INCLAN

INCLAN is a powerful interactive command language that allows the use of variables, FORTRAN-77 mathematical and character expressions, macros, flow control (loops, conditional statements, jumps), parallelization, the production of graphics etc.

When reading an input command line the command interpreter executes the following steps:

- An optional comment, i.e. text following a comment sign “#”, is discarded.
- The values of variables are substituted from right to left.
- The command line is split into elements (defined as sequences of non-blank characters separated by blank characters). The first element becomes the command name, and the following elements become command parameters.
- If the command name matches a user-defined alias, the alias is expanded.
- If the command name matches a built-in command of INCLAN, it is executed by the command interpreter itself.
- Otherwise, if the command name matches a user-defined command, it is executed by the command interpreter.
- Otherwise, if the command name matches a command of the program unambiguously, it is executed by the program.
- Otherwise, the command interpreter looks for a macro with the given command name and, if it is found in the current macro search path, executes it. If no such macro is found, an error occurs.

Special characters

The following characters have a special meaning for INCLAN. To use them literally, they usually must be preceded by a backslash.

$

“**$***variable*” substitutes the value of the *variable* in the command line. Substitutions proceed from left to right. If the value of the variable or function call starts and ends with single quotes (i.e. if it is a FORTRAN-77 character string), the delimiting single quotes are removed before inserting the value.

%

“**%***variable*” substitutes the value of the *variable* in the command line. Substitutions proceed from left to right. Single quotes that delimit FORTRAN-77 character strings are retained.

{ }

The curly braces in “**{$***variable***}**” or “**{%***variable***}**” separate the variable name *variable* from immediately following text. “**${***expression***}**” or “**%{***expression***}**” substitute the result value of the FORTRAN-77 *expression*.

( )

“**$***variable***(***format***)**” uses the given FORTRAN-77 *format* to convert the numeric value of a variable into the string that is substituted in the command line. If the value of the *variable* is a comma-separated list, “**$***variable***(***n***)**”, where *n* is an integer expression, substitutes with the *n*-th element of this list. “**$***variable***(***m***:***n***)**”, where *m* and *n* are integer expressions, substitutes with the substring between positions *m* and *n* of the value of the *variable*. These three possible uses of parentheses cannot be used simultaneously.

;

separates commands that stand on the same line. Note, however, that commands that form blocks (e.g. **do** . . . **end do**, **if** . . . **end if**) must always appear as the first command on a line.

:

“*Label***:**” denotes a label that can be used as the target of a jump in a **goto** statement.

\

“**\\***c*” treats the character *c* literally and allows the use of special characters in normal text, “\” at the end of a line indicates that the statement continues on the following line.

"

"*text***"** treats *text* as a single parameter, even if it contains spaces. Variable substitutions in the *text* still occur.

' — **'***text***'** treats *text* as a single parameter; the single quotes remain part of the text. Single quotes are used to delimit FORTRAN-77 character string constants. Variable substitutions in the *text* still occur.

— Text between a comment sign "**#**" and the end of the line is treated as a comment and skipped by the program.

@ — Commands preceded by "**@**" are only echoed if the variable **echo** has the value **full** or **FULL**. "**@**" has its special meaning only if it occurs as the first character of a command.

! — "**!***string*" recalls the last interactive command that started with *string*. "**!**" has its special meaning only if it occurs as the first character of a command.

^ — "**^***string***^***replacement***^**" executes the last interactive command again after replacing the first occurrence of *string* by *replacement*. The third caret is optional unless the *replacement* string has trailing blanks. "**^**" has its special meaning only if it occurs as the first character of a command.

Variables

The command line interpreter allows the use of variables in two different ways:

- Similar to shell-variables in the UNIX operating system as variables whose value can be substituted into the command line. In this case, the value of a variable is a general character string and has no particular type.
- As variables in FORTRAN-77 expressions. In this case, the value of a variable must be an integer, real, complex, logical or character constant, according to the rules of FORTRAN-77. In particular, character strings must be delimited with single quotes.

Variables can be used in both ways simultaneously which makes them a powerful tool of the command language.

A variable *name* consists of up to 32 letters, digits, or underscore characters "_". The *value* of a variable is always stored as a character string and only converted temporarily to an integer, real, or complex number during the evaluation of a FORTRAN-77 expression.

There are several types of variables:

Local variables

exist only within the macro where they are declared, and in macros called from this macro. With the exception of the command line parameters of a macro, which are always local, local variables must be declared in **var** or **syntax** statements. They exist until they are removed with **unset** or until the end of the macro in which they are declared is reached.

Global variables

are always visible, except when they are hidden by local variables with the same name. Variables that are not local are global. The user can introduce new global variables simply by using a variable with a new name. Global variables exist until they are removed with **unset**.

Special variables

are variables that can be created and used by the user but have also a special meaning to the command interpreter.

System variables

are variables that are used and, possibly, set by the program (not exclusively by the user with **eval**, **set** etc.). System variables are always global.

There are several ways to insert the value of a variable or the result value of an expression into the command line:

Basic substitutions

Substitutions of the form **$***variable* or **%***variable* insert the complete value of the variable (without trailing blanks) into the command line. Substitutions with “**$**” differ from those with “**%**” only if the value of the variable starts and ends with single quotes, i.e. if it is a FORTRAN-77 character constant: with “**%**” the delimiting single quotes are retained in the substitution, with “**$**” they are removed. A variable name that is immediately followed by a letter, digit, or underscore character must be enclosed in curly braces: “**{$***variable***}**”.

```
x:=4.6; y:=2.0; sum=x+y; t:=a sum             Set variables
print "This is $t: $x + $y = $sum"        Substitute values
This is a sum: 4.6 + 2.0 = 6.60000

s:=’$t’            Create a FORTRAN-77 string from a normal variable
print "\$s = $s; \%s = %s"        With and without single quotes
$s = a sum, %s = 'a sum'

print "{$t}mer"
a summer
```

Fortran format

Substitutions of the form **$***variable***(***format***)** or **%***variable***(***format***)** are used to format integer or real values of variables according to a FORTRAN-77 format. A *format* that contains the letter “I” or “i” applies to integer numbers, all other *formats* to real numbers.

```
x:=4.6; y:=2.0; sum=x+y
print "$x + $y = $sum(E12.3)"
4.6 + 2.0 =    0.660E+01
```

Substring

Substitutions of the form **$***variable***(***n***:***m***)** or **%***variable***(***n***:***m***)**, where *n* and *m* are positive integer expressions, are used to substitute with the substring between character positions *n* and *m* of the value of a *variable*. Substring expressions can also appear on the left hand side of assignment statements.

```
t:=a sum
print "another $t(3:5)"
another sum
t(3:)::=program                         Assignment to a substring
print "$t"
a program
```

List element

If the value of a *variable* is a comma-separated list, "**$***variable***(***n***)**" or "**%***variable***(***n***)**", where *n* is a positive integer expression, substitute with the *n*-th element of this list.

```
s:=17,28,,56,"This is the end"
do i 1 length('s')        length returns the number of elments
  print "Element $i: $s(i)"
end do
Element 1: 17
Element 2: 28
Element 3:
Element 4: 56
Element 5: This is the end
```

Function call

"**$***function*" or "**%***function*" substitute with the result value of a *function* without parameters, "**$***function***(***parameters***)**" or "**%***function***(***parameters***)**" substitute with the result value of a *function* with *parameters*. If there are several *parameters*, they are separated by commas.

```
x=2.5; print "log(x)= $log(x)"
log(x) = 0.916291
```

Expression

"**${***expression***}**" or "**%{***expression***}**" substitute with the result value of an *expression*.

```
x=2.5; y=10.0; print "x/y = ${x/y}"
x/y = 0.250000
```

All substitutions in the command line proceed from right to left. This allows, for example, to compose a variable name from the values of other variables before it is used in a substitution.

```
command list_param             User-defined command list_param
  do i 1 nparam
    print "Parameter $i: $p$i"
                          $p$i inserts the value of the i-th command line
                          parameter.
  end do
end

list_param 17 second last                          Call list_param
Parameter 1: 17
Parameter 2: second
Parameter 3: last
```

Special variables

The following variables have a special meaning for the command interpreter:

echo determines which commands are echoed, i.e. copied to standard output before execution. The possible settings are:

NULL (or not set at all) In macros, all commands except those built into the command line interpreter are echoed; interactive commands are not echoed.

off Commands are not echoed.

on Both in macros and interactively, all commands except those built into the command line interpreter are echoed.

large Same as **on**, except that the echo is surrounded by blank lines.

full All commands are echoed, and the corresponding line numbers in macros are given.

OFF Same as **off**, except that this setting can only be overridden by another value written in capital letters.

ON Same as **on**, except that this setting can only be overridden by another value written in capital letters.

LARGE Same as **large**, except that this setting can only be overridden by another value written in capital letters.

FULL Same as **full**, except that this setting can only be overridden by another value written in capital letters. This setting is particularly useful for debugging macros in which the echo is suppressed.

Labels are not included in the echo, but variable substitutions are. Statements preceded by “@” are only echoed if **echo** has the value **full** or **FULL**.

erract is a variable for error handling in macros. If an error occurs within a macro, the value of **erract** is executed as a command. By default the **exit** command is executed, i.e. the program returns to interactive input. Setting erract has no effect on errors produced by interactively entered commands: These errors are always reported and the program continues with the execution of the next statement.

```
set erract="show; quit"
```

In case of an error in a macro a listing of all global variables is given, and the program is stopped. Such error handling can be useful if the program is used non-interactively.

info determines which messages are written to standard output and into the protocol file. The possible settings are:

none No messages are written.

minimal A minimal set of messages is written, in general a single line for each command that is executed.

normal The “normal” amount of messages is written.

full The “full” amount of messages is written.

debug The “full” amount of messages and additional undocumented messages for debug purposes are written.

Optionally, this variable may have two of the above values, separated by a comma. In this case, the first value applies to standard output, the second to the protocol file.

nparam denotes the number of command line parameters of the current macro.

nproc denotes the maximal number of processors that will be used for the execution of parallel do-loops.

p1, **p2**, . . . are the default names for the command line parameters of a macro. These names may be changed at the beginning of the macro.

path denotes the search path for macro files in the form of a comma-separated list of directories.

prompt denotes the prompt for interactive input. If this variable is not defined or blank, no prompt is written but multiple blank lines of input and the end of the execution of a macro are indicated by the word “Ready” on a separate line.

protocol denotes the name of the protocol file into which standard output is duplicated under the control of the variable **info**. If this variable is not defined or blank, no protocol file is written.

timing is a system variable to control the reporting of CPU times. CPU times are given for all commands (except for those that are built into the command line interpreter) that need more seconds of CPU time than the value of **timing** indicates.

Expressions

The command interpreter can evaluate general FORTRAN-77 integer, real, complex, logical and character expressions. Expressions can appear in **eval** statements, as conditions of **if** statements, as command parameters when a numeric value is expected, and as substring and element index expressions.

An expression is built according to the rules of FORTRAN-77 from constants, variables, and function calls. These basic items can be combined by operators (“**+**”, “**–**”, “*”, “**/**”, “**”, “**.eq.**”, “**.ne.**”, “**.lt.**”, “**.le.**”, “**.ge.**”, “**.gt.**”, “**.and.**”, “**.or.**”, “**.not.**”, “**.eqv.**”, “**.neqv.**”, “**==**”, “**!=**”, “**<**”, “**<=**”, “**>=**”, “**>**”) and grouped by parentheses.

There are the following differences to the rules of FORTRAN-77:

- The data type “double precision” is not supported.
- The data type “logical” is represented by the integer values 0 (false) and 1 (true). Any integer expression can be used in place of a logical expression, with 0 representing “false”, and all other values representing “true”.
- Variable, function and operator names are case sensitive. The names of logical operators and intrinsic functions must be written in lower

case.

- The logical operators “**==**”, “**!=**”, “**<**”, “**<=**”, “**>=**”, “**>**”, “**&&**”, “**||**”, and “**!**” can be used in place of its respective FORTRAN-77 equivalents “**.eq.**”, “**.ne.**”, “**.lt.**”, “**.le.**”, “**.ge.**”, “**.gt.**”, “**.and.**”, “**.or.**”, and “**.not.**”.
- All FORTRAN-77 intrinsic functions (except “dble”, “dprod”, “lge”, “lgt”, “lle” and “llt”) are available by their generic names but not under special names. For example, the absolute value function is known by the name “**abs**” but not by the special names “iabs” or “cabs”.
- There are additional intrinsic functions (see below).
- Blanks can only appear at “reasonable” places but not inside of numbers, variable names etc.

Intrinsic functions

In the following list of all INCLAN intrinsic functions, arguments are denoted by

n	integer
r	real
c	complex
s	string
x	integer or real, unless types are given explicitly
z	real or complex

The result type of an intrinsic function is only given explicitly if it differs from the type of the argument(s).

abs(*x*)

Absolute value; the argument *x* is of any numeric type, for complex arguments the result is real.

acos(*r*)

Arc cosine; $|r| \le 1$, $0 \le \mathrm{acos}(r) \le \pi$.

aimag(*c*)

Real function that returns the imaginary part of *c*.

aint(*r*)

Discard fractional part; the result if of type real.

anint(*r*)

Closest integer; the result if of type real.

asin(*r*) Arc sine; $|r| \le 1$, $-\pi/2 \le \mathrm{asin}(r) \le \pi/2$.

atan(*r*) Arc tangent; $-\pi/2 \le \mathrm{atan}(r) \le \pi/2$.

atan2(r_1,r_2) Argument of the complex number $r_2 + ir_1$ (not $r_1 + ir_2$!); r_1 and r_2 must not both be zero, $-\pi \le \mathrm{atan2}(r_1, r_2) \le \pi$.

char(*n*) Character function that returns the character with number *n*.

cmplx(x_1,x_2) Complex function that returns $x_1 + ix_2$; both arguments must have the same type.

conjg(*c*) Complex conjugate.

cos(*z*) Cosine.

cosh(*r*) Hyperbolic cosine.

cputime Real function that returns the CPU time (in seconds) since the start of the program.

date Character function that returns the current date in the form *dd*–*mm*–*yy*.

def(*s*) Logical function that returns 1 if a variable with name *s* exists and has a value different from **NULL**, or 0 otherwise.

dim(x_1,x_2) Positive difference; $\mathrm{dim}(x_1, x_2) = \max(x_1 - x_2, 0)$.

exist(*s*) Logical function that returns 1 if a variable with name *s* exists, or 0 otherwise.

existfile(*s*) Logical function that returns 1 if a file with name *s* exists, or 0 otherwise.

exp(*z*) Exponential function.

external(*s*) Character function that returns the value of the external (i.e. non-local) variable with name *s* (even if it is hidden by a local variable with the same name), or a blank string if no external variable with this name exists.

external(s_1,s_2) Character function that returns the value of the external (i.e. non-local) variable with name s_1 (even if it is hidden by a local variable with the same name), or s_2 if no external variable with the name s_1 exists.

fitchisq

Real function that returns the χ^2 value of the last linear least-squares fit (see plot subcommand **fit**).

fiterr(*n*)

Real function that returns the standard deviation of the *n*-th fit parameter of the last linear least-squares fit (see plot subcommand **fit**).

fitpar(*n*)

Real function that returns the optimal value of the *n*-th fit parameter of the last linear least-squares fit (see plot subcommand **fit**).

fitprob

Real function that returns the probability that the χ^2 value of the last linear least-squares fit would be exceeded by chance (see plot subcommand **fit**).

getenv(*s*)

Character function that returns the value of the environment variable with name *s*.

getpid

Integer function that returns the UNIX process identification number of the current process.

global(*s*)

Character function that returns the value of the global variable with name *s* (even if it is hidden by another variable with the same name), or a blank string if no global variable with this name exists.

global(s_1,s_2)

Character function that returns the value of the global variable with name s_1 (even if it is hidden by another variable with the same name), or s_2 if no global variable with the name s_1 exists.

ichar(*s*)

Integer function that returns the number of the character *s*.

if(n,x_1,x_2)

Function that returns the argument x_1 if $n \neq 0$, or x_2 otherwise. The arguments x_1 and x_2 can have any type.

index(s_1,s_2)

Integer function that returns the starting position of the first occurence of the string s_2 in the string s_1, or zero if s_2 does not occur as a substring in s_1.

indexr(s_1,s_2)

Integer function that returns the starting position of the last occurence of the string s_2 in the string s_1, or zero if s_2 does not occur as a substring in s_1.

int(*z*)

Integer function that returns the integer part of the real or complex number z.

len(*s*) Integer function that returns the number of characters in *s*.

length(*s*) Integer function that returns the number of elements in the array stored in a variable with name *s*.

lenstr(*s*) Integer function that returns the index of the last non-blank character in *s*.

log(*z*) Natural logarithm; $z \neq 0$, if *z* is real it must be positive, for complex *z* the result has $-\pi < \operatorname{Im} \log(z) \leq \pi$.

log10(*z*) Logarithm to base 10; $z \neq 0$, if *z* is real it must be positive, for complex *z* the result is in the range $-\pi < \operatorname{Im} \log 10(z) \leq \pi$.

macro(*s*) Logical function that returns 1 if a macro with name *s* is available, or 0 otherwise.

match(s_1,s_2) Wildcard match; logical function that returns 1 if the string s_2 matches the string s_1 , or 0 otherwise. The string s_2 may contain wildcards: an asterisk matches zero or more characters, and a question mark matches exactly one character.

max(x_1,x_2,...) Maximum.

min(x_1,x_2,...) Minimum.

mod(x_1,x_2) Remainder of x_1 modulo x_2 ; $\operatorname{mod}(x_1, x_2) = x_1 - x_2 \cdot \operatorname{int}(x_1 / x_2)$, both arguments must have the same type, $x_2 \neq 0$.

mtime(*s*) Integer function that returns the time of last modification (in seconds since a reference date) of the file with name *s*.

nint(*r*) Integer function that returns the integer closest to *r*.

opened(*s*) Logical function that returns 1 if a file with name *s* is currently open, or 0 otherwise.

plotx0, **ploty0**, **plotx1**, **ploty1** Real functions that return the coordinates of the two reference points (X_0, Y_0) and (X_1, Y_1) in the user coordinate system used for graphics (see plot parameters **X0**, **Y0**, **X1**, **Y1**).

rand Real function that returns a pseudo-random number; pseudo-random numbers are uniformly distributed between 0 and 1.

rand(*n*)

Real function that returns a pseudo-random number; pseudo-random numbers are uniformly distributed between 0 and 1. The random number generator is initialized with the seed *n*.

rand(n_1,n_2)

Real function that returns a pseudo-random number; pseudo-random numbers are uniformly distributed between 0 and 1. The random number generator is initialized with the seed n_1, and the result is the n_2-th random number generated from this seed.

real(*x*)

Conversion to real type; the argument *x* must be of type integer or complex, for complex *x* the real part is returned.

sign(x_1,x_2)

Returns the absolute value of x_1 times the sign of x_2; if $x_2 = 0$, its sign is taken as positive, both arguments must have the same type.

sin(*z*)

Sine.

sinh(*r*)

Hyperbolic sine.

sqrt(*z*)

Square root; if *z* is real, it must be non-negative.

tan(*z*)

Tangent.

tanh(*r*)

Hyperbolic tangent.

time

Character function that returns the current time in the form *hh*:*mm*:*ss*.

val(*s*)

Character function that returns the value of the variable with name *s*, or a blank string if no variable with this name exists.

val(s_1,s_2)

Character function that returns the value of the variable with name s_1, or s_2 if no variable with the name s_1 exists.

walltime

Integer function that returns the number of seconds since the start of the program.

Macros

Macros are files containing INCLAN statements. A macro is called by its name that is identical to its filename except for the extension ".dya" that is required for macro files. INCLAN looks for macro files in the directories given by the special variable **path**, or in the explicitly given directory. Command line parameters may be passed into a macro. Within the macro, they are available as local variables that are by default called **p1**, **p2**, ... These variable names can be changed with the **parameter** statement. The local variable **nparam** denotes the number of command line parameters. Macros can be called from within other macros. On-line help information may be included into a macro as lines that start with two comment signs "##". Such lines are copied to standard output when one requests help about a macro with the command **help** *macro*.

The special macro **init** is an initialization macro that is automatically executed when the program starts. Typically, this macro sets the system variable **path** that defines the search path for macro files.

Standard output

This section explain the ways by which commands can write output to the standard output device (in the following simply called "screen") and/or to disk files by using the protocol mechanism or output redirection. The concepts of this section do not apply to output that is written to explicitly named disk files by specific output commands.

Information level

All output has an importance level, and only output that is "important enough" is actually written. The definition of what is "important enough" is given by the special variable **info** that can, in its simple form, take one of five *information level* values:

none	no output at all, except for error messages
minimal	minimal output, in general a one line confirmation
normal	the "normal" amount of output
full	detailed output
debug	additional undocumented debugging output

Protocol file

The output can be duplicated into a protocol file. In fact, different **info** values might be used for output to the screen and to the protocol file. In this case, the info value consists of two simple info values, separated by a comma. A protocol file is written if the **protocol** variable is defined and has a non-blank value that is the name of the protocol file. If the file does not exist when the **protocol** variable is set to the corresponding name, it is created; otherwise the output is appended to an existing protocol file.

```
protocol:=logfile                 Open protocol file “logfile”
info:=minimal,full           Minimal screen output, full protocol
...
protocol:=                                    Close protocol file
```

Output redirection

Output from a command is redirected to a given file if the last parameter of the command is

>*file* Redirect to a new *file*, or overwrite existing *file*. After writing the output, the file remains open.

>*file.* Redirect to a new *file*, or overwrite existing *file*. After writing the output, the file is closed.

>>*file* Append to an existing *file*, or create new *file.* After writing the output, the file remains open.

>>*file.* Append to an existing *file*, or create new *file.* After writing the output, the file is closed.

Blanks between **>** and *file* are not allowed and that the file name must not end with “**.**”. The file name is optional; if it is omitted, the output will be redirected to the previously used *file*. When redirection is used, all output that would otherwise be sent to the screen is written to the given *file*. Standard output and the protocol file are not used.

Built-in commands

The following commands are built into the command interpreter. Their names cannot be abbreviated.

alias

[*name statement*]

Defines a new alias *name*, i.e. an abbreviation, for the given *statement*. The *statement* may contain an asterisk “*” to indicate where the command line parameters are to be inserted. Without parameters, **alias** gives a list of all currently defined aliases.

```
alias ? "print \"\%{*}\""      Simulate a pocket calculator
? 5*7
35
```

ask

prompt variable . . .

Writes the string *prompt* to standard output, reads one line from standard input, and assigns from this line strings separated by blanks to the given variables. The command is usually used for interactive input within macros. A *prompt* that contains blanks must be enclosed in double quotes.

```
ask "First and last point:" begin end
First and last point:
12 45
print "range = $begin...$end"
range = 12...45
```

break

Breaks a do-loop and is only allowed in macros. The execution of the macro is continued with the first statement following the loop.

command

[*name*]

Defines a new globally visible user-defined command within a macro, i.e. a macro within a macro. User-defined commands defined by **command** statements are called by their *name*, possibly followed by parameters, in exactly the same way as macros. Within a macro, a user-defined command can only be called after it was defined. The statement *command* without parameters gives a list of all user-defined commands, and indicates where they are defined.

do

(without parameters) Executes a loop within a macro. The loop is executed unconditionally, i.e. until one of the statements **break, exit, quit** or **return** is encountered.

```
do
  if (filename.eq.' ') break
  ...
end do
```

do

variable start end [*step*] [**parallel** [**continue**]]

Executes a FORTRAN-77 do-loop within a macro. The loop counter *variable* and the integer expressions *start, end,* and *step* have the usual meaning. **Parallel** loops are executed in parallel on **nproc** processors. If the keyword **continue** is present, the program continues immediately with the execution of the next statement after the parallel loop. Otherwise, the next statement after the loop is executed when the parallel loop is finished.

```
do i 1 10
  print "Iteration $i."
end do
```

else

Starts an else clause of a block if-statement.

else if

(*condition***) then**

Starts an else-if clause of a block if-statement.

end

Ends a user-defined command or subroutine.

end do

Ends a do-loop.

end if

Ends a block if-statement.

error

text

Writes the *text* to standard output or into the file with the given *filename* and calls the error handler. This statement is suitable to treat errors that occur during the execution of a macro. If the *text* contains blanks it must be enclosed in double quotes.

eval

variable **=** *expression*

Evaluates the arithmetic or string *expression* according to the rules of FORTRAN-77 and assigns the result to the *variable*. The keyword **eval** can be omitted. In contrast to FORTRAN-77 function names must be given in lowercase letters.

```
eval i = 7
sentence = ’A flexible program!’
j = mod(i,4)**2
l = len(sentence)
show i sentence j l
... Variables:
    i        = 7
    sentence = ’A flexible program!’
```

```
j        = 9
l        = 19
```

external

variable **=** *expression*

or

variable **:=** *value*

assigns a *value* (i. e. a string) or the result of an *expression* to an external (non-local) *variable* even if a local variable with the same name exists. This command can be used to return values from a macro to the calling macro.

```
command swap a b                Command to swap two variables
  var x y                       Declare two local variables, x and y
  x=$external('$a')  Get value of external variable with name $a
  y=$external('$b')  Get value of external variable with name $b
  external $a=y      Assignment to external variable with name $a
  external $b=x      Assignment to external variable with name $b
end

x=10; y=5
print "Before swap: x = $x, y = $y"
Before swap: x = 10, y = 5
swap x y
print "After swap : x = $x, y = $y"
After swap : x = 5, y = 10
```

exit

Returns from a macro to interactive input. Given interactively, it exits from the program.

go to

label

continues execution of a macro at the first line that begins with the *label*. Jumps into loops (**do** . . . **end do**) or conditionally executed statements (**if** . . . **else** . . . **end if**) are not allowed and can lead to unpredictable results. A *label* may consist of letters, digits, and underscore characters "_". A label must be followed by a colon.

```
go to cont
  ...
cont: print "Now at label cont."
```

help

[*topic*]

Gives on-line help for a given *topic*. With no *topic* given, a list of all available help topics is displayed. On-line help for macros can be included in the macro: **help** *macro* shows all lines of the *macro* that start with "##".

if

(*condition***)** *statement*

Executes a logical "if" statement as in FORTRAN-77, i. e. the *statement* is executed if the logical expression *condition* is true. A line with a logical "if" statement must not end with the word **then**.

```
i=-56
if (i.lt.0) print "$i is negative."
-56 is negative.
```

if

(*condition***) then**

Executes a block-"if" statement, as in FORTRAN-77.

```
if (mod(i,2).eq.1) then
  print "$i is an odd number."
else if (def('x') .and. exist('y')) then
  print "x is defined, and y exists."
else if (s.eq.' ') then
  print "The variable s is blank."
end if
```

parameter

variable . . .

Changes the names of the parameters that are passed to a macro; i. e. the parameters **p1**, **p2**, . . . get the names given in the **parameter** statement. The **parameter** statement must precede all other statements in a macro (except **var)** and cannot be used interactively.

plot

subcommand [*parameter* . . .]

Performs a plot subcommand. Plot commands are described separately in the "Graphics" section of this chapter.

print

text [**level=***level*]

Writes the *text* to standard output or into the file with the given *filename*. If the *text* contains blanks it must be enclosed in double quotes. Optionally, the importance **level** of the output can be defined. By default, the importance level is **normal**.

quit

Exits from the program.

readline

file variable [**close**]

Reads one line from a *file* and assigns it to a *variable*. If the file is not yet open, it is opened and the first line is read. If the file is already open, the next line is read. If the end of the file is reached, the variable is set to **EOF** and the file is closed. Optionally, the file can be **close**d after reading a line.

remove

file . . .

Removes one or more disk files.

return

exits from the current macro and returns to the calling macro or, if the macro was called interactively, to interactive input. Given interactively, **return** exits from the program.

set

variable **=** *value*

or, if the keyword set is omitted

variable **:=** *value*

assigns a *value* (i. e. a string) to a *variable*.

```
set i=456
j := 2 + i          Short form of set assigns a string value
k = 2 + i           Short form of eval evaluates an expression
set i j k
i = 456
j = 2 + i
k = 458
```

set

variable . . .

Displays values of *variables*. If no *variable* is specified, all variables that have values different from **NULL** are displayed. If the names of one or several *variables* are given, the values of these variables are displayed.

show

variable . . .

Displays the values of all or selected *global* variables. If no *variable* is specified, all global variables that have values different from **NULL** are displayed. If the names of one or several global *variables* are given, the values of these variables are displayed.

sleep

t

Waits for *t* seconds.

subroutine

name

Defines a new user-defined command within a macro, i.e. a macro within a macro. User-defined commands defined by **subroutine** statements are called by their *name*, possibly followed by parameters, in exactly the same way as macros. User-defined commands defined by a **subroutine** statement are local to the current macro (or macros called through it). Within a macro, a user-defined command can only be called after it was defined.

syntax

format . . .

Analyzes the command line parameters of the current macro. This statement can only be called within a macro. Command line parameters that match with one of the *format* specifications are removed from the list of command line parameters and assigned to a new local variable.

The possible *format* items are:

name=[=]*type*[=default]

Declares a named parameter with the given *name*, *type* and, optionally, *default* value. If the *default* value is absent, the parameter is required, and an error will occur if the parameter is not specified in the macro call.

The optional second “**=**” sign after the *name* indicates that a parameter that matches *name* but does not contain an “**=**” sign is not recognized, otherwise (with only one “**=**” sign after *name*), an error occurs in this situation.

A local variable with the given *name* is created, and either the value specified by the user, or, in its absence, the *default* value is assigned to it. The value must be compatible with the given *type* (see below).

In a macro call, a named parameter can either be specified anywhere in the parameter list in the form “*name*=*value*” or as a positional parameter of the form “*value*” at the same position in the parameter list as the corresponding *format* in the **syntax** statement. Only parameters that appear before “*” or “**” (see below) can be specified as positional parameters without giving their *name*.

A *name* may contain an asterisk “*” to indicate how much it can be abbreviated. By default, all unambiguous abbreviations are allowed. If a *name* starts with an asterisk, then the corresponding parameter is a positional parameter that cannot be given in the form “*name*=*value*”.

name Declares a literal option with the *name*. A local variable with the given *name* is created. If the option *name* is present in the macro call this variable is set to 1 (i.e. the logical value “true”), otherwise it is set to 0.

$name_1$|$name_2$ . . .

Declares a set of mutually exclusive literal options with the names $name_1$, $name_2$, etc. Local variables with the given *names* are created. If one of the option names is present in the macro call, the corresponding variable is set to 1 (i.e. the logical value “true”) and the other variables are set to 0.

** Allows for additional parameters that do not match with one of the *formats*.

* Has the same meaning as “**” except that additional parameters must not contain an “**=**” sign.

Formats must not contain blanks.

A *type* can be one of the following:

*	Any character string.
@i	Integer expression.
@r	Real expression.

[*l*<[=]]@**i**[<[=]*u*]

[*l*<[=]]@**r**[<[=]*u*]

Integer or real expression in the given range.

@ii Integer range, i.e. one of the following:

m	a single integer expression
m..*n*	two integer expressions
m..	using the default value for *n*
..*n*	using the default value for *m*

name$_1$|*name*$_2$ . . .

List of mutually exclusive literals.

@f.*extension* Filename that will be extended with the given *extension*, if necessary (*extension* can also be **$***name* to denote the value of a preceding parameter).

```
command read_file
  syntax format=asc|bin file=@f.$format \
         weight=@r=1.0
```

The command **read_file** has three parameters. The first parameter (**format**) is required and can either be **asc** or **bin**, the second parameter (**file**) is also required and is a filename that will be given the extension **.asc** or **.bin**, depending on the chosen format, and the third parameter (**weight**) is an optional real number with default value 1.0.

```
  ...
end
```

```
read_file asc test
```

Positional parameters and default value for **weight**. Equivalent to setting **format=asc**, **file=test.asc** and **weight**=1.0.

```
read_file file=test format=asc weight=2.0
```

Named parameters in any order.

system

[*UNIX-command*]

Executes a *UNIX-command* by invoking a shell. If no command is specified, an interactive shell is started.

type

macro

displays the macro or user-defined command with the given *name*. Macros in the current path can be listed without giving a path; otherwise the path has to be specified.

unset

variable . . .

Removes one or more variables.

var

variable . . .

declares *variables* as local variables of the current macro. In contrast to normal (global) variables, local variables are only visible within the macro where they are declared and within macros that are called via that macro (except when such a macro declares itself a local variable with the same name). The **var** command must precede any other commands in a macro (except the **parameter** command) and cannot be used interactively.

Graphics

With INCLAN it is possible to produce graphical output in either Postscript of FrameMaker (MIF) format. Graphics is created with the built-in command **plot**. The **plot** command can either be invoked directly, or plot subcommands can be combined with list data in graphics files that can be read with the **plot file** command.

A graphics file can contain one or several blocks of *list data*, i.e. matrices of integer or real numbers in free format. Each row (line) of a list data block must have the same number of entries. The columns of a list data block form vectors called $\boldsymbol{x}$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$,... If a list data block consists of a single column with n numbers, this column is called $\boldsymbol{y}_1$ and an $\boldsymbol{x}$-column with values $1, 2, \ldots, n$ is added implicitly. After reading a block of list data, the graphics system is in *list mode*, and various plot subcommands can be applied to vector expressions formed from the column vectors of the list data block. These vector expressions are general FORTRAN-77 expressions that are evaluated for all vector elements and where the column vectors $\boldsymbol{x}$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$,... are denoted by **x**, **y1**, **y2**,...

Besides list data, a graphics file can contain plot subcommands (and comments starting with **#**) but not other commands; it is not an INCLAN macro.

The following alphabetical list contains all plot subcommands. They are called from INCLAN in the form

plot *subcommand parameters*

and in graphics files in the form

subcommand parameters

Some of the plot subcommands have different parameters in normal and list mode as indicated by “(normal mode)” or “(list mode)” at the right margin.

arc

$x\ \ y\ \ a\ \ [b\ \ [\phi_1\ \phi_2]]$

draws a circle, an ellipse, or part of a circle or ellipse with the center at (x, y), and half axes *a* and *b*. If *b* is omitted, a circle with radius *a* (measured in the *x*-direction) is drawn. Optionally, only the part of the ellipse starting and ending with phase angles ϕ_1 and ϕ_2, respectively, is drawn. The phase angle is 0° on the positive *x*-axis and increases counterclockwise. This command can also be used in list mode, where the parameters are vector expressions.

caro

See section ***mark***.

clip

$x_1\, y_1\ \ x_2\, y_2$

draws a rectangle with corners (x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2) and sets the current clipping path to its border. Subsequent drawing commands will only draw within this rectangular area.

clip

off

resets the clipping path. After this command, graphics will no longer be confined to the rectangular area specified in a previous **clip** command.

close

closes the current output plot file.

comment

text

writes *text* as a comment into the output plot file.

cross

See section ***mark***.

curve

$x_1\, y_1\;\; x_2\, y_2\;\; x_3\, y_3\; x_4\, y_4 \ldots$ (normal mode)

draws a Bézier spline curve defined by the points (x_i, y_i). The total number of points must be $3n + 1$, with integer $n \geq 1$. The resulting curve passes through the points 1, 4, 7,...; the other points guide the curve. Four points define the shape of each segment of the curve: The curve segment leaves (x_1, y_1) along the direction of the straight line connecting (x_1, y_1) with (x_2, y_2) and reaches (x_4, y_4) along the direction of the straight line connecting (x_3, y_3) with (x_4, y_4). The lengths of the lines connecting (x_1, y_1) with (x_2, y_2) and (x_3, y_3) with (x_4, y_4) represent, in a sense, the "velocity" of the path at the endpoints. The curve segment is always enclosed by the convex quadrilateral defined by the four points.

curve

$[[\boldsymbol{x}]\;\; \boldsymbol{y}_1 \ldots]$. (list mode)

draws Bézier spline curves through the points of the given vector expressions $\boldsymbol{x}$, $\boldsymbol{y}_1$,... If no vector expressions are specified, splines are drawn through the points of all list columns. If the $\boldsymbol{x}$-expression is omitted (i.e. if only a single expression, $\boldsymbol{y}_1$, is given), the x-coordinates are taken from the x-column of the list. The number of list points must be $3n + 1$, with integer n.

dot

See section ***mark***.

errorbar

$x\;\; y_1\, y_2$

draws an errorbar defined by the given x- and y-coordinates. This command can also be used in list mode, where x, y_1 and y_2 are three vector expressions.

file

file

reads an input graphics *file* (default extension: **.grf**) containing list data and plot commands and executes the plot commands in the graphics file. Graphics files cannot be nested. If no output plot file is open when the **file** command is executed, and if the first plot command in the graphics file does not open an output plot file explicitly, a new Postscript output plot file with the name *file***.ps** is opened implicitly. An implicitly opened output plot file will be closed when the end of the graphics file is reached.

fit

$\boldsymbol{x}\ \boldsymbol{y}\ \sigma\ f_1 \ldots$ (list mode)

performs a linear least-squares fit of the basis functions given by the vector expressions f_1,... to the data points with x-coordinates, y-coordinates and errors given by the vector expressions $\boldsymbol{x}$, $\boldsymbol{y}$ and σ, respectively. For m basis functions, $f_1, \ldots, f_m$ the optimal linear combination,

$$y(x) = a_1 f_1(x) + \ldots + a_m f_m(x) \quad , \tag{1}$$

is determined by minimizing

$$\chi^2(a_1, \ldots, a_m) = \sum_i \left(\frac{y_i - y(x_i)}{\sigma_i}\right)^2 \quad , \tag{2}$$

where i runs over the list data points. The optimal fit function $y(x)$ is added as another column to the list data. This command does not draw anything. The fit parameters, $a_1, \ldots, a_m$, their standard deviations, χ^2, and the probability that this value of χ^2 would be exceeded by chance are available through the intrinsic functions **fitpar**, **fiterr**, **fitchisq** and **fitprob**, respectively. If the errors σ_i of the data points are unkown, this can be indicated by setting σ to zero in the **fit** command.

```
dot x y1                                     Plot original data points
fit x log(y1) 0 1 x          Logarithmic fit of y = a1 exp(–a2x)
spline x exp(y2)                                   Plot fitted curve
```

frame

xtext=*xtext*	—
ytext=*ytext*	—
tics=*ticaxes*	x,y
labels=*labelaxes*	x,y
grid zero	

draws a rectangular frame with corners (X_0, Y_0) , (X_0, Y_1) , (X_1, Y_0) and (X_1, Y_1) . Subsequently produced graphics is clipped on the borders of the frame. The *x*- and *y*-axes are labeled with the titles *xtext* and *ytext*, respectively. The parameter **tics** and **labels** determines whether tics and numeric labels are drawn. The possible values for *ticaxes* and *labelaxes* are:

off	No labels or tics.
x	Labels or tics only on the *x*-axis.
y	Labels or tics only on the *y*-axis.
x,y	Label or tics on both axes (default).

If the option **grid** is present, a fine grid is drawn. If the option **zero** is present, fine lines will be drawn along $x = 0$ and $y = 0$ (if they fall within the frame).

function

$f_1 \ldots$

plots the functions given by the expressions $f_1(x)$,...

label

axis position text

labels the given *axis* by placing a tic and the *text* at the given *position*. The parameter *axis* can have the following values:

x or **bottom**	Label the *x*-axis, i.e. the horizontal line at *y*-position Y_0 .
y or **left**	Label the *y*-axis, i.e. the vertical line at *x*-position X_0 .
top	Label the horizontal line at *y*-position Y_1 .
right	Label the vertical line at *x*-position X_1 .

If *text* is blank, only a tic is set.

line

$x_1\,y_1\;\;x_2\,y_2 \ldots$ (normal mode) draws a line that connects the points (x_1, y_1), (x_2, y_2),... by straight line segments.

line $[[\boldsymbol{x}]\ \boldsymbol{y}_1 \ldots]$. (list mode)

draws straight lines through the points of the given vector expressions ***x***, ***y***$_1$,... If no vector expressions are specified, straight lines are drawn through the points of all list columns. If the ***x***-expression is omitted (i.e. if only a single expression, ***y***$_1$, is given), the *x*-coordinates are taken from the *x*-column of the list.

mark $x\ \ y$ (normal mode)

where *mark* stands for either **dot**, **square**, **caro**, **plus**, **cross** or **triangle**, marks the position (x, y) with the corresponding symbol. The size of the symbol is determined by the current value of the plot parameter **marksize**.

mark $[[\boldsymbol{x}]\ \boldsymbol{y}_1 \ldots]$. (list mode)

where *mark* stands for either **dot**, **square**, **caro**, **plus**, **cross** or **triangle**, marks the positions given by the vector expressions ***x***, ***y***$_1$,... with the corresponding symbol. If no vector expressions are specified, all points of the list columns are marked. If the ***x***-expression is omitted (i.e. if only a single expression, ***y***$_1$, is given), the *x*-coordinates are taken from the *x*-column of the list.

mif *file*

opens and initializes an output plot *file* in FrameMaker (MIF) format. If another plot file is open when the **mif** command is executed, it is closed.

plus See section ***mark***.

polygon $x_1\, y_1\ \ x_2\, y_2\ \ x_3\, y_3 \ldots$ (normal mode) draws a polygon with the edges (x_i, y_i) . At least three points must be specified.

polygon

$[[\boldsymbol{x}]\ \boldsymbol{y}_1. . .]$. (list mode)

draws polygons with the edges given by the vector expressions $\boldsymbol{x}$, $\boldsymbol{y}_1$,... If no vector expressions are specified, polygons are drawn through the points of all list columns. If the $\boldsymbol{x}$-expression is omitted (i.e. if only a single expression, $\boldsymbol{y}_1$, is given), the x-coordinates are taken from the x-column of the list. The number of list points must be three or more.

ps

file

opens and initializes an output plot *file* in Postscript format. If another plot file is open when the **ps** command is executed, it is closed.

rectangle

$x_1\, y_1\ \ x_2\, y_2$

draws a rectangle with corners (x_1, y_1) , (x_1, y_2) , (x_2, y_1) and (x_2, y_2) . This command can also be used in list mode, where x_1, y_1, x_2 and y_2 are four vector expressions. In list mode, the command can also be used without parameters. In this case a rectangle with corners $((x_{i-1} + x_i)/2, 0)$, $((x_i + x_{i+1})/2, 0)$, $((x_{i-1} + x_i)/2, y_i)$ and $((x_i + x_{i+1})/2, y_i)$, i.e. a histogram bar, is drawn for each point (x_i, y_i) in the list columns (for the first and last point, x_{i-1} and x_{i+1} are replaced by the minimal and maximal x-values, X_0 andX_1 , respectively).

scale

axis $\boldsymbol{f}_1. . .$ **exact** *(list mode)*

performs scaling of the given *axis* (**x** or **y**) on the basis of the vector expressions $\boldsymbol{f}_1$,... Scaling sets the coordinates of the reference points in the user coordinate system (X_0 and X_1 for the x-axis, and Y_0 and Y_1 for the y-axis) such that they include all values of the vector expressions $\boldsymbol{f}_1$,... If the option **exact** is present, then the new coordinates of the reference points will correspond exactly to the minimum and maximum of the vector expressions $\boldsymbol{f}_1$,...; otherwise a small margin will be added in order to avoid that points lie exactly on the boundary.

set

*parameter***=***value* . . .

sets one or several plot *parameters* to the given *values*. The keyword **set** is optional.

shape

$x_1\,y_1\;\;x_2\,y_2\;\;x_3\,y_3\;\;x_4\,y_4\;\;x_5\,y_5\;\;x_6\,y_6$ . . .

(normal mode)

draws a shape enclosed by a closed Bézier spline curve that is defined by the points (x_i, y_i). The total number of points must be 3*n*, with integer $n \geq 2$.

shape

[[***x***] $\boldsymbol{y}_1$. . .].

(list mode)

draws shapes enclosed by Bézier spline curves through the points of the given vector expressions ***x***, $\boldsymbol{y}_1$,... If no vector expressions are specified, shapes are drawn for all list columns. If the ***x***-expression is omitted (i.e. if only a single expression, $\boldsymbol{y}_1$, is given), the *x*-coordinates are taken from the *x*-column of the list. The number of list points must be 3*n*, with integer *n*.

spline

$x_1\,y_1\;\;x_2\,y_2$ . . .

(normal mode)

draws a cubic spline through the points (x_1, y_1), (x_2, y_2),... The spline starts at the first point and ends at the last point with vanishing second derivative. The *x*-values must be increasing: $x_i < x_{i+1}$, for all *i*.

spline

[[***x***] $\boldsymbol{y}_1$. . .].

(list mode)

draws cubic spline curves through the points of the given vector expressions ***x***, $\boldsymbol{y}_1$,... If no vector expressions are specified, splines are drawn through the points of all list columns. If the ***x***-expression is omitted (i.e. if only a single expression, $\boldsymbol{y}_1$, is given), the *x*-coordinates are taken from the *x*-column of the list.

square

See section ***mark***.

text

```
x y text
```

print *text* at position (x, y). The alignment of the text with respect to the reference position (x, y) depends on the current values of the plot parameters **align** and **rotate**. The current values of the plot parameters **font**, **textsize**, **weight** and **angle** define the font used to write the *text*. In addition, the *text* may contain the following embedded text commands:

@T	Change font type to Times.
@H	Change font type to **Arial**.
@C	Change font type to `Courier`.
@S	Change font type to Symbol.
@b	Change to **boldface**.
@i	Change to *italics*.
@^	Start a $^{\text{superscript}}$.
@v	Start a $_{\text{subscript}}$.
@N	Return to standard font, end sub- or superscript.

If the text contains multiple blanks, it must be enclosed in double quotes. Double quotes that are part of the text must be preceded by a backslash.

triangle

See section ***mark***.

write

```
text
```

writes *text* into the output plot file.

Plot parameters are used to define the positioning and appearance of graphics objects. They are set by the plot subcommand **set**:

align

determines how text is aligned with respect to its reference position. Possible values are:

left	The horizontal reference position is at the left margin of the text.
center	The horizontal reference position is in the center of the text.
right	The horizontal reference position is at the right margin of the text.
bottom	The vertical reference position is at the bottom margin of

	the text.
middle	The vertical reference position is in the middle of the text.
top	The vertical reference position is at the top margin of the text.

Horizontal and vertical alignment specifications can be separated by a comma, e.g. **align=center,top.**

Initial value: **left,bottom**.

angle

defines a font property with the possible values:

regular	Regular; not italics.
italics	Italics or oblique.

The Symbol font is only available as **regular**.

Initial value: **regular**.

autoscale

determines whether the user coordinate system is automatically rescaled after reading list data. The possible values are:

off	No automatic scaling.
x	Automatic scaling of the *x*-dimension only.
y	Automatic scaling of the *y*-dimension only.
x,y or **on**	Automatic scaling of both dimensions.

If autoscaling of the *x*-dimension is on, then the values of X_0 and X_1 (plot parameters **X0** and **X1**) are reset after reading list data such that all values in the *x*-column of the list data are in the range between X_0 and X_1 . If autoscaling of the *y*-dimension is on, then the values of Y_0 and Y_1 (plot parameters **Y0** and **Y1**) are reset to include all values in the *y*-columns of the list data. In general, the limits are extended slightly with respect to the exact minimum and maximum in order to avoid that data points lie exactly on the margin.

Initial value: **on**.

border

determines whether the border of a closed figure (a rectangle, a circle, an ellipse, a polygon, a closed Bézier curve, or certain types of marks) will be drawn as a line:

off	Border lines are not drawn.
on	Border lines are drawn.

Initial value: **on**.

color

defines the color, and can have the value **black**, **white**, **red**, **green**, **blue**, **cyan**, **magenta**, or **yellow**. All text and graphics that follows has the given color.

Initial value: **black**.

dash defines the dash pattern used to draw lines. Its value is either blank (which is equivalent to **solid**), or a comma separated list of numbers, or one of the following literals:

solid	Solid lines.
dotted	Dotted lines; equivalent to **1**.
dashed	Dashed lines; equivalent to **5,4**.
dot-dashed	Dot-dashed lines; equivalent to **5,2,1,2**.

General dash patterns are specified by a comma separated list of numbers that define the lengths (measured in points) of alternating solid and invisible stretches.

Initial value: **solid**.

fill defines the fill pattern used to draw areas. Its value is an integer between 0 and 15 with the following meaning:

□	0	Empty; do not fill areas.
■	1	Full color.
■ ■ ■ ■ □ □	2–7	Progressively less saturated shading or color.
□	8	White; covers other graphics.
▨ ▧ ▩ ▤ ▦ ▨ ▨	9–15	Different types of hatching.

Initial value: 0.

font defines the font type and can have the following values:

Times	Times.
Arial	Arial.
Courier	Courier.
Symbol	Symbol.

Initial value: **Arial**.

linewidth defines the current linewidth in points (1 pt = 0.353 mm).

Initial value: 1.

marksize defines the mark size in points (1 pt = 0.353 mm). If the mark is a circle, the mark size corresponds to the diameter. For other types of marks, similar conventions apply.

Initial value: 6.

mode defines the input mode to line and area drawing commands and can have the following values:

normal	Coordinates are specified explicitly on the command line.

list Coordinates are taken from vector expressions, and the corresponding command is applied to all points in the list.

The input mode is automatically set to **list** when a graphics file with list data is read.

Initial value: **normal**.

rotate

defines the direction in which text is written and can have the following values:

off Text is written horizontally, from left to right.

on Text is written vertically, from bottom to top.

Initial value: **off**.

textsize

defines the font size in points (1 pt = 0.353 mm).

Initial value: 12.

weight

defines a font property with the possible values:

regular Regular; not bold.

bold Bold.

The Symbol font is only available as **regular**.

Initial value: **regular**.

x0, y0, x1, y1

define the positions of the two reference points (x_0, y_0) and (x_1, y_1) in the standard coordinate system. The standard coordinate system has its origin in the center of an A4 sheet and uses points (1 pt = 0.353 mm) to measure distances in both dimensions. The *x*-axis points to the right, and the *y*-axis points up.

Initial values: $x_0 = -250$, $y_0 = -375$, $x_1 = 250$, $y_1 = 375$.

X0, Y0, X1, Y1

define the positions of the two reference points (X_0, Y_0) and (X_1, Y_1) in the user coordinate system. All positions and distances are measured in the user coordinate system except for linewidth, text size, mark size, and dash patterns, which are always specified in points. These plot parameters are changed implicitly by the **scale** command or if autoscaling is enabled. The values of these plot parameters are available in INCLAN as intrinsic functions: **plotx0**, **ploty0**, **plotx1** and **ploty1**.

Initial values: $X_0 = -250$, $Y_0 = -375$, $X_1 = 250$, $Y_1 = 375$.