ELSE` command was followed by an invalid first argument.
User action : correct typing error. `ELSE` can only have no argument or `IF` as first argument.

`E-ELSE IF, Invalid syntax`
`SIC, ELSE` command with argument `IF`. The only accepted syntaxes for `ELSE IF` are:
`ELSE IF <logical expression>`
and
`ELSE IF <logical expression> THEN`

`I-ERROR, occured in <String> at line <Integer>`

`I-ERROR, occured in Loop <Integer> (<Real>) at Line <Integer>`
`SIC`, error traceback facility. The message contains traceback of an execution error while executing nested macros, stack or loops. Macro names, loop number and index values are given together with the lines being executed.
User action : If a `PAUSE` occurred correct the erroneous line, execute it and continue the nested macros execution by command `CONTINUE`, or abort execution by command `SIC\QUIT`. If an error recovery command is active, it has been automatically executed before resuming the nested macros execution.

`I-ERROR, occured in Program`
`SIC`, error traceback facility. An error occurred in subroutine mode.
User action : signal the error to the programmer.

`I-ERROR, occured in Error recovery mode`
`SIC`, error traceback facility. An error occurred in the error recovery command or procedure. A `PAUSE` is issued.
User action : correct the erroneous recovery procedure and resume execution.

`F-EVALUATE, Invalid precision <Integer>`
Any command with a numerical argument. This is an internal logic error in the arithmetic processor (or a memory error on your machine!).
User action : Please submit an SPR. If you need the result, try toF modify your expression (reorder, use intermediate variables...).

`F-EVALUATE, Internal logic error`
Any command with a numerical argument. This is an internal error in SIC.
User action : submit an SPR. You can try going around the error by modifying your expression.

`E-EXAMINE, Undefined variable <String>`
`SIC, EXAMINE` command with an argument. The specified variable does not exist.
User action : check for typing errors.

`F-EXAMINE, Invalid data format, internal logic error`
`SIC, EXAMINE` command.
User action : Submit an SPR.

`W-EXAMINE, No known variable`
`SIC, EXAMINE` command without argument. No variable has been defined yet.

10.4 F

`E-FOR, Empty list`
`SIC, FOR` command. The compilation mode is not entered.
User action : Reenter `FOR` command with a list of values.

`E-FOR, Incomplete list :`
`SIC, FOR` command. The invalid list is typed with a pointer to the error.
User action : correct the error.

`E-FOR, Input level too deep`
`SIC, FOR` command. The execution level is too high, too many macros or loops are nested, the loop cannot be executed.
User action : avoid so deeply nested situations by concatenating all macros in a single one instead of nesting them. Deep nesting (more than 8 execution levels) is almost invariably unnecessary.

`E-FOR, Invalid loop :`
`SIC, FOR` command. The invalid list is typed with a pointer to the error.
User action : correct the error.

`E-FOR, Logical expression is too long`
`SIC, FOR` command with `/WHILE` option. The logical expression specified in the `/WHILE` option is too long to be stored and evaluated.
User action : make it simpler using intermediate variables.

`I-FOR, Loop <Integer> has finished`
`SIC, VERIFY` mode. Information message

`I-FOR, Loop <Integer> is running with index <Real>`
`SIC, VERIFY` mode. Information message.

`W-FOR, Loop <Integer> compilation aborted`
`SIC`, loop compilation mode. This message is typed after a `QUIT` command has been typed to abort a loop compilation.

`E-FOR, Loop buffer overflow`
: `SIC`, loop compilation mode. Too many commands were entered in the loop buffers. Loop compilation failed.
User action : Put the remaining commands in a macro, and execute the macro within the loop.

`W-FOR, Line not valid in this context, ignored`
: `SIC`, loop compilation. The user attempted to insert an invalid command (such as `HELP`, `EDIT`) into a loop buffer.
User action : such commands cannot be placed in loops.

`W-FOR, No variable or list`
: `SIC, FOR` command. Either the loop variable or the list of values is missing.
User action : Correct the command line.

`E-FOR, Only <Integer> levels of FOR - NEXT loops`
: `SIC, FOR` command. The user attempt to nest too many loops.
User action : find another solution to your problem than nesting so many loops.

`E-FOR, Syntax error in list :`
: `SIC, FOR` command. The invalid list is typed with a pointer to the error.
User action : correct the error.

`E-FOR, Too many arguments in list`
: `SIC, FOR` command. The `FOR` list is too long.
User action : Run two (or more) consecutive loops with part of the list to span all values in your original list.

`E-FUNCTION, Invalid function name <string>`
: `SIC`, function definition routine. A user program defined function has an invalid name. Function names are limited to 15 characters and must begin with a letter. This is a programming error.
User action : notify the programmer.

`F-FUNCTION, SIC is not loaded`
: `SIC`, function definition routine. The program tries to define a function before SIC has been initialized. This is a programming error.
User action : notify the programmer.

`E-FUNCTION, Too many arguments to function`
: `SIC`, function definition routine. A user program defined function has too many arguments. Function cannot have more than 4 arguments. This is a programming error.
User action : notify the programmer.

`E-FUNCTION, Too many functions`
: `SIC`, function definition routine. The program attempts to define too many user functions. This is a programming error.
User action : notify the programmer who should contact the authors if he really needs so many functions.

`E-FUNCTION, <Integer> arguments to function <String>`
`SIC`, function definition routine. A user defined function has a negative number of arguments. This is a programming error.
User action : notify the programmer.

10.5 G

`E-GETCOM, Line too long, buffer overflow`
`SIC`, command reading routine. The command line is too long for the internal buffer.
User action : make it shorter if possible. If not, ask the programmer to increase the line buffer size and relink the program. (Maximum line length in SIC is 2048 characters, so we hardly think you may be limited by this).

`E-GETCOM, Read error on macro file, unit <Number>`
`SIC`, command reading routine. An error occured while reading command from a macro.
User action : this is most likely due to a hardware problem, unless you are attempting to read a binary file... Check the macro. Type the macro to see what command could not be read, execute it and resume macro execution.

10.6 H

`W-HELP, Error opening <String>`
`SIC, HELP` command. The Help file for a language does not exist.
User action : check with the programmer or system manager that all logical names have been correctly defined.

`W-HELP, Language <String> is in library only mode`
`SIC, HELP` command. The specified language is in library mode, no help is available for it.
User action : commands from this language cannot be accessed interactively, except by specifying the full language name.

`W-HELP, No help for <String>`
`SIC, HELP` command. The specified command ¡String¿ is not documented.
User action : unless you are quite sure of their behaviour (or you like risks) avoid using those undocumented commands.

10.7 I

`E-IF, Invalid syntax`
`SIC, IF` command. The second argument of the command (if present) was not `THEN`.

`E-IMPLICIT, Invalid variable name <string>`
`SIC, LET` command. A variable used in the implicit loop has an invalid name. Variable names must be shorter than 15 characters and begin with a letter.
User action : Use a valid variable name.

`E-IMPLICIT, Too many variables`
`SIC, LET` command. The total number of variables defined exceeds the SIC limit. Temporary variables used in implicit loops are included in this count.
User action : Delete a few useless variables and retry.

`E-IMPLICIT, Variable already exist`
`SIC, LET` command. This is an internal logic error.
User action : Submit an SPR. While waiting for the correction, you may try modifying your expression.

`E-INCARNATE, Bad incarnation type`
`SIC`, type conversion routine. The type conversion routine was called with a non numeric output type. This is a programming error.
User action : Notify the programmer.

`E-INCARNATE, Bad variable type`
`SIC`, type conversion routine. The type conversion routine was called with a non numeric input type. This is a programming error.
User action : Notify the programmer.

`E-INTER, Ambiguous command, could be :`
`SIC`, monitor routine. The command name is ambiguous.
User action : specify more characters or specify language to avoid ambiguities.

`E-INTER, Ambiguous option, could be :`
`SIC`, monitor routine. The option name is ambiguous.
User action : specify more characters.

`W-INTER, No command on line`
`SIC`, monitor routine. A command line only contained the language field.

`E-INTER, Too many words in line`
`SIC`, monitor routine. The user program tried to access more than 100 arguments. This is a programming error.
User action : if you really need so many arguments, submit an SPR.

`E-INTER, No options allowed for command <String>`
`SIC`, monitor routine. The command has no options.
User action : Suppress options from the command line.

`E-INTER, Unbalanced quote count`
`SIC`, monitor routine. A command line has an odd number of double quotes.
User action : Correct typing mistake and reenter command.

`E-INTER, Unknown command`
`SIC`, monitor routine. The command does not exist in any active language.
User action : check spelling or bring more languages in the active scope (Command `SIC\SIC`).

`E-INTER, Unknown command <String> for language <String>`
`SIC`, monitor routine. The command does not exist in the specified language.
User action : check spelling.

`E-INTER, Unknown language <String>`
`SIC`, monitor routine. The specified language is not known.
User action : Check spelling. If good, verify you are using the right program.

`E-INTER, Unknown option <String> for command <String>`
`SIC`, monitor routine. The command has no such option
User action : check spelling.

10.8 J trough L

`F-LET, Cannot assign arrays`
`SIC`, calling program. This is a programming error. The program attempts to assign values to an array through a call to `SIC_LET_REAL` (or `SIC_LET_INTE...`). This is not allowed.
User action : notify the programmer. If you need to assign values to an array, call `SIC_DESCRIPTOR` and do the assignment in your program.

`E-LET, Header structures cannot be assigned`
`SIC, LET` command. The variable to be assigned is a generic header name.
User action : Add the % symbol after the generic header name to assign a header structure.

`E-LET, Invalid attribute <string>`
`SIC, LET /NEW` command. The only recognised attributes for a variable are `GLOBAL` and `LOCAL`.

`E-LET, Memory allocation failure`
`SIC, LET` commands. The memory needed as work space could not be obtained from the operating system, due to a shortage of system resources or quota. This message is preceded by the VMS error message. On a typical site, this error will only occur if you are using (very) big arrays or images.
User action : delete any unused variable, and then retry. If this does not work, exit the program, reenter it and retry. If this is not sufficient, consider whether you *really* need such big arrays. If the answer is yes, you might consider asking your system manager to increase the relevant quota.

`E-LET, Operation not supported on string arrays`
`SIC, LET` command. String arrays cannot be assigned directly.
User action : Define the string arrays element by element, using a loop.

`E-LET, Readonly variables cannot be modified`
`SIC, LET` command. You are attempting to modify a protected variable declared by the program.
User action : this is not allowed. Define another variable if you need temporary storage.

`E-LET, Readonly headers cannot be modified`
`SIC, LET` command. You are attempting to modify a protected header.
User action : This is not allowed. Redefine the header with write access if needed.

`E-LET, Trailing arguments in assignement`
`SIC, LET` command. While trying to use the element by element `LET` command, you omitted or added one argument.
User action : Check the array size, and count the number of arguments.

`E-LET, Undefined header <String>`
`SIC, LET` command. The assigned header is not defined.
User action : Check variable name for typing error.

`E-LET, Undefined variable <String>`
`SIC, LET` command. The assigned variable is not defined.
User action : first check variable name for typing error. If you want to assign a new variable, use `DEFINE` command or option `/NEW` of command `LET` to define it.

`E-LET, Variable type does not match declaration`
`SIC, SIC_LET_xxx` subroutine. This is a programming error. The program attempted to assign a value of wrong type to a defined variable.
User action : notify the programmer.

`E-LOGICAL, Error evaluating <String>`
`SIC`, argument decoding routine. Evaluation of a logical expression failed. This message is usually preceded by a more detailed text indicating why the expression could not be evaluated.
User action : check for undefined variables.

`E-LOGICAL, Invalid logical expression`
`SIC`, argument decoding routine. An invalid logical expression was found (most likely in an `IF, FOR /WHILE or ELSE IF` command). This message is usually preceded by a more detailed text indicating why the expression is invalid.
User action : correct the expression.

10.9 M

`E-MACRO, Input level too deep`
`SIC, @` command. A macro could not be executed due to an execution level too high.
User action : finish some macro execution before activating this one. Eventually, you may need to rearrange your macros to avoid so many execution levels.

`E-MACRO, Recursive call to macro <String>`
`SIC, @` command. A recursive call occurred to the specified macro.
User action : correct the macros which are causing this problem, recursive calls are prohibited.

`E-MACRO, Unable to open macro <String>`
`SIC, @` command. The macro file could not be opened for read.
User action : check for typing error, and possibly for privilege violation. This message is followed by a second line of text indicating a more precise reason.

`E-MATH, Unmatch Closing bracket`
`SIC`, Mathematic and logical expression analysis modules.
User action : Correct the typing error.

`E-MATH, Missing operator after string`
`SIC`, function definition module. The expression is incorrect.
User action : Correct the typing error (misplaced parenthesis normally).

`E-MTH, Arithmetic expression is too complex`
`SIC`, Mathematic and logical expression analysis modules. The expression could not be analyzed because of complexity.
User action : break it in several expression, using intermediate variables.

`E-MTH, Error in FIND_OPERATOR`

`E-MTH, Error in READ_OPERAND`
`SIC`, Mathematic and logical expression analysis modules. These message usually follows more specific ones.
User action : In case the other messages require to submit an SPR, please indicate the complete list of error messages from MTH.

`E-MTH, Comparing arrays of inconsistent dimensions`
`SIC`, Mathematic and logical expression analysis modules. An expression contains an illegal mix of arrays with different dimensions.
User action : You probably got confused with variable names. Correct the expression.

`E-MTH, Comparing non scalar variables`
`SIC`, Mathematic and logical expression analysis modules. A logical expression attempts to compare by order (.GT. .LT. .GE. and .LE.) two arrays. Such comparisons are invalid.
User action : You probably got confused with variable names. Correct the expression.

`E-MTH, Error reading operand <String>`
`SIC`, Mathematic and logical expression analysis modules. The corresponding string could not be analyzed.
User action : Correct anay (likely) typing error.

`W-MTH, Free operand in BUILD_TREE`
`SIC`, Mathematic and logical expression analysis modules. This is an internal logic error. Some work space allocated during evaluation has not been freed correctly. The result is nonetheless correct.
User action : Submit an SPR, with the faulty mathematic formula.

`E-MTH, Inconsistent mixture of Arithmeti, Logical and Character expression`
`SIC`, Mathematic and logical expression analysis modules. The expression is invalid.
User action : Correct your expression (you got confused about variable types, most likely).

`F-MTH, Internal logic error in <String>`
`SIC`, Mathematic and logical expression analysis modules. An expression was successfully analyzed, but could not be evaluated because of an internal error in the analysis modules.
User action : Submit an SPR, with the faulty mathematic formula. Simplify your formula, or add parenthesis to avoid possible ambiguities and try again. Eventually break your formula into several consecutive ones.

`F-MTH, Invalid arithmetic formula`
`SIC`, Mathematic and logical expression analysis modules. The formula is invalid.
User action : Correct it. You may have confused some operators or variables.

`F-MTH, Invalid character string <String>`
`SIC`, Mathematic and logical expression analysis modules. The character string is invalid (empty string, or missing closing quote (″)
User action : Correct it. You may have confused some operators or variables.

`F-MTH, Invalid number of arguments in call to <String>`
SIC, Mathematic and logical expression analysis modules. The function call list is incorrect.
User action : Correct it.

`F-MTH, Invalid syntax`
SIC, Mathematic and logical expression analysis modules. This is an internal logic error. An expression contained only opening parenthesis.
User action : Submit an SPR with the faulty expression.

`E-MTH, Level <Number> should already have been evaluated`
SIC, Mathematic and logical expression analysis modules. This is an internal logic error.
User action : Submit an SPR with the faulty expression.

`E-MTH, Mathematics on arrays of inconsistent dimensions`
SIC, Mathematic and logical expression analysis modules. You are trying to combine arrays with inconsistent dimensions.
User action : Correct the expression.

`E-MTH, Memory allocation failure`
SIC, LET command. Memory needed as work space could not be obtained from the operating system, due to a shortage of system resources or quota. On a typical site, this error will only occur if you are using (very) big arrays.
User action : delete any unused variable, clear the plot if any, and then retry. If this does not work, exit the program, reenter it and retry. If this does not work, try using *images* instead of *arrays*. If this is not sufficient, consider whether you *really* need such big arrays. If the answer is yes, you might consider asking your system manager to increase the relevant quota.

`W-MTH, Missing operand in formula <String>`
SIC, Mathematic and logical expression analysis modules. One operator or function is left without operand after parsing.
User action : Correct the expression.

`E-MTH, Missing operator after closing bracket`
SIC, Mathematic and logical expression analysis modules.
User action : Correct the expression.

`E-MTH, Result type mismatch`
SIC, Mathematic and logical expression analysis modules. You are trying to assign a logical value to a numerical variable or so.
User action : Correct the expression.

`W-MTH, Result was not yet assigned`
SIC, Mathematic and logical expression analysis modules. This is an internal logic error. An expression was successfully analyzed, but could not be evaluated because of an internal error.
User action : Submit an SPR, with the faulty mathematic formula. This is only a warning, and in principle the result should be correct. If not, simplify your formula, or add parenthesis to avoid possible ambiguities and try again. Eventually break your formula into several consecutive ones.

`W-MTH, Scratch operand remaining <integer>`
SIC, Mathematic and logical expression analysis modules. This is an internal logic error. An expression was successfully analyzed, but could not be evaluated because of an internal error.
User action : Submit an SPR, with the faulty mathematic formula. This is only a warning, and in principle the result should be correct. If not, simplify your formula, or add parenthesis to avoid possible ambiguities and try again. Eventually break your formula into several consecutive ones.

`E-MTH, Too many operands`
SIC, Mathematic and logical expression analysis modules. Formula is too complex, and does not fit in the internal buffer.
User action : Simplify your formula, and try again. Use intermediate variables to break your formula into several smaller pieces. If this is a serious limitation to you, submit an SPR, and we will increase the buffer size.

`E-MTH, Too many operands in function call`
SIC, Mathematic and logical expression analysis modules. A function was called with the wrong number of arguments.
User action : Correct the call

`E-MTH, Unknown variable <String>`
SIC, Mathematic and logical expression analysis modules. The parsing module was expecting a variable.
User action : This is presumably due to a typing mistake. Correct your expression.

`E-MTH, Unknown function <String>`
SIC, Mathematic and logical expression analysis modules. The parsing module was expecting a user defined function.
User action : This is presumably due to a typing mistake. Correct your expression.

`E-MTH, Unknown logical or relational operator <String>`
SIC, Mathematic and logical expression analysis modules. The parsing module was expecting an operator.
User action : This is presumably due to a typing mistake. Correct your expression.

10.10 O through R

`E-ON, Unknown argument <String>`
SIC, ON command.
User action : See `HELP ON`.

`E-PARSE, Implicit transposition not yet supported`
SIC, Array dimension parser. The specified array subset is invalid, because it requires an implicit transposition of the array variable.
User action : Read the section upon what array variables.

`E-PARSE, Index <Integer> exceeds dimension <Integer> of <String>`
SIC, Array dimension parser. The specified array subset is invalid, because the arrays size is exceeded
User action : Correct your error.

`E-PARSE, Variable <String> has only '<Integer> dimensions`
SIC, Array dimension parser. The specified array subset is invalid, because the dimension does not exist in the array.
User action : Correct your (typing) error.

`W-PAUSE, <^C> ignored, level too deep`
SIC, monitor routine. The user pressed `<^C>` during a command execution, but the execution level is too high to allow a `PAUSE` to be delivered. The execution continues normally.
User action : None, unless a definite interruption is needed in which case the user might consider typing `<^Y>`...

`W-PAUSE, Error returned by aborted command ignored`
SIC, monitor routine. A `PAUSE` was generated as the result of pressing `<^C>` during a command execution, but the command completed with an error status. The `<^C>` takes precedence over the error to avoid using the error recovery procedure. This message follows a `"I-PAUSE, Generated by pressing <^C>"` message.
User action : As for any pause.

`I-PAUSE, Generated by pressing <^C>`
SIC, monitor routine. A `PAUSE` was generated as the result of pressing `<^C>` during a command execution. The previous command completed normally.
User action : Type any command you want. The interrupted execution level will be restarted by command `CONTINUE`.

`F-PAUSE, Level depth too large`
SIC, monitor routine. An error occured, but no `PAUSE` could be delivered because the input level is already too deep. The program aborts execution with a symbolic stack dump. This error can (in principle) only occur if you are using a set of nested macros as error recovery procedure, and with an invalid command in one of the macros...
User action : Correct error and restart the program. Avoid using such complex error recovery systems.

`F-PAUSE, Session is not interactive`
SIC, monitor routine. An error occured, but no `PAUSE` could be delivered because the session is a batch mode. The program aborts execution with a symbolic stack dump.
User action : correct the invalid command which caused the error and resubmit the job.

`E-RECALL, Command line not found`
SIC, `RECALL` command. No command line in the current stack buffer matches the abbreviation given.
User action : Use the `TYPE` command to see if the line you need does exist. This error may be due to an incorrectly specified language field. See `HELP RECALL`.

`E-RECALL, Non existent line in buffer`
SIC, `RECALL` command. The requested line does not exist in the current stack buffer.
User action : Specify a valid command number.

10.11 S

`E-SEXA, Invalid minute field`

SIC, Sexagesimal decoding routine. The minute field is negative, or greater than 60.
User action : Correct typing errors.

E-SEXA, Invalid second field
SIC, Sexagesimal decoding routine. The minute field is negative, or greater than 60.
User action : Correct typing errors.

E-SEXA, Syntax error
SIC, Sexagesimal decoding routine. Valid syntaxes for sexagesimal arguments are +DD.DDD, +DD:MM.MMM, and +DD:MM:SS.SSS.
User action : Correct typing errors.

W-SIC, Ambiguous keyword, choices are :
SIC, SIC command. The first argument is ambiguous.
User action : specify more characters.

W-SIC, Cannot change SIC\ language status
SIC, SIC command. Information message : language SIC\ cannot be removed from the active scope.

W-SIC, Cannot set HELP mode <String>
SIC, SIC command. The user requested an invalid mode for HELP.
User action : specify a valid mode. Valid modes are PAGE and SCROLL.

W-SIC, Cannot set <String> language <String2>
SIC, SIC command. Information message : languages can only be ON or OFF. Library only languages cannot be brought into the active scope.

E-SIC, Cannot set <String> switch <String2>
SIC, SIC command. The status <String2> does not exist for the switch <String>
User action : check for typing errors.

E-SIC, Command invalid in this context <string>
SIC, monitor routine. Command IF, ELSE and ENDIF can only be used in procedures.
User action : use a procedure.

F-SIC, Commands must be character*12
SIC, initialization routine. The command names are too long or too short. Execution aborts.
User action : notify the programmer.

F-SIC, Demonstration period exhausted, Call your system manager

User action : ask your system manager to buy an authorized copy.

F-SIC, Duplicate language name
SIC, initialization routine. The language name is already used. Execution aborts.
User action : notify the programmer.

W-SIC, Edit mode requires an ANSII terminal
SIC, SIC command. Information message : the user requested the EDIT mode, but is not logged on a ANSII (or compatible) video terminal. EDIT mode is left OFF.

`I-SIC, Editor is <String>`
: `SIC, SIC` command. Information message specifying which text editor is used by command `EDIT`.

`I-SIC, HELP mode is <String>`
: `SIC, SIC` command. Information message specifying the `HELP` mode.

`W-SIC, Incorrect nesting of IF blocks`
: `SIC`, monitor routine. Some `IF` blocks were not properly nested and are still opened when a macro or loop terminates. The opened blocks are closed by SIC.
User action : although this is only a warning, it may be wise to check the current macro for possible error(s).

`F-SIC, Initialization error number <Integer>`
: `SIC`, initialization routine. An undocumented initialization error occurred. Execution aborts.
User action : notify the programmer, who should submit an SPR.

`F-SIC, Internal logic error LIRE = -1`
: `SIC`, command reading routine. This is a fatal bug check.
User action : Submit an SPR, with as much information as you can (log file, program listing, etc...).

`E-SIC, Invalid nesting of loops and IF blocks`
: `SIC`, monitor routine. Some `FOR` loops were not properly nested and are still opened when an `IF` block terminates.
User action : Check the current macro for error(s).

`W-SIC, Invalid precision <string>`
: `SIC, SIC` command. The only supported precisions are `SINGLE` (or `REAL`) and `DOUBLE`. Previous precision is kept.
User action : specify a valid precision.

`F-SIC, Language initialization failure`
: `SIC`, initialization routine. This message is preceded by a more detailed account of the problems. This is a fatal error, and the program execution aborts with a symbolic stack dump.
User action : notify the programmer.

`F-SIC, Programming error:  recursive call to SIC`
: `SIC`, monitor routine. The programmer made a recursive call to SIC. This is a fatal error, and the program execution aborts with a symbolic stack dump.
User action : notify the programmer.

`W-SIC, Session is not interactive, EDIT and MEMORY Off`
: `SIC`, monitor routine. Information message. This message appears at the beginning of the program (usually when nobody is available to read it...).

`F-SIC, SIC is not loaded`
: `SIC`, monitor routine. The program attempted to use SIC before initializing the interpreter. This is a programming error.
User action : notify the programmer.

`I-SIC, <String> switch is <String2>`
`SIC, SIC` command. Information message.

`I-SIC, <String> language is <String>`
`SIC, SIC` command. Information message.

`W-SIC, Sub-process <String> is still active`
`SIC`, exit routine. A sub-process has been created earlier by the `SYSTEM` command. The sub-process is not deleted.
User action : You can attach to this sub-process at any time by the VMS command `ATTACH`, or delete it by the VMS command `STOP`.

`E-SIC, Too many IF blocks`
`SIC`, monitor routine. You are attempting to nest `IF` blocks too deeply.
User action : Do not. Find another way to solve your problem.

`F-SIC, Too many commands and options. This program is only dimensioned for <Integer> user-defined commands.`
`SIC`, initialization routine. The program has too many commands and options. Execution aborts.
User action : notify the programmer, who may submit an SPR (even though this is no an error in SIC just a limitation).

`F-SIC, Too many languages`
`SIC`, initialization routine. The program has too many languages. Execution aborts.
User action : notify the programmer.

`W-SIC, Undefined character expression <string>`
Formatting routine. The specified string is not a valid character string. This message is usually preceded by other ones that give additional information.
User action : Correct typing mistake(s)

`W-SIC, You are using a demonstration version`
`SIC`. The version of SIC you are using is a demonstration version with a limited validity period (usually 3 to 6 months). Contact the authors about a permanent licence (available at no cost for academic institutions).
User action : Beware that the validity period will expire...

`E-SYMBOL, Invalid symbol name <String>`
`SIC, SYMBOL` routine. A symbol name must begin with an alphabetic uppercase character.
User action : use a valid name.

`F-SYMBOL, SIC is not loaded`
`SIC, DEFINE_SYMBOL` routine. The program attempts to define a symbol before SIC has been initialized. This is a programming error.
User action : notify the programmer.

`E-SYMBOL, String too long, translation failed`
`SIC`, monitor routine. The line buffer is too short to accomodate the symbol translation. The command is not executed, and an error occurs.
User action : : If possible shorten your command or symbol translation. Eventually contact the programmer.

`E-SYMBOL, Symbol definition too long`
`SIC, SYMBOL` command. The equivalence name is too long. The symbol is undefined.
User action : cut the definition in two symbols and use concatenation when translation is required.

`W-SYMBOL, Symbol name too long <String>`
`SIC, SYMBOL` command. Symbol names must be shorter than 12 characters.
User action : use shorter symbol names.

`W-SYMBOL, Symbol truncated to <String>`
`SIC, SYMBOL` command. A symbol name contained more than 12 characters and has been truncated.

`E-SYMBOL, Too many symbols`
`SIC, SYMBOL` routine. There are too many symbols, the new definition has not been added.
User action : delete unwanted symbols before adding a new one.

`I-SYMBOL, Table is empty`
`SIC, SYMBOL` command. Information message, there are no symbols defined.

`I-SYMBOL, Table contains :`
`SIC, SYMBOL` command. Information message, followed by the list of symbols and equivalence strings.

`W-SYMBOL, Undefined symbol <String>`
`SIC, SYMBOL` command. The specified symbol is undefined.

`E-SYSTEM, Sub-process cannot be activated`
`SIC, SYSTEM` command with or without arguments. The sub-process could not be created because of lack of system resources. The VMS error message follows this error.
User action : If the command had an argument, retry it without as you may connect to an existing subprocess. If this fails, it is generally due to an exceeded quota of subprocesses. If you have other subprocesses running, the `SYSTEM` command lists the current subprocesses and prompts you to which one you want to attach. If none is available, the `SYSTEM` command will return you an error.

`E-SYSTEM, Sub-process <String> could not be attached`
`SIC, SYSTEM` command without argument, or with option `/PROCESS.` The programs failed to attach to an existing sub-process, previously created by the `SIC\SYSTEM` command, or by another program. If the option `/PROCESS` was not present, the command will try to create a new one.
User action : Check process name in case you used the `/PROCESS` option.

`W-SYSTEM, More than <number> sub-processes active`
`SIC, SYSTEM` command with or without arguments. You have reached the maximum number of sub-processes allowed within SIC˙
User action : If the command had an argument, retry it without as you may connect to an existing subprocess. If this fails, the `SYSTEM` command lists the current subprocesses and prompts you to which one you want to attach. If none is available, the `SYSTEM` command will return you an error.

`E-SYSTEM, Sub-process cannot be created`
`SIC, SYSTEM` command with or without arguments. The sub-process cannot be created, usually because of lack of system resources. The VMS error message follows this error.
User action : If the command had an argument, retry it without as you may connect to an existing subprocess. If this fails, it is generally due to an exceeded quota of subprocesses. If you have other subprocesses running, the `SYSTEM` command lists the current subprocesses and prompts you to which one you want to attach. If none is available, the `SYSTEM` command will return you an error.

10.12 T

`E-TYPE, Cannot open <String>`
`SIC, TYPE` command. The specified file or macro could not be opened. This message is followed by a more precise reason.
User action : check for typing errors.

`E-TYPE, Error reading <String>`
`SIC, TYPE` command. A read error occured during the typing of a file or macro. The `TYPE` command aborts.
User action : most likely you are trying to type a binary file or something like this... Otherwise, it is a hardware problem. Notify your system manager.

10.13 U through Z

`E-VARIABLE, Internal error, no back pointer`
`SIC, DELETE /VARIABLE` command. This is an internal logic error.
User action : Submit an SPR.

`E-VARIABLE, Invalid variable name <String>`
`SIC, DEFINE LET /NEW or FOR` commands. Variable names must be at most 15 characters and begin with a letter.
User action : use a valid name.

`E-VARIABLE, Program defined variables are protected`
`SIC, DELETE /VARIABLE` command. You attempt to delete a variable that has been created by program. This is not allowed.

`F-VARIABLE, SIC is not loaded`
Calling program. The program attempts to define variables before the interpreter has been initialized. This is a programming error.
User action : notify the programmer.

`E-VARIABLE, Too many variables`
`SIC, SIC, DEFINE LET /NEW or FOR` commands. You attempted to define too many variables.
User action : Delete unused variables and retry. If this is not sufficient, submit an SPR, and we will increase the buffer size.

`E-VARIABLE, Variable <String> already exists`
`SIC, DEFINE or LET /NEW`. The specified name is already a known variable.
User action : use a different name, or delete the variable before.

`E-VARIABLE, Variable name too long`
: `SIC, DEFINE LET /NEW or FOR` commands. Variable names must be at most 15 characters.
User action : use a shorter name.

`E-ZCRONGNEUGNEU, j'y arrive pas`
: Congratulations, **you got a free bottle of champagne** if...
you can reproduce the error.
User action : Contact the authors.

11 Task demonstration

11.1 demo

```
EXAMPLE          Demonstration program, mostly used for GILDAS tests
PRIMES           Compute primes numbers (used for GILDAS tests)
```

11.2 EXAMPLE

```
EXAMPLE
```

This is a sample program doing nothing, but used to test GILDAS...

11.3 PRIMES

```
PRIMES
```

This is a sample program computing primes numbers up to an internal limit, used to test GILDAS.

Index