SIC

a GILDAS working group software

February 7, 2002

SIC

Sympathetic Interpreter of Commands
Sympathique Interpréteur de Commandes
Version 7.0

The GILDAS working group is a collaborative project of the Observatoire de Grenoble (1,3) and IRAM (2), and comprises: F. Badia\textsuperscript{2}, D. Broguière\textsuperscript{2}, G. Buisson\textsuperscript{1}, L. Desbats\textsuperscript{1}, G. Duvert\textsuperscript{1}, T. Forveille\textsuperscript{1}, R. Gras\textsuperscript{3}, S. Guilloteau\textsuperscript{1,2}, R. Lucas\textsuperscript{1,2}, R. Ner\textsuperscript{i} and P. Valiron\textsuperscript{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. Begon\textsuperscript{3}, S. Delahaye\textsuperscript{3}, A. Dutrey\textsuperscript{1,2}, C. Kahane\textsuperscript{1}, P. Monger, J.L. Monin\textsuperscript{1} and all GILDAS users are gratefully acknowledged.
## Contents

1 Introduction ................................................. 4

2 The SIC Monitor ........................................... 5
   2.1 Basic Features ......................................... 5
   2.1.1 Syntax .................................................. 5
   2.1.2 The Prompt .............................................. 6
   2.1.3 The on-line HELP ......................................... 6
   2.1.4 The Stack ................................................ 6
   2.1.5 Line Editing Facility ..................................... 7
   2.1.6 The Log File ............................................ 7
   2.1.7 Symbols .................................................. 8
   2.2 Variables and Expressions ................................. 8
       2.2.1 Definitions and Assignments .......................... 8
       2.2.2 Functions and Operators .............................. 9
       2.2.3 Vector Operations ...................................... 10
       2.2.4 Implicit Loops ......................................... 10
       2.2.5 Conditional Assignment ................................ 11
       2.2.6 Size casting ........................................... 11
       2.2.7 GILDAS Images ......................................... 11
       2.2.8 GILDAS Headers ......................................... 12
       2.2.9 Character Variables and Implicit Formatting .......... 13
       2.2.10 Initializing variables from external files .......... 14
   2.3 SIC as a programming language ............................ 14
       2.3.1 Procedures (or Command Files) ....................... 14
       2.3.2 Loops ................................................... 15
       2.3.3 Structured Programming and Logical Expressions .... 15
       2.3.4 Execution Level ....................................... 16
       2.3.5 Error Recovery ........................................ 17
   2.4 The GUI ("Graphics-User-Interface") Mode ................ 18
       2.4.1 Detached menus ....................................... 18
       2.4.2 Assigning variables in "Window" mode ............... 19
       2.4.3 Actions and Buttons in "Window" mode ............... 19
       2.4.4 Help file structure .................................. 20
   2.5 Interacting with the Operating System .................... 20
       2.5.1 File Operations ....................................... 21
       2.5.2 SYSTEM command: VMS Version ....................... 21
       2.5.3 SYSTEM Command: Unix operating system .......... 22
   2.6 Customizing ............................................... 22
       2.6.1 Logical Names ........................................ 22
       2.6.2 User Defined Commands ............................... 22
       2.6.3 Initialization File ................................... 22
       2.6.4 The SIC Command .................................... 23

3 SIC Language Internal Help .................................. 24
   3.1 ACCEPT .................................................. 24
       3.1.1 ACCEPT /ARRAY ..................................... 24
       3.1.2 ACCEPT /BINARY ..................................... 24
       3.1.3 ACCEPT /COLUMN ..................................... 24
       3.1.4 ACCEPT /FORMAT ..................................... 24
       3.1.5 ACCEPT /LINE ....................................... 25
   3.2 @ ....................................................... 25
   3.3 BEGIN ................................................... 25


<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4 BREAK ................................................................. 25</td>
</tr>
<tr>
<td>3.5 COMPUTE ............................................................. 26</td>
</tr>
<tr>
<td>3.6 CONTINUE ........................................................... 27</td>
</tr>
<tr>
<td>3.7 SYSTEM ............................................................. 27</td>
</tr>
<tr>
<td>3.7.1 SYSTEM VMS ...................................................... 27</td>
</tr>
<tr>
<td>3.7.2 SYSTEM UNIX .................................................. 28</td>
</tr>
<tr>
<td>3.8 DEFINE .............................................................. 28</td>
</tr>
<tr>
<td>3.8.1 DEFINE CHARACTER ........................................... 29</td>
</tr>
<tr>
<td>3.8.2 DEFINE COMMAND .............................................. 29</td>
</tr>
<tr>
<td>3.8.3 DEFINE DOUBLE ............................................... 29</td>
</tr>
<tr>
<td>3.8.4 DEFINE FITS ................................................... 29</td>
</tr>
<tr>
<td>3.8.5 DEFINE FUNCTION ............................................. 30</td>
</tr>
<tr>
<td>3.8.6 DEFINE HEADER ............................................... 30</td>
</tr>
<tr>
<td>3.8.7 DEFINE IMAGE ............................................... 31</td>
</tr>
<tr>
<td>3.8.8 DEFINE INTEGER ............................................ 32</td>
</tr>
<tr>
<td>3.8.9 DEFINE LOGICAL ............................................. 32</td>
</tr>
<tr>
<td>3.8.10 DEFINE REAL .............................................. 32</td>
</tr>
<tr>
<td>3.8.11 DEFINE STRUCTURE .................................... 32</td>
</tr>
<tr>
<td>3.8.12 DEFINE TABLE ............................................ 32</td>
</tr>
<tr>
<td>3.8.13 DEFINE /LIKE ............................................. 33</td>
</tr>
<tr>
<td>3.9 DELETE ............................................................. 33</td>
</tr>
<tr>
<td>3.10 EDIT ............................................................... 33</td>
</tr>
<tr>
<td>3.11 ELSE ............................................................... 34</td>
</tr>
<tr>
<td>4 SIC Programming Manual ........................................ 35</td>
</tr>
<tr>
<td>5 SIC Error Messages and Recovery Procedures .................. 36</td>
</tr>
<tr>
<td>5.1 A through C ...................................................... 37</td>
</tr>
<tr>
<td>5.2 D ................................................................. 37</td>
</tr>
<tr>
<td>5.3 E ................................................................. 39</td>
</tr>
<tr>
<td>5.4 F ................................................................. 40</td>
</tr>
<tr>
<td>5.5 G ................................................................. 42</td>
</tr>
<tr>
<td>5.6 H ................................................................. 42</td>
</tr>
<tr>
<td>5.7 I ................................................................. 42</td>
</tr>
<tr>
<td>5.8 J through L ..................................................... 44</td>
</tr>
<tr>
<td>5.9 M ................................................................. 45</td>
</tr>
<tr>
<td>5.10 O through R .................................................... 47</td>
</tr>
<tr>
<td>5.11 S ................................................................. 49</td>
</tr>
<tr>
<td>5.12 T ................................................................. 52</td>
</tr>
<tr>
<td>5.13 U through Z ..................................................... 52</td>
</tr>
</tbody>
</table>
1 Introduction

SIC (*) is a command line interpreter, written in FORTRAN and callable as a subroutine by any program. It provides a command language rather similar to the VAX-VMS DCL language, with the following major features:

- resolution of command abbreviations
- definition of symbols
- macro capabilities 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 on VT200 compatible devices

NEW!: GUI interface on Motif systems

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 programming manual for the user who would like to use SIC to create its own application. Chapter 5 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:

\[ [\text{LANG}] \text{COMM} [\text{ARG1} [\text{ARG2} [...]]] [\text{/OPT1} [\text{ARG11} [...]]] [\text{/OPT2} [...]] \]

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 (Blanks 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 2.5), 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 2.13).

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

\begin{verbatim}
SIC> HELP -
SIC> EDIT
\end{verbatim}

is interpreted as

\begin{verbatim}
SIC> HELP EDIT
\end{verbatim}

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

\begin{verbatim}
SIC> LABEL "A very long-
SIC> string must be cut"
\end{verbatim}

which is interpreted as

\begin{verbatim}
SIC> LABEL "A very long string must be cut"
\end{verbatim}

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:

\begin{verbatim}
SIC> DRAW RELOCATE 13- !This is an example of comments
SIC> .45 15.00 /USER ! and continuation
\end{verbatim}

is interpreted as

\begin{verbatim}
SIC> DRAW RELOCATE 13.45 15.00 /USER
\end{verbatim}

BUT do not try this one

\begin{verbatim}
SIC> LABEL "A lon-
SIC> g caption" ! A long string
SIC> g caption" ! with continuation mark
\end{verbatim}
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 may contain up to 300 commands,
less if the commands are very long. It 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. However to bypass VMS line editing, you must strike GOLD (¬PF1>key
first.

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.

If you are using an ANSI 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
- `<&>` Set advance mode (for WORD moves)
- `<$>` 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

These commands are the same as in the EDT Digital text editor, except for Backspace (§).

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.

Note that on VMS operating systems, this editing mode somehow conflicts with the basic DEC line editing facility, in the sense that the DEC editing facility has priority over it. To access to the keypad editing mode you should press the GOLD (`<PF1>`) key. Otherwise, the Up and Down arrows will only allow you to retrieve the last typed line (default DEC line editing).

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 GACLOG 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 interpreter 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 15 characters long, upper case only, 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
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 known operators are:

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

and 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 4 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,

```
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:

```
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 preceding example, C[,1,1] could have replaced C[,10,1]. The indices can be previously defined scalar variables. Leading commas may be omitted.

```
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.

```
LET A[;3] = B[;8]
```

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]
FUR J 1 TO 10
FUR 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

\[
\text{DEFINE REAL } A[10,10] \\
\text{LET } A[I,J] = \exp(-(I-5)^2-(J-6)^2) /\text{WHERE } (I+J).LT.10
\]

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

\[
\text{DEFINE REAL } A[10,10] \\
\text{FOR J 1 TO 10} \\
\text{FOR I 1 TO 10} \\
\text{IF (I+J).LT.10 THEN} \\
\quad \text{LET } A[I,J] = \exp(-(I-5)^2-(J-6)^2) \\
\text{ENDIF} \\
\text{NEXT} \\
\text{NEXT}
\]

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

\[
\text{DEFINE REAL } B \ C \ D /\text{LIKE } A
\]

is equivalent to

\[
\text{DEFINE REAL } B[10,5,30] \ C[10,5,30] \ D[10,5,30]
\]

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

\[
\text{DEFINE IMAGE Variable File Keyword}
\]

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

\[
\text{READ use an existing image as Read-Only variable} \\
\text{WRITE use an existing image as Read-Write variable. The user must have Write access to the file.}
\]
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. Table 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 TABLE 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARGENE</td>
<td>Integer</td>
<td>Length of general section</td>
</tr>
<tr>
<td>VARNDIM</td>
<td>Integer</td>
<td>Number of dimensions (ReadOnly)</td>
</tr>
<tr>
<td>VARDIM</td>
<td>Integer[4]</td>
<td>Dimensions (ReadOnly)</td>
</tr>
<tr>
<td>VARCHNCON</td>
<td>Double[3,4]</td>
<td>Conversion formulae for the 4 axes: Value at reference pixel, Increment</td>
</tr>
<tr>
<td>VARBLANK</td>
<td>Integer</td>
<td>Length of blanking section</td>
</tr>
<tr>
<td>VARBLANK</td>
<td>Real[2]</td>
<td>Blanking and tolerance</td>
</tr>
<tr>
<td>VAREXTRA</td>
<td>Integer</td>
<td>Length of extrema section</td>
</tr>
<tr>
<td>VARMAX</td>
<td>Real</td>
<td>Maximum</td>
</tr>
<tr>
<td>VARMIN</td>
<td>Real</td>
<td>Minimum</td>
</tr>
<tr>
<td>VARMAX</td>
<td>Integer[4,2]</td>
<td>Position of max and min</td>
</tr>
<tr>
<td>VARDESC</td>
<td>Integer</td>
<td>Length of units and system section</td>
</tr>
<tr>
<td>VARIUNIT</td>
<td>Char*12</td>
<td>Image unit</td>
</tr>
<tr>
<td>VARIUNIT1</td>
<td>Char*12</td>
<td>First axis type</td>
</tr>
<tr>
<td>VARIUNIT2</td>
<td>Char*12</td>
<td>Second axis type</td>
</tr>
<tr>
<td>VARIUNIT3</td>
<td>Char*12</td>
<td>Third axis type</td>
</tr>
<tr>
<td>VARIUNIT4</td>
<td>Char*12</td>
<td>Fourth axis type</td>
</tr>
<tr>
<td>VARSYS</td>
<td>Char*12</td>
<td>Coordinate system</td>
</tr>
<tr>
<td>VARPX</td>
<td>Integer</td>
<td>Length of position section</td>
</tr>
<tr>
<td>VARSRC</td>
<td>Char*12</td>
<td>Source name</td>
</tr>
<tr>
<td>VARRA</td>
<td>Double</td>
<td>Right Ascension</td>
</tr>
<tr>
<td>VARDEC</td>
<td>Double</td>
<td>Declination</td>
</tr>
<tr>
<td>VARL1</td>
<td>Double</td>
<td>Galactic longitude</td>
</tr>
<tr>
<td>VARL2</td>
<td>Double</td>
<td>Galactic latitude</td>
</tr>
<tr>
<td>VAREPOCH</td>
<td>Real</td>
<td>Epoch of coordinates</td>
</tr>
<tr>
<td>VARP1</td>
<td>Integer</td>
<td>Length of projection section</td>
</tr>
</tbody>
</table>
VAR\$TYPE  Integer  Projection type (code)
VAR\$AO    Double  first coordinate of projection center
VAR\$DO    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\$FREQOFF Double  Frequency offset
VAR\$RESTFRE Double  Rest Frequency
VAR\$VELRES Real    Velocity resolution
VAR\$VLOFF  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 becomes a dummy variable of type header, which can only be referenced in a further DELETE /VARIABLE command. VAR can have at most 6 characters. 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.

 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 GILDAS data file can be obtained within SIC as follows:

 DEFINE IMAGE A Oldfile.gdf READ
 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 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 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 mathematical 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 implicitly 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".

2.2.10 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 tokens &1, &, 2, ..., &8 can be used in the body of the procedure, and when a procedure is invoked its arguments will be substituted for these tokens. Substitution occurs also within the character strings. A procedure is executed by
SIC> @ Procedure_Name [P1,...]

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 substituted to the token &1 during execution. Up to nine 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 9 loop 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. "LogicalExpression" 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 Logical_Expression [THEN]
...
ELSE IF Logical_Expression [THEN]
...
The syntax is similar to FORTRAN with two major differences. First, the logical expression of an IF command cannot be followed by another command to be executed, and the THEN keyword is optional. Second, ELSE and IF must be separated by at least one space or tab in the ELSE IF command. Nesting of IF blocks up to 20 levels is allowed. In addition, provided the restriction on the 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:

\[
L = ("I am happy".EQ.GOOD) (.FALSE.)
\]

\[
L = ("Let it be".EQ.GOOD) (.TRUE.)
\]

\[
L = ("I am happy".EQ."GOOD") (.TRUE.)
\]

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 (0, FOR), suspend (PAUSE), resume (CONTINUE) or abort its execution (NEXT, QUIT, BREAK, RETURN). Errors occurring 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 1 changes to 'SIC:>' 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.  It is always a non ambiguous abbreviation of CONTINUE, unless you redefine it as a Symbol.

EXIT  
extit 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 occurred.

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 occurred 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 occurred. 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 keypad edition is possible, the command which caused the error is automatically displayed for correction by the user. The prompt is put in inverse video to indicate that edition is going on. 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 Ø, CONTINUE, EXIT, NEXT. They will behave exactly as 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 transmit 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

When a windowing system is available, SIC provides facilities (called GUI mode) to create documented input windows by which the user can modify variables and execute pre-defined commands. Currently, such facilities are available for Motif-based systems (soon for MAC-OS, and later for Windows-95).

SIC can create 3 types of windows:

- 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 input 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.

- Optional windows
  which are created by the GUI\ BUTTON command. Such windows are a 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 pull-down 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 REFRESH" REFRESH
GUI\BUTTON "ZOOM" ZOOM
GUI\BUTTON "HARDCOPY /PLOT" HARDCOPY
GUI\MENU /CLOSE

GUI\GO

Several detached menus can be activated at once.

2.4.2 Assigning variables in “Window” mode
When the “Main input 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 input window” (or “Optional window”). 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 input 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 input window” is created when command GUI\GO is typed. Four buttons are defined by default:

- the OK button,
  which sets all variables defined in the main and optional 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 optional 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 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 widget in a “optional” window. This window is by default hidden, but can be 
unveiled by the user.

The “optional” window 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 END
```

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 (currently VMS, Unix, Mac-OS, 
MS-Windows), 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.

SIC> SIC DIRECTORY [NewDir]
   controls the working directory. Without argument, the current working directory is listed. With one
   argument, the working directory is changed. Standard Unix syntax applies to change the working
   directory.

SIC> SIC\SIC COPY FileIn 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. Standard SIC file naming convention applies. Both files should
   reside on the same disk. To move files across different disks, use SIC COPY and SIC DELETE.

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: VMS Version

SYSTEM ["Command"]
   SYSTEM /PROCESS Process_Name
   The recommended use of the SYSTEM command is without any argument. In this case, SIC either creates
   a sub-process or attaches the terminal to a sub-process previously created. The created sub-process has
   originally the prompt $$. To return to the SIC monitor, type QUIT. Next SYSTEM commands will attach
   to the same sub-process which is not deleted when the program using SIC stops. It is recommended to
   use this form, rather than the command with argument, because it avoids the overhead of creating one
   sub-process per command. The sub-process is just created once.

   Before attempting a new process creation, SIC identifies the existing subprocesses, and prompts you
   to which one you want to attach. Press the RETURN key if you want to create a new one.

   In addition, you can attach to other processes created by other programs or commands using the
   SYSTEM /PROCESS command. In this case, to return to SIC you have to issue a VMS command ATTACH
   Prcnam where Prcnam is the name of the process which is running SIC (Use the VMS command SHOW
   PROCESS /SUBPROCESS to determine the process name Prcnam). QUIT will usually not work (although it
   may).

   With arguments, the SYSTEM command effectively activates any valid VAX-VMS command by creating
   a temporary sub-process executing the requested command. Control is returned to SIC after completion
   of the command. This mode is not recommended because of the overhead required to create the process.
   For example you may issue the command

   SIC> SYSTEM DIR
   to list your current directory. Note that you cannot modify a local assignment in this way because of
   the specificities of the sub-process. Quotes are usually not necessary to delimitate the command passed
to VMS; the only exception concerns the options, for which you must avoid confusion between SIC and
SYSTEM.

   SIC> SYSTEM "DIR /FULL *.PRO" is perfectly valid
   SIC> SYSTEM DIR /FULL *.PRO will produce the message

E-INTER, Unknown option /FULL for command SYSTEM
2.5.3 SYSTEM Command: Unix 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 SIC must 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 their 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).

On VMS systems, standard VMS logical names are used.

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 "@ PR:P_INPUT.GRAPHIC" PR:INPUT_GRAPHIC.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.
2.6.4 The SIC Command

The SIC command controls several internal parameters of the SIC monitor.

SIC> SIC EDIT [ON] [OFF] [Editor_Name]
Controls whether line edition is possible or not, or select the default text editor. On VMS, use EDT or TPU (Vax Text Processing Utility). On Unix, use "vi" or "emacs" or any other you wish (note the distinction between lowercase and uppercase). On Unix with the X-Window system, specifying "emacs &" will allow to launch the editor in a separate window.

SIC> SIC VERIFY [ON] [OFF]
controls the listing of the command flow. Commands executed in procedure mode (macro or loop) will be echoed to the terminal if VERIFY is ON, and will not be echoed if is OFF.

SIC> SIC MEMORY [ON] [OFF]
allows to activate or deactivate automatic insertion of commands executed in interactive mode.

SIC> SIC Language[] [ON] [OFF] allows to activate or deactivate one of the current languages of the program. Commands of inactive languages are not scanned by the SIC monitor. Language SIC[] itself cannot be deactivated of course.

SIC> SIC HELP [PAGE|SCROLL|CONTENT|INDEX]
controls the HELP mode.
- SCROLL scroll simple help
- PAGE type simple help, with a prompt to the user for continuation after about 20 lines have been typed.
- CONTENT Display PostScript documentation about the command on X-Window systems
- INDEX Display first reference to the command in the PostScript manual on X-Window systems.

SIC> SIC PRECISION [SINGLE] [DOUBLE] [AUTO]
controls the precision in which mathematic formulae are evaluated. Automatic precision uses the precision of the result.

SIC> SIC EXTENSION [.Defext]
controls the default macro extension.

SIC> SIC OUTPUT [FileName] can be used to redirect the output of command SAY to the specified FileName. Without argument, the current SAY output file is closed.

SIC> SIC LOGICAL LogName [Translation] can be used to list, define or redefine logical names. Without argument, all logical names are displayed.

The SIC command without arguments produces a summary of the internal SIC status, and with a single argument it shows the status of that argument.
3 SIC Language Internal Help

3.1 ACCEPT

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

3.1.1 ACCEPT /ARRAY

[SIC]\ acceptance 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.

3.1.2 ACCEPT /BINARY

[SIC]\ acceptance 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.

3.1.3 ACCEPT /COLUMN

[SIC]\ acceptance Var_1 [Var_2 [...] /COLUMN File_Name [/FORMAT String]
[/LINE Begin End]

Read ONE or SEVERAL 1-Dimensional variables in list-directed (free)
format from a formatted file File_Name. Mixing CHARACTER and NUMERIC
(REAL, etc...) arrays is not available yet, and only one character array
can be read at a time. The user can specify a format using the /FORMAT
option, and select a line range using the /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.

3.1.4 ACCEPT /FORMAT

Specifies a fortran format to read the input file for command ACCEPT
ARRAY or ACCEPT /COLUMN. This option is invalid with /BINARY. example:
DEFINE CHARACTER CITY*16[44]
DEFINE REAL X[44] Y[44]
ACCEPT X Y /COLUMN "cities.dat" /LINE 4 / FORMAT "20x,F8.3,ix,F8.3"
ACCEPT CITY /COLUMN "cities.dat" /LINE 4 / FORMAT "2x,A16"

3.1.5 ACCEPT /LINE

Indicates the line range to be read in the input file for command
ACCEPT/ARRAY or ACCEPT/COLUMN. This option is invalid with /BINARY

3.2 @

[SIC]\@ Macro_Name [Par1 [Par2 [...]]]

Reads commands from macro Macro_Name and executes them. Up to 8
parameters can be given. These parameters will be substituted to the
tokens 1, 2, ..., 8 found in the body of the macro, even within
character strings (e.g. Par1 for token 1). Default file extension is
program dependent (usually the program name, or .pro), and can be
changed using command SIC EXTENSION. 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 (EDIT, TPU, EDU or EVE on VMS, Emacs or Vi on Unix, NotePad or
WordPad on MS-Windows), by typing command EDIT with the macro file name
as argument (see EDIT).

3.3 BEGIN

[SIC]\BEGIN Procedure|Help|Data FileName

Begins 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 GAGPROC:.

3.4 BREAK

[SIC]\BREAK

Terminates 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+I.EQ.0
    BREAK
END IF
NEXT

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

3.5 COMPUTE

[SIC]COMPUTE OutVar OPERATION InVar [Parameters]

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

The following operations are available on REAL and DOUBLEPRECISION arrays, regardless of their nature:
- MAX  MIN  MEAN  RMS  SUM  PROD
  for which InVar is an array of rank 1 to 4 (a 1-D to 4-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 supported.
- HISTOGRAM
  Example: "COMPUTE outvar HISTOGRAM invar [hmin] [hmax] [bval]
  [eval]"
  which puts 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 Parameters can be absent. One can use '*' to omit any of them.

The number of bins is dictated by the first dimension of the array min 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]).

The following transformations are available:
- FFT+  (Direct Fast Fourier Transform)
- FFT-  (Inverse Fast Fourier Transform)
  which accept REAL or COMPLEX parameters. 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.
- FOURT+  (Direct Fast Fourier Transform)
- FOURT-  (Inverse Fast Fourier Transform)
  which operates on input and output on COMPLEX Arrays of dimension [2,NX, NY]
- COMPLEX
Populates the Real part of outvar (complex variable) with invar (real).
- 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.
- CMPMUL as in COMPUTE varout CMPMUL varin1 varin2:
  varout = PRODUCT of Varin1 and Varin2. All complex arrays.

Miscellaneous Operations:
- DATE as in COMPUTE creation_date DATE filename
  Where 'filename' is a file name and returns the last modification
date in the integer variable 'creation_date'. Used in procedures to
test if file has been changed, etc...
- GAG_DATE as in COMPUTE idate GAG_DATE "15-DEC-2035"
stores the date "15-DEC-2035" as a modified Julian date in the
integer variable 'idate'. This modified julian date has little
meaning outside the rather sparse community of souls gathered around
the CLASS program, however...

3.6 CONTINUE
[SIC\]CONTINUE or C

Resumes 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.

3.7 SYSTEM
[SIC\]SYSTEM ["Command"] [/PROCESS Process_Name]

See HELP SYSTEM VMS or HELP SYSTEM UNIX for operating-system dependent
help.

3.7.1 SYSTEM VMS
[SIC\]SYSTEM ["Command"] [/PROCESS Process_Name]
VMS Help version

Execute the VAX-VMS command specified by Command, or attach or spawns a
sub-process.

If specified with an argument, the SYSTEM command tries to execute the
requested VMS command by creating a temporary sub-process executing the
requested command. Control is returned to SIC after completion of the
command and the sub-process deleted. This mode is NOT RECOMMENDED,
because it is time consuming.
If no command is specified, the SYSTEM command spawns a sub-process or attaches the terminal to an existing one. The sub-process prompts for SYSTEM commands at the user's terminal, with a $\$\$ prompt to distinguish it from the parent process. Type QUIT to return to SIC. The first time the SYSTEM command is used, the SIC monitor looks for all existing sub-processes and asks the user the name of the sub-process to attach. Answer by specifying the complete name, or press RETURN to create a new sub-process. If no sub-process is available, the SYSTEM command creates automatically a new one. If a sub-process creation fails, the SYSTEM command identifies your current sub-processes, and prompts you to which one you want to attach. Following executions of the SYSTEM command simply attach to the same sub-process.

Any sub-process created with the SYSTEM command is not deleted when you exit from the program running SIC, so that you may connect to it by VMS command ATTACH later (or by other program using SIC with the SYSTEM command of course). To kill the sub-process, type LOGOUT (in the sub-process!) instead of QUIT to return to SIC. You can also stop your unwanted sub-processes (from the parent process) by the VMS command

$ STOP <Sub-process_name>

If the option /PROCESS is specified, the SYSTEM command attaches the terminal to the specified process, which may have been created in any manner. No command is allowed in this case.

3.7.2 SYSTEM UNIX

[SIC\]SYSTEM ["command name"]

Unix version

If no argument is given, start a subshell by running the default user shell (sh, csh or ksh). The subshell can be terminated by typing 'exit' or 'bye' or 'logout', depending on Unix version, and controls return to SIC in this case.

If an argument is given, execute the Unix command in a subshell.

Note that new Unix environment variables cannot be defined in such a way, since it is a subshell. In particular, use command SIC DIRECTORY to change your working directory.

SHORT CUTS:

Unix commands can also be executed directly from the SIC level using the token, e.g.

No case conversion occurs in such a case, and the "unix is directly executed in a subshell.

3.8 DEFINE

[SIC\]DEFINE Type Var1 [Keys...] [Var2 [Keys...] [...] ] [/GLOBAL]
3 SIC LANGUAGE INTERNAL HELP

[/LIKE VarLike]

If Type is FUNCTION, define a user-function. If Type is COMMAND, define a new user command. 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, STRUCTURE or FITS. 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.

3.8.1 DEFINE CHARACTER

[SIC\]DEFINE CHARACTER Var1*N1[DIM1] [...]  

CHARACTER variables can be scalar or multi-dimensional. The length of CHARACTER variable is specified after an * after the name of the variable. The dimension field (same syntax as for other arrays) should follow the length declaration.

3.8.2 DEFINE COMMAND

DEFINE COMMAND NAME "Equivalent command" [Help_File]

defines a new SIC command. The new command will be part of the USER\language, and will be translated into its equivalent command at execution time. Help_file is an optional argument indicating the name of the associated help text. The standard rules for help syntax applies (see SIC documentation for details). User defined commands 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:inputeg.hlp

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

3.8.3 DEFINE DOUBLE

[SIC\]DEFINE DOUBLE Var1[DIM1] [Var2[DIM2] [...]]

DOUBLE variables can be multi-dimensional. Up to 4 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]

where Var is the variable name and n1 to n4 are integer constants or variables.

3.8.4 DEFINE FITS

DEFINE FITS Var File [/GLOBAL]

defines 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.
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)

These variables are defined as part of a structure. The 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 (an obsolete FITS structure) is not implemented.

DEFINE FITS only works to read FITS files, but not to create them.

### 3.8.5 DEFINE FUNCTION

**DEFINE FUNCTION** NAME(X,Y,Z) Expression(X,Y,Z)

This defines an arithmetic user function of several variables. The maximum number of variables is 4. The function definition can reference any of the known mathematical operators and intrinsic or program defined functions, but not previously defined user functions (i.e., user function definition is not recursive).

### 3.8.6 DEFINE HEADER

**DEFINE HEADER** Var1 File1 Key1 [Var2 File2 Key2 [...] ] [/GLOBAL]

This defines variables associated to the HEADER of the GILDAS images located in the specified files. The keyword must be either READ or WRITE. 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$EXTREA**: Integer (Length of extrema section)
- **VAR$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\EPOCH    Real        Epoch of coordinates
VAR\PROJ     Integer     Length of projection section
VAR\TYPE     Integer     Projection type (code)
VAR\X0       Double      first coordinate of projection center
VAR\D0       Double      second coordinate of projection center
VAR\ANGLE    Double      position angle of projection
VAR\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\FREQOFF  Double      Frequency offset
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. the VAR\%item variables are ReadOnly or
ReadWrite according to the keyword Key, except for the dimension
variables, which cannot be modified.

3.8.7 DEFINE IMAGE

DEFINE IMAGE Var1 File1 Key1 [Var2 File2 Key2 [...] ] [/GLOBAL]

defines variables associated to both the content and the header of the
GILDAS images located in the specified files. 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.
3.8.8 DEFINE INTEGER

[SIC\]DEFINE INTEGER Var1[DIM1] [Var2[DIM2] [...]]

INTEGER variables can be multi-dimensional. Up to 4 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]
where Var is the variable name and n1 to n4 are integer constants or variables.

3.8.9 DEFINE LOGICAL

[SIC\]DEFINE LOGICAL Var1[DIM1] Var2[DIM2] [...]?

LOGICAL variables can be multi-dimensional. Up to 4 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]
where Var is the variable name and n1 to n4 are integer constants or variables.

3.8.10 DEFINE REAL

[SIC\]DEFINE REAL Var1[DIM1] [Var2[DIM2] [...]?

REAL variables can be multi-dimensional. Up to 4 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]
where Var is the variable name and n1 to n4 are integer constants or variables.

3.8.11 DEFINE STRUCTURE

DEFINE STRUCTURE Str [/GLOBAL]

defines 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...

An entire structure and all its associated members is deleted by a single command DELETE Str /VARIABLE.

3.8.12 DEFINE TABLE

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

defines variables associated to GILDAS images located in the specified file. 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 image of the specified type.
The dimensions must then be specified in the dimension field of the
variable name, as for a standard variable.
- 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.

3.8.13 DEFINE /LIKE
[SIC]\DEFINE Type Var1 [Var2 [...] /LIKE VarLike

DEFINE IMAGE Var1 File1 Key1 [...] /LIKE VarLike

The /LIKE option allows definition of Type REAL, LOGICAL, DOUBLE or
INTEGER arrays with dimensions identical to those of the (existing)
VarLike array. IMAGES and TABLES can also be defined in this way. The
dimension field must not be specified in such a case.

The /LIKE option is incompatible with the DEFINE FUNCTION and DEFINE
HEADER commands.

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

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

3.10 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
text editor to edit this file. If a file name is given, the specified file is
editted.

Different editors are available: TPU and EDT on VMS systems, Emacs and
Vi on Unix systems, possibly more if your system manager has defined
others. The default behaviour is the following:
- On VMS systems, the TPU editor is called, with the default section
as defined by logical name TPUSECINI. TPUSECINI usually points to a
section using the TPU editor with EDT interface and additional
features added by the Groupe d'Astrophysique such as double window
mode, but you may have redefined it to your own section.
- On Unix systems, Emacs with EDT mode is normally used.

The behaviour can be changed using the command SIC EDIT, or specifying at system level a logical name for GAG_EDIT. For example to use the EDT editor, you could type

SIC> SIC EDIT EDT

within a program using SIC. EDT will then be used for any EDIT command typed during the program execution. Alternatively, you could define your default SIC editor by, on VMS systems, using the DCL command

DEFINE GAG_EDIT EDT

on Unix systems, inserting the following command in your $HOME/.gag.dico file

GAG_EDIT emacs

3.11 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"
4 SIC Programming Manual

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

"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).
5 SIC ERROR MESSAGES AND RECOVERY PROCEDURES

List of Error Messages

5.1 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.

5.2 D

E-DECOD, 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-DECOD, 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-DECOD, Error computing <String>
SIC, Argument decoding routines. Some error occurred 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 occurred.

E-DECOD, 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-DECOD, 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-DECOD, Missing argument number <Integer> of Command <String>

E-DECOD, 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.
5 SIC ERROR MESSAGES AND RECOVERY PROCEDURES

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 increase 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.

5.3 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, occurred in <String> at line <Integer>

I-ERROR, occurred 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, occurred in Program
SIC, error traceback facility. An error occurred in subroutine mode.
User action: signal the error to the programmer.

I-ERROR, occurred 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.

5.4 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.

5.5 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 occurred 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.

5.6 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.

5.7 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.
5  

SIC ERROR MESSAGES AND RECOVERY PROCEDURES

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: Correct 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.
5  SIC ERROR MESSAGES AND RECOVERY PROCEDURES

5.8  J through L

E-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_INT). 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 assignment
   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.
5 SIC ERROR MESSAGES AND RECOVERY PROCEDURES

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.

5.9 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 any (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.

5.10 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 occurred, 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 occurred, 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.
5 SIC ERROR MESSAGES AND RECOVERY PROCEDURES

5.11 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
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 ENDF 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 ANSI terminal
SIC, SIC command. Information message: the user requested the EDIT mode, but is not logged on
a ANSI (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 accommodate 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.

E-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.

5.12 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 occurred 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.

5.13 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.
5 SIC ERROR MESSAGES AND RECOVERY PROCEDURES

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, 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-ZCRONGNEU!NEU! j'y arrive pas
   Congratulations, you got a free bottle of champagne if...
   you can reproduce the error.
   User action: Contact the authors.
Index

ACCEPT, 14, 24
/ARRAY, 14, 24
/BINARY, 14, 24
/COLUMN, 14, 24
/FORMAT, 24
/LINE, 25

BEGIN, 25
BREAK, 16, 25

COMPUTE, 26, 37
CONTINUE, 16, 17, 27

DEFINE, 8, 28
/LIKE, 33
/CHARACTER, 29
/COMMAND, 22, 29
/Doubles, 29
/FLTS, 29
/Function, 30
/HEADER, 12, 30
/IMAGE, 12, 31, 38
/INTERGER, 32
/LOGICAL, 32
/REAL, 32
/STRUCTURE, 32
/TABLE, 12, 32

DELETE, 33
/SYMBOL, 8
/VARIABLE, 13

EDIT, 6, 33
ELSE, 15, 34
ELSE IF, 15
ENDIF, 15
EXAMINE, 8, 14
/Function, 9
EXIT, 17

FOR, 15
/WHILE, 15

GUI Mode, 18
GUI E30FBU TTON, 18, 20
GUI E30FGO, 18, 19
GUI E30FMENU, 18
GUI E30FPANEL, 18, 20
/DETACH, 18

HELP, 5, 6

IF, 15

LET, 8, 13
/CHOICE, 19
/FILE, 19
/INDEX, 19
/LIKE, 11
/NEW, 8
/PROMPT, 19
/RANGE, 19
/WHERE, 11

Main input window, 18

NEXT, 15, 17

ON ERROR, 17
Optional window, 18

PANEL
/DETACH, 18
PAUSE, 17
QUIT, 17

RECALL, 6
RETURN, 17
BASE, 17
ERROR, 17

SAY, 14, 23
SIC, 20, 23
COPY, 21
DELETE, 21
DIRECTORY, 21, 22
EDIT, 7, 23
EXTENSION, 15, 23
HELP, 23
LOGICAL, 22, 23
MEMORY, 23
OUTPUT, 23
PRECISION, 8, 23
RENAME, 21
VERIFY, 23

SYMBOL, 8
SYSTEM, 20–22, 27
/PROCESS, 21
UNIX, 28
VMS, 27

TYPE, 15

54