Instruction guide for Logoview NT

Date of printing of manual :31/05/99 18.20.00
Version :219
Last modified:1/10/98 19.02.00

CONTENTS

THE DATA

VARIABLES

6 different types of variable can be used for Logoview NT programming: BYTE, WORD, TRIGGER, REAL, DWORD and STRING.

TYPE	PREFIX	NUMBER OF BITS	RANGE
TRIGGER	?	1	0 - 1
BYTE	!	8	0 .. 255
WORD	#	16	-32768 .. 32767
REAL	@	32	+/-1E(+/-38)
DWORD	:	32	-2147483648 .. 2147483647
STRING	$	0	0-511 bytes

All the Logoview NT variables can be identified by a mnemonic name comprising up to 10 alphanumeric characters or by the variable's address.
The names of addresses of the variables must be preceded by a prefix that specifies the type. If the type is not specified the type that is active at that moment will be used as the default value Logoview NT will automatically insert the right prefix.
The following are valid variable names:

#test, ?trig, !0, #10, @f1, $20, ?1023, $STR

The TRIGGER variables are the individual BITS packaged inside the words. Logoview NT automatically masks them so that they appear as single variables to the programmer. In all cases however, expression calculation is converted into DWORD variables. The BYTE and WORD variables are also converted into DWORD. The BYTES are simply converted by zeroing the most significant part. The WORDS are converted by expanding the sign. To change a WORD into a DWORD without expanding the sign use the LOWORD function().

For example:

```
ASSIGN#A=-1
ASSIGN:A=#A
ASSIGN:B=LOWORD(#A)
```

:A contains -1 whilst B contains 65535

In the rest of this manual the term INT will be used to refer to any expression containing only BYTE, WORD, DWORD and TRIGGER variables. The is therefore calculated by using only whole numbers.

The STRING type variables enable sequences of characters or STRINGS to be stored. Within each STRING variable up to 511 characters of any type can be stored except for the character 0 (ZERO). This is used as a terminator.

VECTORS

The variable is therefore a vector and is indexed. The index is placed between square parentheses and must follow the name or offset of the variable.

#var [10*2+#v1]

This expression identifies the variable with an offset:

(VAR offset) + (10*2) + (contents of V1)

The expression between the square parentheses is always calculated using whole numbers: if it includes variables, constants or REAL functions these are first converted into INT and are then used in the calculation.

CONSTANTS

The whole constants can be expressed in both decimal and hexadecimal form; in the latter case the number must be preceded by the number from the Ox characters.

Hexadecimals: 0xffff, 0x12

Decimals : 65535, 18

The range of whole constants is the same as the range of DWORD variables.

Real constants can be expressed both in normal decimal notation and in exponential notation.

123.24, 0.123, 1E23, 1.234E-14, 5.6E-7

Both the real constants and the variables can have a maximum exponent of 38.

The string constants consist of a sequence of ASCII characters between two apexes. For example:

"ABCDEF", "GHI JKL"

The maximum length of a string constant is 255 characters.

EXPRESSIONS

The expressions are used whenever parameters are given to the Logoview NT instructions. An expression can be of two types only: STRING expression or NUMERIC expression.
The two types are incompatible and cannot be swapped over.
Some instructions require a specific variable. Normally, the variable request is of a specific type. In other words, if a numeric variable is required, it is not possible to use a string variable as a parameter or vice versa.

STRING EXPRESSION

A string expression has a very simple form because it can consist only of a string constant or of a string variable. The strings must not be altered in any way. To manipulate the strings, use the specific instructions (STR…).

NUMERIC EXPRESSION

A numeric expression may be whole or real, depending on the type of numeric values used and the type of result. If the result has to be of the REAL type, the entire expression must be calculated as being real and all the INT constants and variables are therefore converted before making the calculations. In the same way, if the expression contains REAL constants or variables but needs to produce an INT result it is calculated as REAL and the result is then converted into INT before it is used. On the other hand, if an expression has to return an INT result and contains only INT constants and variables it will be calculated only with whole numbers.

The expressions contained inside the square parentheses are exceptions to this rule. Such expressions must always produce an INT result and even if the expressions contain REAL variables they are calculated by whole numbers. In such cases the REAL variables and constants are changed into INT before the calculations are made.

Within the numeric expressions functions can also be used (see below) whose parameters follow the rule described above and whose results influence the calculation of the entire expression. If a function that returns a REAL number is used in an expression all the expression will be calculated as if it were REAL. On the other hand, if a function that returns an INT result, the INT result will be converted into a REAL result.

OPERATORS

The following operators can be used in the numeric expressions:

+	Addition
-	Subtraction
•	Multiplication
/	Division
%	Module
&	AND, binary
|	OR binary
^	XOR binary
~	NOT binary
=	Equality
<>	Inequality
<	Lesser than
>	Greater than
<=	Lesser than or equal to
>=	Greater than or equal to

The relational operators give the following results:

-1: relation has taken place
0: relation has not taken place

These are the operational priorities, in descending order:

1) - and ~ operational units
2) * /%&
3) +-|^
4) = < > <= >= <>

The operators & (AND) and **|** (OR) are binaries (bit to bit) and can be used for logic expressions. However, the relational expressions must be placed between brackets:

((v1) & (v2)) **|** (v3=0)

The result is TRUE if (v1 AND v2) is TRUE or if (v3=0)

False condition: numeric expression same as 0

True condition: numeric expression different from 0

FUNCTIONS and INSTRUCTIONS for programming events

We now offer a rapid review of the functions and instructions that are available in Logoview NT for programming events.

These types of values are used in the functions and prototypes instructions:

INT:	expression with whole result
REAL:	expression with real result
VAR:	variable
STR:	string

VARIABLES SYSTEM CONSTANTS

The set variables or set constants can be used in any expression just by specifying their names.

System data:

INT **DAY:**	Day of month(1-31)
INT **MONTH:**	Current month(1-12)
INT **YEAR:**	Current year(1990-...)
INT **HOUR:**	Current time(0-23)
INT **MINUTES:**	Minutes (0-59)
INT **SECONDS:**	Seconds (0-59)
INT **DAY_W:**	Day of the week (0=SUNDAY 6=SATURDAY)

Complete date of last record of a read file (with a READ REC):

INT **R_DAY:**	Day of month (1-31)
INT **R_MONTH:**	Current month (1-12)
INT **R_YEAR:**	Current year (1990-...)
INT **R_HOUR:**	Current time (0-23)
INT **R_MINUTES:**	Minutes (0-59)
INT **R_SECONDS:**	Seconds (0-59)

Other uses of variables:

INT **RAND:**	returns a random number between 0 and 32767
REAL **PI:**	3 141592

FUNCTIONS

- **INT ABS (INT)**

Calculates absolute value of whole parameter

- **INT ACTBOX**

Returns number of active or set BOX with a SET BOX or number from which the event was triggered.

- **REAL ACOS (REAL)**

Calculates the cosine arc of the parameter in the range - PI/2...PI/2p; the parameter must be between -1 and 1, otherwise an error is generated and the program stops.

- **INT ASCII (STR st, INT chr)**

Returns the ASCII code of the index character ’chr’ in the ’st’ string. If ’chr’ is greater or the same as the length of the string the ASCII returns 0 (value not permitted as a string character).

- **REAL ASIN (REAL)**

Calculates the sine arc of the parameter in the range - PI/2...PI/2; the parameter must be between -1 and 1, otherwise an error is generated and the program stops.

- **REAL ATAN (REAL)**

Calculates the tangent arc of the parameter in the range - PI/2...PI/2.

- **REAL ATOF (STR)**

Converts a string numeric value into a REAL numeric value. For example: ATOF("25.2") returns 25.2

- **INT ATOI (STR)**

Converts a numeric value into a whole string. For example, ATOI("25") returns 25.

- **###REAL CEIL (REAL)**

Returns the smallest whole number that is greater or the same as the specified parameter.

- **REAL COS (REAL)**

Calculates the angle cosine, expressed in radiants that has given as a parameter.

- **REAL COSH (REAL)**

Calculates the hyperbolic cosine of the parameter: if the calculated value is too great, the function generates an error and stops the program.

- **### REAL EXP (REAL)**

Calculates the parameter’s exponential value. If the calculated value is too great an error is generated and the program stops.

- **### REAL FABS (REAL)**

Calculates the absolute value of the REAL parameter.

- **###REAL FLOOR (REAL)**

Returns the largest whole number that is the same as or less than the specified parameter.

- **### INT F_EXIST (STR fname)**

Checks the existence of the ’fname’ file. If the name does not contain any path (e.g. ’test.arc’) the path is used in which the application is located.
Returns:
0 if the file does NOT exist.
1 if the file exists

- **### INT HIWORD (INT dw)**

Returns the 16 most significant bits of the DWORD dw.

- **### REAL INT (REAL)**

Returns the parameter value rounded off to the nearest whole number. For example, if 13.2 is given as a value, 13 is returned. If on the other hand the parameter is 13.8, the result will be 14.

- **### REAL LOG (REAL)**

Calculates the natural logarithm of the parameter; if the parameter is negative an error is generated and the program stops.

- **### REAL LOG10 (REAL)**

Calculates the logarithm in the parameter’s base 10; if the parameter is negative an error is generated and the program stops.

- **INT LOWORD (INT dw)**

Returns the 16 least significant bits of DWORD dw.

- **INT NEWLINE**

Returns the code corresponding to the ‘newline’

- **INT TAB**

Returns the code corresponding to the tabulator.

- **REAL POW (REAL x, REAL y)**

Returns xy. If x is 0 and y is different from 0 the function returns 1. On the other hand, if x is zero and y is less than 0 or if x and y are both 0 or if x is negative and y is not an entire number the function generates an error and the program stops. For example, to calculate 23, it is sufficient to use the function: pow(2.3).

- **### INT RGB (red INT, green INT, blue INT)**

Enables a color to be specified from the three basic components: red, green and blue. This function can be used for all the instructions that have a color as a parameter.
The color is always full; Logoview NT automatically searches inside the current palette and returns the color that is nearest to the one specified by the three parameters.

- **INT RGB_A (red INT, green INT, blue INT)**

Enables a color to be specified from the three basic components: red, green and blue. This function can be used for all the instructions that have a color as a parameter.
The color is always the one specified by the three parameters. If it is not available in the current palette it will be simulated by Logoview NT by means of a screen.

- **### INT ROL (INT v, INT n)**

Shift left
v= value to be shifted
n= number of positions

- **### INT ROR (INT v, INT n)**

Shift right
v= value to be shifted
n= number of positions

- **INT SAR (INT v, INT n)**

Arithmetical shift to RIGHT (e.g. -2=FFFE -->> FFFF)
v= value to be shifted
n= number of positions

- **### INT SHL (INT v, INT n)**

Shift left
v= value to be shifted
n= number of positions

- **### INT SHR (INT v, INT n)**

Shift right (Es. -2=FFFE -->> 7FFF)
v= value to be shifted
n= number of positions

- **REAL SIN (REAL)**

Calculates angle sine expressed in radiants given as a parameter.

- **### REAL SINH (REAL)**

Calculates parameter’s hyperbolic sine; if the sine is too great it generates an error and the program stops.

- **### REAL SQRT (REAL)**

Calculates the parameter’s square root; if the parameter is negative an error is generated and the program stops.

- **### INT STRBRK (STR st,STR set)**

Seeks the first occurrence of a character in the ’set’ inside the ’st’ string.
Returns:
the index of the found character
-1 : character not found

- **### INT STRCHR (STR s,INT c)**

Seeks the first occurrence of the character ’C’ inside string ’s’.
Returns:
the index of the found character
-1 : character not found

- **### INT STRCMP (STR s1,STR s2,INT nc)**

Compares strings ’s1’ and ’s2’. Each ’s1’ character is compared with the corresponding ’s2’ character to the end of one of the two strings or until ’nc’ characters.
Returns :

-1: s1<s2
0: s1=s2
+1: s1>s2

- **### INT STRCMPI (STR s1,STR s2,INT nc)**

Compares strings ’s1’ and ’s2’ in case-insensitive mode (i.e. no distinction is made between "AA" and "aa"). Each ’s1’ character is compared with the corresponding character in ’s2’ to the end of one of the two strings or until ’nc’ characters.
Returns:
-1: s1<s2
0: s1=s2
+1: s1>s2

- **INT STRCSPN (STR st,STR set)**

Calculates the length of the initial segment of the ’st’ string, which is made up of the characters that are NOT included in ’set’.

- **### INT STRLEN (STR s)**

Returns the length ’s’.

- **### INT STRRCHR (STR s,INT c)**

Seeks the last occurrence of character ’C’ inside string ’s’.
Returns the index of the found character
-1: Character not found

- **### INT STRSPN (STR st,STR set)**

Calculates the length of the initial segment of the ’st’ string, which is made up of the characters that are included in ’set’.

- **### INT STRSTR (STR s1,STR subs)**

Seeks the first occurrence of the string ’subs’ inside ’s1’
Returns the index or -1 if not found

- **### REAL TAN (REAL)**

Calculates the tangent of the angle, expressed in radiants, given as a parameter. If the corner tends to be a multiple of P1/2 (the tangent tends towards infinity) the function generates an error that stops the program .

- **### REAL TANH (REAL)**

Calculates the parameter’s hyperbolic tangent.

INSTRUCTIONS

Each instruction is put forward with its own syntax and with a short description of how it works. If the function cannot be immediately be understood, a short example is given to illustrate the operating principle.

Many of the functions that are current in Logoview 32 have been removed from Logoview NT. This is because the new 32 bit Windows operating systems are very different from MS-DOS. We have tried not to decrease available functions: in fact some of the instructions that are not current can be simulated by others. For example, the functions

SET DATE and SET TIME are no longer available in Logoview NT. The date and time can be changed by calling up the event on the control panel by means of this instruction:

EXEC “control timedate.cpl”

*   ‘COMMENT’

♦###**Syntax**

*   STR **comment**

♦**Parameters**

comment: Optional string

♦**Description**

Enables a comment line to be inserted inside the events.
Select this instruction to insert any sequence of characters that will be a more or less brief description of the individual activities within an event.

♦**Example**

Event 11:

**   This comment is an example of on-line documentation*
**   of the functions that take place inside an event:*
LET #VIS=1

- *The value 1 RETURN is assigned to the WORD type VIS variable.*
-

ALARMS COUNT

♦###**Syntax**

ALARMS COUNT VAR INT **num**, INT **class**, INT **area**

♦**Parameters**

num:	Alarm number
class:	Class index
area:	Area index

♦**Description**

Returns in **{num}** numbers of alarms current in an area and class date. To consider all the classes, select **{class}** = -1. To consider all the areas, select **{area}** = -1.

♦**Return value**

Return value in **{num}.**

ALARMS EDIT

♦ ######**Syntax**

ALARMS EDIT VAR INT **ris**, INT **class**, INT **area,** INT **flg**

♦ **Parameters**

ris: Result

0 = Ok

-1 = Alarms not defined

class: Indicates class

area: Indicates area

flg: Type of disabled functions

1	=	Editing Messages
2	=	Editing Thresholds
4	=	Enable / Disable individual alarm
8	=	Disable / Enable all alarms
16	=	Change Area
32	=	Change Class
64	=	Modify Class

♦ **Description**

Opens the alarm modification window in the specified area and class. To view all the classes, just specify **{class}** = -1. To consider all the areas, specify **{area}** = -1.
This function enables the operator to modify alarm settings by enabling them or disabling them. The new settings take priority over those set during the development phase. They are kept in files that are separate from the ***.gme*** (that contains the developed application) in particular:

File with extension***.cls*** contains class modifications.
File with extension***.sgl*** contains modifications to analog thresholds.
File with extension***.sts*** contains status modifications (enabled/disabled).
File with extension***. sms*** contains message modifications.

To delete the new settings, delete the corresponding files. The step restores the configurations set during the development phase.

♦ **Return value**

Return value in **{ris}.**

ALARMS LOG

♦###**Syntax**

ALARMS LOG INT **status**, INT **type**

♦**Parameters**

status: Activate/Deactivate Log

0 = Deactivate
1 = Activate

type: Type of Log to activate or deactivate

0 = Disable/Enable alarm
1 = Modify thresholds
2 = Alarm recognition
3 = Activate alarm
4 = Reset alarm
5 = Modify dead range
6 = Modify class
7 = Close/reopen history alarms DB
Status=0: Close
Status=1: Reopen

♦**Description**

Activates or deactivates a Log type both on history file and printer.

If **{type=7}** this function will the enable the alarms database to be opened/closed in order to carry out maintenance operations.

ALM GETANALOG

♦ ###**Syntax**

ALM GETANALOG VAR **alm**, REAL VAR **ll,** REAL VAR **l,** REAL VAR **h,** REAL VAR **hh,** REAL VAR **bd,** INT VAR **msk,**

♦ **Parameters**

alm: Alarm variable or threshold trigger
ll: Low low threshold value
l: Low threshold value
h: High threshold value
hh: High high threshold value
bd: Dead range value
msk: Active thresholds
Bit 0 = threshold ll (1 = activate 0 = deactivate)
Bit 1 = threshold l (1 = activate 0 = deactivate)
Bit 2 = threshold h (1 = activate 0 = deactivate)
Bit 3 = threshold hh (1 = activate 0 = deactivate)

♦ **Description**

This instruction enables the configuration of an analog alarm to be read. .The variable **{msk}** contains the status of the thresholds. If **{msk}** = -1 the variable is not an analog alarm; this means that an incorrect variable **{alm}** has been given.

ALM GETSTATUS

♦ ###**Syntax**

ALM GETSTATUS VAR **alm**, VAR **status**

♦ **Parameters**

alm: Alarm or threshold trigger variable
status: Threshold or alarm status (-1 incorrect variable)

♦ **Description**

This instruction returns the status of a digital alarm, an analog alarm or of an analog threshold.

♦ **Return value**

Return value in **{status} :**

0	=	No alarm
1	=	Acquire current alarm
2	=	Current alarm recognized
3	=	Alarm disabled
4	=	Alarm reset but not recognized

ALM SAVECFG

♦ **Syntax**

ALM SAVECFG INT **type**

♦ **Parameters**

type: Type of configuration to save

0: Not used, maintained for compatibility with Logoview 32
1: Alarm status (file .STS)
2: Values of thresholds and dead ranges (file .SGL)
3: Messages (file .SMS)
4: Configuration of classes (file .CLS)

♦ **Description**

Save on file the modifications to the configuration of alarms made during runtime. This function is normally called up automatically when the alarm editing window is closed.

ALM SETANALOG

♦###**Syntax**

ALM SETANALOG VAR **alm**, VAR REAL **ll,** VAR REAL **l,** VAR REAL **h,** VAR REAL **hh,** VAR REAL **bd**

♦**Parameters**

alm: Alarm or threshold trigger variable
ll: New low low threshold value
l: New low threshold value
h: New high threshold value
hh: New high high threshold value
bd: New dead range value

♦**Description**

This instruction modifies the configuration of the thresholds and the dead range of an analog alarm in accordance with the specified parameters.

ALM SETSTATUS

♦ ###**Syntax**

ALM SETSTATUS VAR **alm**, VAR **status**

♦ **Parameters**

alm: Alarm or threshold trigger variable

status: New alarm status

0	=	No alarm
1	=	Acquire current alarm
2	=	Current alarm recognized
3	=	Alarm disabled
4	=	Alarm reset but not recognized

♦ **Description**

This instruction returns the status of a digital alarm, an analog alarm or of an analog threshold according to **{status}**.

This function can be used both to recognize an event alarm **{status}** = 2 and to disable it. **{status}** = 3. The transition from old status to new status must obviously be conceptually correct: for example, status cannot be changed from 0 to 4.

ALM SIZEDISP

♦ **Syntax**

ALM SIZEDISP INT **height**, INT **cmd**

♦ **Parameters**

height: height in pixel

cmd: Command and flags

Commands:

0: Initializes default height

1: Maximizes display

2: Hides display

3: Restores default dimensions

4: Restores but maintains current dimensions

Flags:

16: Stops resize by user

♦ **Description**

Modifies the dimensions of the display of current alarms. The parameter **{height}** is used only with command 0. The parameter **{cmd}** contains a command (between 0 and 4) and a flag that modifies the command action. For example, to restore default dimensions and prevent them being modified by the user specify **cmd**=19 (3+16).

BOX

♦ **Syntax**

BOX INT **x1**, INT **y1**, INT **x2**, INT **y2**, INT **col**

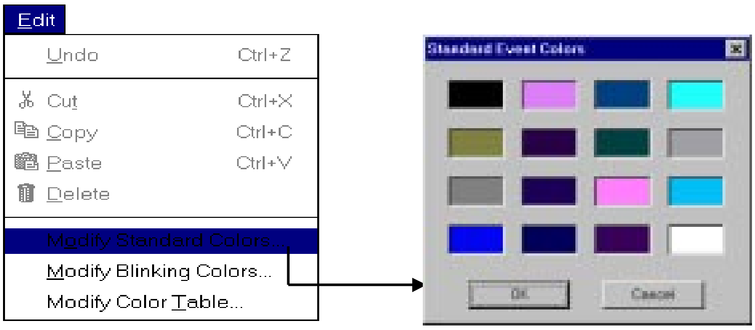
♦ **Parameters**

x1,y1,x2,y2:	Rectangular coordinates
col:	Rectangular color

♦ **Description**

It traces a colored rectangle **{col}** on the work video display. The angle coordinates are in the top left hand corner of the monitor and are expressed in PIXEL. The color number is the same as the one configured in the table of the *Standard event colors.*

Macros **RGB** and **RGB_A** can be used as parameter**{col}**.



BUILD PATH

♦ **Syntax**

BUILD PATH INT **type**, STR **fname**, VAR STR **path**

♦ **Parameters**

type:	Type of path to be constructed:
	0: Path in which the current application is stored
	1: Path from which Logoview is run
	2: Path for temporary files as configured by system
	3: Path in which Windows is located
	4: Path of first CD.ROM installed
fname:	Name of file without path
path:	Name of file with path

♦ **Description**

This instruction constructs a file name with the path of set directories. The parameter **{type}** establishes the set path to be used for constructing the name; the parameter **{fname}** is the file name to which the path is added. It may also consist of an empty string: in this case only the path finishing with \ will be copied.

♦ **Example**

Suppose that the application has been installed in C:\LogoNt\MyApp, Logoview is installed in C:\LogoNT, Windows is installed in D:\WinNT and the system variable TEMP is the same as D:\TEMP:

BUILD PATH 0,"myfile.txt",$L0
** Returns in $L0: C:\LogoBT\MyApp\myfile.txt*
BUILD PATH 1,"myfile.txt",$L0
* Returns in $L0: C:\LogoBT\myfile.txt
BUILD PATH 2,"myfile.txt",$L0
* Returns in $L0: D:\TEMP\myfile.txt
BUILD PATH 3,"myfile.txt",$L0
* Returns in $L0: D:\WinNT\myfile.txt

CALL

♦ **Syntax**

CALL INT **e**, VAR **ris**, [, PAR **par1**, ... , PAR **par15**]

♦ **Parameters**

e:	Event number
ris:	Result
par1:	Variable or value given as a parameter value
:	
par15:	Variable or value given as a parameter value

♦ **Return value**

{ris} contains the result that the called-up event has returned in the instruction **RETURN**.

♦ **Description**

§>Carries out a subroutine from event **{e}**. This instruction is the same as **SUB** but enables the result **{ris}** returned by the called-up event to be used. Also in this case the event number **{e}** can be replaced by a variable that contains the number. In this case the value of the variable must be checked with which the event number is given that is to be run because if the events are outside the defined range of events running the event could cause the system to crash.

It is also possible to use the name of the event to run it. When an event is run with the **CALL** function the called-up event finishes with the instruction **RETURN**. At the end of this event the program flow returns to the following instruction of the calling event. Orphan **CALLS** must not be made i.e. calls must not be made without the corresponding **RETURN,** otherwise there is a risk of causing a system emergency because of lack of space on the stack. The maximum number of nested **SUBS** or **CALLS** is 50.

♦ **Example**

Suppose that we wish to implement the function that calculates the area of a rectangle and to use it in several points of our application. One very simple solution is to write the code that carries out the required operation in a single event (e.g. event 10) and to use the **CALL** in all the points in which one wishes to calculate the area of the rectangle.

```
Event 10:
*loc is a local variable, p1 and p2 are the function parameters.

LET #loc = #p1 * #p2
RETURN #loc

Event no:
* the WORD variables 'side1' and 'side2' contain the two sides of the rectangle
....
CALL 10, #area, #lato1, #lato2
...
* the WORD variable of the time area name contains the area of the rectangle
```

CHANGE COL

♦ **Syntax**

CHANGE COL INT **x1**, INT **y1**, INT **x2**, INT **y2**, INT **col1**, INT **col2**

♦ **Parameters**

x1,y1,x2,y2:	Rectangle coordinates
col1:	Previous color
col2:	New color

♦ **Description**

Within the rectangle defined by the coordinates given as a parameter color **{col1}** changes into **{col2}** on the work video screen. The coordinates in the top left-hand of the monitor are expressed in PIXEL. The colors are defined in the *Standard event colors* table.

Macros **RGB** and **RGB_A** can be used as parameter**{col}**.

CH SETDIM

♦ **Syntax**

CH SETDIM STR **id**, INT **flg**, INT **num**

♦ **Parameters**

id:	Object identifier
flg:	0 = lines 1 = columns
num:	New number of lines or columns

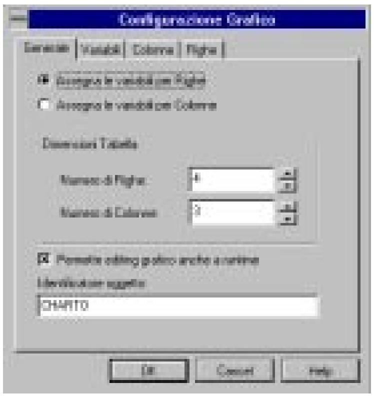
♦ **Description**

Modifies the number of lines or columns of the graph identified by **{id}**.

The settings modify the development settings but they are not saved and are therefore lost during the subsequent implementation of the application.

This instruction can be carried out only by events that have an assigned frame. This may be implemented by, for example, buttons, animations and interactive fields. If there is no assigned frame (as in the case of an event run by timer) the function returns an error. This behavior is identical in all **CH** faults.

The parameter **{id}** is indicated in the graphic configuration window, which can be accessed by pressing . This refers to a box positioned on the frame that has carried out an event containing the instruction.



● Assegna le variabili per Righe

○ Assegna le variabili per Colonne

Dimensioni Tabella

Numero di Righe: 4

Numero di Colonne: 3

☒ Permette editing grafico anche a runtime

Identificazione oggetto: CHART0

CH SETLABEL

♦ **Syntax**

CH SETLABEL STR **id**, INT **flg**, INT **idx**, STR **label**

♦ **Parameters**

id:	Graph identifier
flg:	0 = lines 1 = columns
idx:	Line or column index
label :	New label

♦ **Description**

Modifies the label of the element of a line or a column of the graph identified by **{id}**.

This instruction can be carried out only by events that have an assigned frame that is carried out by buttons, animations and interactive fields. If there is no assigned frame (as in the case of an event run by Timer) the function returns an error. This behavior is identical for all the **CH** functions.

CH SETVAR

♦ **Syntax**

CH SETVAR STR **id**, INT **idx**, VAR **vr**

♦ **Parameters**

id: Graph identifier
idx: Line or column index
vr: Variable

♦ **Description**

Modifies the group of variables used to update the graph **{id}**. **{idx}**. This is the number of the line and the column, which depends on the configu5ration of the graph.

This instruction can be carried out only by events that have an assigned frame that is carried out by buttons, animations and interactive fields. If there is no assigned frame (as in the case of an event run by Timer) the function returns an error. This behavior is identical for all the **CH** functions.

CH UPDATE

♦ **Syntax**

CH UPDATE STR **id**

♦ **Parameters**

id: Graph identifier

♦ **Description**

This updates the graph **{id}** according to variables content. This instruction must be used to enter the new variable values in the graph. The graph is plotted automatically once only: when the frame is loaded. This function must be used for all subsequent updates.

This instruction can be carried out only by events that have an assigned frame that uses buttons, animations and interactive fields. If there is no assigned frame (as in the case of an event run by Timer) the function returns an error. This behavior is identical for all the **CH** functions.

CHECK DATE

♦ **Syntax**

CHECK DATE INT **day**, INT **month**, INT **year**, VAR **err**

♦ **Parameters**

day: Day of month to be checked
month: Month to be checked
year: Year to be checked
err: Error code (0=Ok)

♦ **Description**

This instruction checks that the date given as a parameter is correct.

♦ **Return value**

Code of error in **{err}**:

0	=	Ok
1	=	Incorrect day
2	=	Incorrect month
4	=	Incorrect year

CLOSE FILE

♦ **Syntax**

CLOSE FILE INT **arc**

♦ **Parameters**

arc: File number

♦ **Description**

Close the file assigned to the file and then stops the reading and writing operations on the file in question.
As for the other functions that refer to the file the file must have first been defined and opened.
The file is one of the 20 defined in the section *History files* of the window *Format Configuration.* This can be reached by the button in the main tools bar or by using the relevant menu command.
To describe the file characteristics and to create the record, see the description in the chapter on the files, in the Formats section of the Logoview NT manual.

If the file has not been created or has not been opened an error message is generated by Runtime.

CLOSE SCREEN

♦ **Syntax**

CLOSE SCREEN [STR **scrname]**

♦ **Parameters**

scrname: name of screen to be closed

♦ **Description**

If it is called up without parameters it closes the frame from which the instruction came. If it is without parameters it can be called only by events assigned to graphic elements positioned on the frames or titanium modules. It has no effect on events created, for example, by timers, which therefore have no ’assigned’ frame. If the parameter is specified the instruction closes only the specified screen, if it is open. In this case it can only be called up by events that do not have an assigned frame. Warning: the parameter cannot be the name of a titanium frame or of a trend - in this form the instruction closes only the actual frames.

COMMAND PLC

♦ **Syntax**

COMMAND PLC STR **plc**, STR **node**, INT **cmd**, VAR INT **ris** [,INT **flg**]

♦ **Parameters**

plc: Name of driver station from which to make the reading
node: Node of the network hosting the driver station
cmd: Command to send to the PLC driver:
= **0** Start
= **1** Stop
= **2** Pause
ris: Operation result
flg : = **1** return immediately if other messages are imminent

♦ **Description**

Sends the command **{cmd}** to the PLC driver.

♦ **Return value**

Result of the operation in the parameter **{ris} :**

0 = Ok
otherwise = Error

CONVERT ESC

♦ **Syntax**

CONVERT ESC STR **text**, INT **op**

♦ **Parameters**

text: String to be converted

op: Type of conversion

0 = convert escape sequences into these characters

1 = convert characters into escape sequences

♦ **Description**

Depending on the parameter**{op}** this instruction converts sequences \n \t \\ into the corresponding characters (op = 1) and vice versa (op = 0). This instruction therefore enables the characters \n \t \ to be used in the string variables by converting them when they are used. For example, in an **MESSAGEDLG** in which new lines need to be made. The opposite procedure may be useful if the string is loaded by a file.

COPY VAR

♦ **Syntax**

COPY VAR VAR **source**, VAR **dest**, INT **nvar**, [INT **flg**]

♦ **Parameters**

source: Name, variable source
dest: Name, variable destination
nvar: Number of variables to be copied
flg: if absent or if = **0** animations are not updated
(tables, trends and interactive fields)
= **1**, updates tables, trends and interactive fields
(this nevertheless slows down implementation) in this case
copying will be slower.

♦ **Description**

This instruction enables a certain number of variables, even of differing types, to be copied onto other variables.
The variables can be confined to individual events or be general for the entire application.

♦ **Example**

Copies 55 variables from the variable REAL "var_real" onto 55 variables starting with variable "local_word"; the decimals will be lost:

COPY VAR @var_real,#local_word,55

CREATE FILE

♦ **Syntax**

CREATE FILE STR **name**, INT **arc**, VAR **err [**,INT **flag]**

♦ **Parameters**

name:	File name		
arc:	File number		
err:	Error code		
flag :	**0**	=	Create Logoview NT file (default)
	1	=	Create file compatible with LW32

♦ **Description**

The instruction creates a historical file. If the file already exists, the function **CREATE FILE** destroys it; if the file does not exist, a new one is created. Apart from creating a new file, the instruction also opens the file for reading and writing.

A Logoview NT file enables up to a maximum of 64 Kb per record to be stored. In other words:

65535 BYTE/TRIGGER o **32767** WORD o **16383** DWORD/REAL

On the other hand, there is no limit to the maximum offset of variables that can be stored. In a Logoview 32 file, variables with an offset that is greater than **65535** cannot be stored**.**

To share a Logoview NT file (compatible with Logoview 32 **{flg}**=1) with Logoview 32 insert the following line into the AUTOEXEC file of the PC with Logoview 32:

SET TZ = EST-1DST-2,M3.5.0,M10.5.0

This ensures that the Logoview NT and Logoview 32 time schedules have the same TimeZone configuration.

♦ **Return value**

Errors on history files in the variable **{err} :**

0=	No error	
-1	=	Error during opening of file
-2	=	Error during creation of file
-3	=	Error in format number
-4	=	Writing error (disk full or other)
-5	=	Reading error
-6	=	Memory insufficient for data on file fields
-7	=	Variable does not exist
-8	=	Header incorrect
-9	=	Version number incorrect
-10	=	READ_ONLY file opened

CURSOR

♦###**Syntax**

CURSOR INT **v** [, INT **x,** INT **y**]

♦**Parameters**

v: Operation code
0, 1 = Not used
2 = Disable keys assigned to interactive buttons
3 = Re-enable keys assigned to interactive buttons
4 = Place cursor on absolute coordinates **{x,y}**
5 = Place cursor in reference box **{x}**

x,y: Absolute coordinates

♦**Description**

This instruction makes it possible to enable or disable keys assigned to interactive buttons. It also enables the cursor to be placed in any position on the frame by means of absolute coordinates or by using a reference box.

ATTACH DB

♦ **Syntax**

ATTACH DB INT **hnd**, VAR **err**

♦ **Parameters**

hnd: handle DB

err: Error code

♦ **Description**

Adds an empty record to the **DB**. The added record becomes the current record.

♦ **Return value:**

Error codes returned as **{err}**:

0: Ok

<0: Error, see DB errors table

DB BOTTOM

♦ **Syntax**

DB BOTTOM INT **hnd**, INT **err**

♦ **Parameters**

hnd:	DB handle
err:	Error code

♦ **Description**

Moves the reading/writing pointer in the DB to the last record of the file using the current TAG. The last record is not the last record to have been added to the DB but the last record in the sorting method of the current TAG.

♦ **Return value**

Error codes returned as **{err}** :

0=	Ok	
3:	=	End of File
<0:	=	Error (see DB errors table)

DB CLOSE

♦**Syntax**

DB CLOSE INT **hnd**

♦**Parameters**

hnd: DB handle

♦**Description**

Close the database specified by the handle **{hnd}** and all the assigned index files. The handle must be a value returned by CREATE DB or OPEN DB.

DB CLOSEIDX

♦ Syntax

DB CLOSEIDX INT **hnd**

♦ Parameters

hnd: Handle index file

♦ Description

Close the index file specified by the handle **{hnd}**. The handle must be a value returned by the CREAIDX DB or OPENIDX DB**.**

DB CREATE

♦ **Syntax**

DB CREATE STR **fnm**, INT **des**, VAR **hnd**, VAR **err** [,INT **flg**]

♦ **Parameters**

fnm: Name of database to be used
des: Description to be used
hnd: Handle of created DB. -1 if incorrect
err: Error code
flg: Bit mask :

1 = Enables files to be shared
2 = No duplicate keys
4 = Inverted sorting

♦ **Description**

Creates a new database file (.DBF) and the assigned index file (.CDX). After creating and opening the file, the first TAG of the index file and the current record pointer is positioned on the first record. The **DB** is opened EXCLUSIVELY unless the parameter **{flg}**is set at 1.

The created file is assigned a handle. This must be used for all further references to the created **DB.** This handle is returned to **{hnd}** and must be used in all the subsequent instructions in which reference is made to the created DB.

DB CREATE requires the parameter **{des}** to contain the index of the record format used. This record format must have been defined in the "RECORD CONFIGURATION" window.

♦ **Return value**

If **{hnd}** =-1, **{err}**contains an error code. See table with DB error codes.

DB CREATEIDX

♦ **Syntax**

DB CREATEIDX STR **name**, INT **db**,VAR **hnd**,VAR **err**, STR **tag**, STR **expr**, STR **filter** [,INT **flg**]

♦ **Parameters**

name: Name of index file
db: Database handle
hnd: Handle of created index file. -1 if incorrect
err: Error code
tag: Name of TAG (max.. 10 characters)
expr: Sorting expression (max. 240 characters)
filter: Expression for filter (max. 240 characters)
flg: Bit mask :

1 = Enables files to be shared
2 = No duplicate keys
4 = Inverted sorting

♦ **Description**

Creates a new index file **{name}** for an already open **DB**. The parameter **{db}** specifies the **{handle}** of the database from which another index is to be created. The parameter **{tag}** must be a string of alphanumeric characters used to identify the TAG with the instruction **DB TAGSELECT**. The parameters **{expr}** and **{filter}** must be two dBase expressions (for the syntax of dBase expressions see chapter 9: “The Formats” of the Logoview NT PRIMER. **{expr}** determines the sorting method of the new tag that has been created; **{filter}** enables the keys to be identified that should be fitted into **{tag}**. The parameter **{flg}** enables the instruction's default behavior to be modified. The index file is created in the exclusive mode by default, ensures that there are duplicate keys and sorting is in ascending order.

Specify **{flg=1}** to share the index file. **{flg=2}** inhibit the duplicate keys. In this case, each time that a key is added to the Logoview tag check that the key is not already current. Specify **{flg=4}** for tag sorting in descending order. The **{handle}** that is returned to **{hnd}** must be used for each subsequent reference to the index file (for example in a **DB CLOSEIDX**).

♦ **Return value**

If **{hnd}** =-1, **{err}** contains an error code. See table with DB error codes.

DB DELETE

♦ **Syntax**

DB DELETE INT **hnd**

♦ **Parameters**

hnd: DB handle

♦ **Description**

The current record is marked as **{deleted}**. In fact, however, the record is not removed from the file until a DB PACK is carried out.

DB FIELDINFO

♦Syntax

DB FIELDINFO INT **hnd**, VAR **err**, STRING **name**, VAR **type**, VAR **len**, VAR **dec**

♦Parameters

hnd:	DB handle
err:	Error code
name:	Field name
type:	Type ’C’=CHAR ’D’=DATA ’N’=NUMERIC
len:	Length in characters
dec:	Number of decimals

♦Description

Returns information about a certain DB field. **{hnd}** is the DB handle to which the field **{name}** belongs. A code that returns field type is returned to **{type}**:

’C’=67:	CHARACTER
’D’=68:	DATE
’N’=78:	NUMERIC FIELD
’F’=70:	FLOATING POINT FIELD
’L’=76:	LOGIC
’M’=77:	MEMO

{len} returns the length of the field in characters whilst **{dec}** is the number of decimals (only NUMERIC and FLOATING POINT FIELDS).

♦Return value

Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB FIELDNAME

♦ **Syntax**

DB FIELDNAME INT **hnd**, VAR **err**, INT **nfld**, VAR STRING **name**

♦ **Parameters**

hnd: DB handle
err: Error code
nfld: Field number (1<=nfld<=Field number of record)
name: Field name

♦ **Description**

Returns index field name **{nfld}** of the DB whose handle is specified in the parameter **{hnd}**.

♦ **Return value**

Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB FLUSH

♦ **Syntax**

DB FLUSH INT **hnd**, VAR **err**

♦ **Parameters**

hnd:	DB handle
err:	Error code

♦ **Description**

Forces writing onto disk of the records buffered in the memory. Normally, the modified records are written when a DB SKIP or DB GO occurs. However, in order to be sure that the current record is written onto the disk at a given moment, use **DB FLUSH**.

♦ **Return value**

Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB GETFIELD

♦ **Syntax**

DB GETFIELD INT **hnd**, VAR **err**, STR **field**, VAR **val** [,INT **flg**]

♦ **Parameters**

hnd:	DB handle
err:	Error code
field:	Field name to be read
val:	Field contents
flg:	used only for DATA fields; = **1** indicates that the date is returned in the format "CCYYMMDD"

♦ **Description**

Reads **{field}** from the current record and places it in the variable **{val}**. The parameter **{flg}** is used only with the DATE fields. If the date field is 1 it is returned in the format "CCYYMMDD". If **{flg}** is zero or absent it is formatted in accordance with the Windows configuration. This function is also available even if the record description index was not supplied when the DB was opened or created.

♦ **Return value**

Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB GETREC

♦ **Syntax**

DB GETREC INT **hnd**, VAR **err**

♦ **Parameters**

hnd: DB handle
err: Error code

♦ **Description**

Reads the current record from the database **{hnd}** and places it in the variables that were configured in the Database section of the formats configuration window. This function is available only even if the record description index was supplied when the DB was opened or created.

♦ **Return value**

Return value
Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB GETSTAT

♦ **Syntax**

DB GETSTAT INT **hnd**, INT **type**, VAR **data**

♦ **Parameters**

hnd: DB handle

type: Type of information required

0 = Number of current record (>=1)

1 = Number of record in DBF file

2 = EOF or BOF (0: Ok, 1=EOF, 2=BOF)

3 = Number of fields per record

data: Information required (<0 if error)

♦ **Description**

Requests information on Database with handle **{hnd}**. The information required must be specified by the parameter **{type}**.

{type} =0 : returns current record number. 1<= record number in DBF file

{type}=1 : returns number of records in DBF file, including those marked as deleted.

{type}=2 : returns 0 to normal condition, 1 End Of File status (file pointer positioned beyond beginning of file).

{type}=3 : returns field number for each record.

♦ **Return value**

If **{date}**<0 is an error code. See DB errors table.

DB GO

♦**Syntax**

DB GO INT **hnd**, INT **pos**, VAR **err**

♦### **Parameters**

hnd: DB handle
pos: Record number
err: Error code

♦**Description**

Places writing/reading pointer inside the DB on the record No. **{pos}**. Positioning must be absolute and must not take account of the selected tag and DB sorting.

♦**Return value**

Error codes returned as **{err}** :

0	=	Ok
5	=	Record does not exist
<0	=	Error (see DB errors table)

DB LOCK

♦ **Syntax**

DB LOCK INT **hnd**, INT **rec**, VAR **err**

♦ **Parameters**

hnd:	DB handle		
rec:	No. of record for lock (if rec>0)		
	0	=	Lock entire DBF file
	-1	=	Lock DBF file and all index files
err:	Error code		

♦ **Description**

This instruction enables a record, all the DB or all the DB and the index files to be locked. Each new lock eliminates the previous lock, thereby eliminating the problem of dead-lock. Furthermore, if the entire DBF file (**{rec}**=0 or **{rec}**=-1) is locked it disables automatic locking and therefore the instructions that update DB (e.g. DB FLUSH) no longer lock-unlock and the file will remain locked until a DB UNLOCK instruction is issued.

♦ **Return value**

Result in **{err}**

0=	Ok	
50	=	The record or file has already been locked by another user
<0	=	Error

DB OPEN

♦Syntax

DB OPEN STR **fname**, INT **des**, VAR **hnd**, VAR **err** [,INT **shared**]

♦Parameters

fname: Name of Database to be opened.
des: Description of record to be used -1 if none
hnd: Handle of DB open. -1 if incorrect
err: Error code
shared: If current and =1: the file is opened so that it can be shared.

♦Description

Opens the Database **{fname}** and the assigned index file. Selects first tag and is positioned on first record.

When the file is opened a handle is assigned that must be used for all additional DB reference. The handle is returned to **{hnd}** and must be used in all the subsequent instructions in which reference is made to the created **DB**.

The DB is opened exclusively unless the **{shared}** parameter is current and =1. The parameter **{des}** specifies the record description that must be used. Value -1 can also be used as a parameter. In this case no record description is used and the instructions **DB GETREC** and **DB PUTREC** cannot be used. All the other **DB** instructions can be used normally.

♦Return value

If **{hnd}** =-1, **{err}** it contains an error code. See table with DB error codes.

DB OPENIDX

♦ **Syntax**

DB OPENIDX STR **fname**, INT **DB**, VAR **hnd**, VAR **err** [,INT **shared**]

♦ **Parameters**

fname: Name of index file to be opened
db: Database handle
hnd: Handle of index file. -1 if incorrect
err: Error code
shared: If current and =1: the file is opened so that it can be shared

♦ **Description**

Opens a new index file**{name}** for a DB that is already opened. The parameter **{db}** specifies the handle of the database from which another index must be opened. The handle returned to **{hnd}** must be used for each successive reference to the index file (for example in a DB CLOSEIDX). The file is opened in the EXCLUSIVE mode unless the **{shared}** parameter is set at 1.

♦ **Return value**

If {hnd} **=-1,** {err} contains an error code. See table with DB error codes.

DB OPENVIEW

♦ **Syntax**

DB OPENVIEW STR **dbname**, INT **flags**, VAR INT **res**, INT **start**

♦ **Parameters**

dbname: Dbase file name
flags: 1 = Opens read-only file
2 = Opens a mode window
4 = Disables new records appendix
8 = Disables record deletion
res: Operation result
start: If > 0 record on which to position oneself in open database.

♦ **Description**

Opens a Titano window to display a database; the parameter **{dbname}** must correspond to a .tdb file created with Titano or with an editor that can be purchased separately.

♦ **Return value**

Result in **{res}**:
0 = Ok key found
otherwise = Error

DB PACK

♦ **Syntax**

DB PACK INT **hnd**, VAR **err**

♦ **Parameters**

hnd: DB handle

err: Error code

♦ **Description**

Eliminates the records marked as deleted by the DB. In addition, it recreates the assigned index files and repositions the DB on the first record. The file that is to be made into a pack should first be opened exclusively so that other applications do not access the DB during the pack operation. If other software accesses the DB during the pack incorrect data may be read.

♦ **Return value**

Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB PUTFIELD

♦ **Syntax**

DB PUTFIELD INT **hnd**, VAR **err**, STR **field**, VAR STR **val** [,INT **flg**]

♦ **Parameters**

hnd: DB handle
err: Error code
field: Field name to write
val: Field contents (string or number)
flg: is used only for the DATE fields; if the date is 1 it must be in the format "CCYYMMDD"

♦ **Description**

Writes **{val}** into the **{field}** of the current record. The parameter is used **{flg}** only with the DATE fields: if the date is 1 it must be in the format "CCYYMMDD". If **{flg}** is zero or if it is absent the date must be formatted in accordance with the Windows configuration. This function is also available even if the index of the record description was not supplied when the DB was opened or created.

♦ **Return value**

Error codes returned as **{err}** :

0= Ok
<0 = Error (see DB errors table)

DB PUTREC

♦ **Syntax**

DB PUTREC INT **hnd**, VAR **err**

♦ **Parameters**

hnd:	DB handle
err:	Error code

♦ **Description**

Writes the record current in the database **{hnd}** by taking the contents of the fields of the variables that were configured in the Database section of the format configuration window. This function is only available even if the index of the record description was supplied when the DB was opened or created.

Return value

Error codes returned as **{err}**:

0= Ok

<0 = Error (see DB errors table)

DB QUERYEND

♦ **Syntax**

DB QUERYEND INT **hnd**

♦ **Parameters**

hnd: DB handle

♦ **Description**

Ends use of a query. Must always be recalled at the end of a query.

DB QUERYSET

♦ Syntax

DB QUERYSET INT **hnd**, STR **sort**, STR **filter**, VAR **err**

♦ Parameters

hnd:	DB handle
sort:	dBase expression for sorting
filter:	dBase expression for query
err:	Error code

♦ Description

Initializes a DB query. The two expressions **{sort}** and **{filter}** must be dBase expressions like those used to generate the index files. **{sort}** defines the sorting method. This may consist of just a field name or it can be a zero string (""). In this case the sorting set by the current tag is used. **{filter}** is an expression that gives the Logic result and therefore enables a given record number of the DB to be selected.

♦ Return value

Error codes returned as **{err}** :

0	=	Ok
<0	=	Error (see DB errors table)

DB QUERYSKIP

♦Syntax

DB QUERYSKIP INT **hnd**, INT **nrec**, VAR **err**

♦Parameters

hnd:	DB handle
nrec:	Shift record number
err:	Error code

♦Description

Shifts **{nrec}** record in the current query and enables the other query records to be accessed. The shift refers to the current record. **{nrec}** can be negative only if the appropriate flag is specified in the DB QUERYSTART.

♦Return value

Error codes returned as **{err}**:

0	=	Ok
3	=	End of File
4	=	Beginning of File
<0	=	Error (see DB errors table)

DB QUERYSTART

♦ **Syntax**

DB QUERYSTART INT **hnd**, INT **flg**, VAR **err**

♦ **Parameters**

hnd: DB handle
flg: **1** = Start from last record **2** = Enable negative skip
err: Error code

♦ **Description**

Activates a query with parameters set in the DB QUERYSET. This instruction might require a certain time to be carried out. After it has been carried out the file pointer will be located on the first record of the query (on the last if **{flg}**=1). To move onto the other records use DB QUERYSKIP. The parameter **{flg}** enables the start record to be set (0=First, 1=Last) and skips with negative (flg=2) can be authorized. If **{flg}**=1 negative skips are always authorized.

♦ **Return value**

Error codes returned as **{err}** :

0 = Ok
3 = End of File, No record current in the required query.
<0 = Error (see DB errors table)

DB REINDEX

♦ **Syntax**

DB REINDEX INT **hnd**, VAR **err**

♦ **Parameters**

hnd: DB handle

err: Error code

♦ **Description**

Regenerates the index files assigned to the DB. The Reindex operation is necessary when a DBF file has been updated but one of the index files (.CDX) has not been updated because, for example, it was closed or because the update was made by another node. The file on which a Reindex is carried out is opened exclusively so that other applications cannot accede to the index files during the reindexing operation. If other software accedes to the index files during reindexing incorrect data readings may occur.

♦ **Return value**

Error codes returned as **{err}**:

0= Ok

<0 = Error (see DB errors table)

DB SEEK

♦ **Syntax**

DB SEEK INT **hnd**, STR **value**, VAR INT **res** [,INT **next**]

♦ **Parameters**

hnd: DB handle
value: String to be searched for in current TAG
res: Result (0=Found)
next: If specified and if = 1 continues from previous SEEK

♦ **Description**

Seeks key **{value}** in DB by using current TAG. If TAG is a character **{value}** it can be shorter than the keys in the tag. In this case SEEK will stop at first record that corresponds to the characters supplied as **{value}**. If it is a DATE field **{value}** must be formatted as "CCYYMMDD".

♦ **Return value**

Result in **{res}**:

0 = Ok key found
2 = Key not found. Only partial match has been found between the tag and search keys. The file pointer is positioned on the next record.
3 = Key not found. The value to be sought is greater than the last key in the TAG.
5 = The record with the key does not exist.
<0 = Error (see DB errors table)

DB SEEKREAL

♦**Syntax**

DB SEEKREAL INT **hnd**, REAL **value**, INT VAR **res** [, INT **next**]

♦**Parameters**

hnd:	DB handle
value:	REAL - seek in current TAG
res:	Result (0=Found)
next:	If specified and if= 1 continue from previous SEEK

♦###**Description**

Seek the key **{value}** in the DB by using current TAG. Unlike DB SEEK, **DB SEEKREAL** enables any number to be searched for, regardless of how it is formatted (length and number of decimals).

♦**Return value**

Result in **{res}**:

0=	Ok key found
3=	Key not found. The SEEK value is greater than the last key in the TAG.
5=	The record with the key does not exist.
<0	= Error (see DB errors table)

DB SKIP

♦ **Syntax**

DB SKIP INT **hnd**, INT **nrec**, VAR **err**

♦ **Parameters**

hnd:	DB handle
nrec:	Record number
err:	Error code

♦ **Description**

Shifts the reading/writing pointer in the **{nrec}** DB forward (if positive) or backward (if negative). The shift occurs by using current TAG.

♦ **Return value**

Error codes returned as **{err}** :

0	=	Ok
3	=	End Of File. Positioning beyond end of file.
4	=	Beginning of File. Positioning before beginning of file.
<0	=	Error (see DB errors table)

DB TAGSELECT

♦ **Syntax**

DB TAGSELECT INT **hnd**, STR **name**, VAR **err**

♦ **Parameters**

hnd: DB handle
name: Name of TAG to be selected
err: Error code

♦ **Description**

Selects **{name}** as current TAG. A zero string "" is required for sorting by record number **{name}** .

♦ **Return value**

Error codes returned as **{err}**:

0 = Ok
<0 = Error (see DB errors table)

DB TAGSELECTED

♦ **Syntax**

DB TAGSELECTED INT **hnd**, VAR STR **name**, VAR INT **err**

♦ **Parameters**

hnd:	DB handle
name:	Name of selected TAG
err:	Error code

♦ **Description**

Returns current TAG as **{name}**. If **{name}** contains a zero string “” no selected TAG exists.

♦ **Return value**

Error codes returned as **{err}**:

0= Ok
<0 = Error (see DB errors table)

DB TOP

♦ **Syntax**

DB TOP INT **hnd**, VAR **err**

♦ **Parameters**

hnd:	DB handle
err:	Error code

♦ **Description**

Shifts the reading/writing pointer in the DB onto the first record of the file by using current TAG. The first record is not the first record added to the DB but the first record in the sorting method of the current TAG.

♦ **Return value**

Error codes returned as **{err}** :

0	=	Ok
3	=	End of File
<0	=	Error (see DB errors table)

DB UNLOCK

♦**Syntax**

DB UNLOCK INT **hnd**, VAR **err**

♦**Parameters**

hnd:	DB handle
err:	Error code

♦**Description**

Eliminates all the locks on a DB and on the assigned files.

♦###**Return value**

Value returned as **{err}**:

0= Ok
<0 = Error

DIFF DATE

♦ **Syntax**

DIFF DATE VAR **dif**, INT **g1**, INT **m1**, INT **a1**, INT **g2**, INT **m2**,INT **a2**

♦ **Parameters**

dif:	result in days. Obtained from the difference in dates
g1:	day of the first date
m1:	month of the first date
a1:	year of the first date
g2 :	day of the second date
m2 :	month of the second date
a2 :	year of the second date

♦ **Description**

This instruction calculates the difference between the two dates. It in fact calculates the difference **{g1,m1,a1}-{g2,m2,a2}**. The result is expressed in days and is stored in **{dif}**.

DIFF TIME

♦ **Syntax**

DIFF TIME VAR **dif**, INT **o1**, INT **m1**, INT **s1**, INT **o2**, INT **m2**,INT **s2**

♦ **Parameters**

dif:	result in seconds obtained from the difference in time
o1:	hour of first hour
m1:	minutes of first hour
s1:	seconds of first hour
o2:	hour of second hour
m2:	minutes of second hour
s2:	seconds of second hour

♦ **Description**

This instruction calculates the difference between two hours. It in fact calculates the difference **{o1,m1,s1}-{o2,m2,s2}**. **}**. The result is expressed in seconds and is stored in **{dif}**.

DISPLAY

♦**Syntax**

DISPLAY INT **x**, INT **y**, INT **col1**, INT **col2**, STR **s**, [,...**par**, ...]

♦**Parameters**

x,y: Writing coordinates
col1: Message color
col2: Background color
s: Formatting string containing %
par: Optional parameters, max. 16, assigned to % current in str

♦**Description**

Displays the string for the coordinates **{x,y}** using the colors **{col1}** and **{col2}**. The string displayed uses proportional characters. The string is processed by an instruction that combines the parameters **{s}** and **{v1...}** in the manner described below. String **{s}** can contain codes that enable whole or real numbers to be displayed. These codes are preceded by the character **%**. When the string is displayed it is scanned from left to right in search of special codes; for each code there must be an optional parameter. If there are more parameters than codes the additional parameters are ignored. On the other hand, if there are fewer codes than parameters, the result cannot be forecast.
Great care must be taken to ensure that the codes **%** match the correct type of variable or expression. For example, a **%d** **must** be matched by an INT. Similarly, a **%f** **must** be matched by a REAL variable. If this simple rule is not followed the results cannot be forecast.
If there are no **<%>** codes in the string the string need not be followed by numeric parameters. To display the character **<%>** write it doubled as: **%%**. A formatting code has the following composition::

%[flag][width][.accuracy]type

The flag and type fields consist of single characters whilst the width and accuracy codes are whole numbers without a mark

♦### **TYPE**

The field type is always compulsory and is what determines the matching type of optional parameter.

d: INT with sign
u: INT without sign
o: INT without octal sign
x X: INT without hexadecimal sign
c: INT interpreted as ASCII sign
f F: REAL in decimal format (0.001)
e E: REAL in exponential format (1e-3)
g G: REAL in the most compact format between %f, %e
s: STRING- string variable

♦**FLAG**

The flag field is optional and may have one of the following values:
- Justifies to the left within the specified field.

When used with **<x>** the characters Ox precede the number; when used with **<o>** 0 precedes the number. When used with **<e , f>** it forces the displayed number to contain the decimal point (e.g. 10 0000). When used with **<g>** it forces the number to contain the decimal point and prevents the excess zeroes from being truncated.
If the flag is not current the numeric parameters are justified on the right.

♦**WIDTH**

The width field is an entire non-negative that represents the total width of the field in which the user wishes to display the parameter datum. If the parameter width exceeds the width limits no number truncation takes place and all the characters are displayed.

♦### **ACCURACY**

The accuracy field varies according to the type of code.

<d,u,x> Accuracy specifies the minimum number of figures to be displayed. If the number of figures of the parameter is less than accuracy, the displayed value will be filled by zeroes. The value is not truncated when the number exceeds accuracy. The default value, i.e. the value if accuracy is absent, is 1.

<e,f> Accuracy specifies the number of figures that must be printed after the decimal point. The last figure is rounded off. The default value is 6. However, is accuracy is zero or if there is only , no figure is displayed after the decimal point.

<g> Accuracy specifies the greatest number of significant figures that must be displayed. The default value is 6.
Instead of the width and accuracy values a * can be specified. In this case the required value will be taken from the list of optional parameters. This parameter must be displayed inside the field. For example:

```
"%*d",10,w1
```

Accuracy has the value 10 and variable w1 is justified on the right inside dimension field 10. The following special characters can be inserted into the field.

\n = return to top
\f = jump page

DISPLAY BOX

♦ **Syntax**

DISPLAY BOX INT **box**, INT **fg**, INT **bk**,INT **code**, STR **str** [,...**par**, ...]

♦ **Parameters**

box: Number of reference box in which to print if -1 uses the default box. If there is no default box all the screen is used.

fg: Color of message

bk: Color of background

code: Formatting code:

0 = Centering

1 = Align to left

2 = Align to right

8 = Align at top

16 = Align at bottom (1,2 and 8,16 can be added up to combine the effects, for example 17 align to left at bottom)

str: String for printout containing %

par : Optional parameters, max. 16, assigned to % current in string

♦ **Description**

Analogous to the **DISPLAY** but prints only inside a box in proportional mode. Also enables alignment and centering within box. The box is completely filled with the background color.

ELLIPSE

♦ **Syntax**

ELLIPSE INT **xc**, INT **yc**, INT **rx**, INT **ry**, INT **col**

♦ **Parameters**

xc,yc:	Center coordinates
rx,ry:	Ellipse
col:	Color

♦ **Description**

Traces a color ellipse **{col}**, center **{xc,yc}** and axes **{rx,ry}.**

The coordinates for the top left hand corner of the monitor are expressed in PIXEL. The color **{col}** is defined in the table *Standard event colors Table.*

Macros RGB and RGB_A can be used as parameter{col}.

END

♦ **Syntax**

END

♦ **Description**

This function finishes the end of the program flow. This is made necessary because the flow, after the last instruction of the current event has been carried out, continues to carry out the first event that follows in the list of the Logoview NT events.

The **END** function is used when several different trigger events are defined to keep them separate from one another.

The **END** command can clearly be used more than once for the whole program. An event must always finish with an **END** or a **RETURN**. These instructions are interchangeable.

END WHILE

♦ **Syntax**

END WHILE

♦ **Description**

This instruction enables the program to recognize the end of the repetition loop .

This instruction must be used only in conjunction with the **WHILE** function. If the interpreter encounters an **END WHILE** function without the corresponding **WHILE** function an error message is generated.

EVENT

♦Syntax

EVENT INT **e**

♦Parameters

e: Event number

♦Description

Program control is transferred to the event **{e}**. A leap takes place and there is no possibility of returning to the event **{e}**. In this case again, the event number **{e}** can be replaced by a variable that contains the number.

It is also possible to use the name assigned to the event as a parameter **{e}**.

♦Example:

```
Event 10:

* If T4 is the same as 1 call up event 40
IF  ?T4=1
   EVENT  40
END IF

END
```

EXEC

♦Syntax

EXEC STR **prog_name**, VAR INT **err** [, INT **flags**[,INT **type**]]

♦Parameters

prog_name:	Name of program to be launched		
err:	Error code		
flags:	Application of window status run		
	0	=	Normal
	1	=	Maximized
	2	=	Minimized
	3	=	Hidden
type:	Type of function		
	0	=	Run only practicable programs
	1	=	Also documents

♦Description

Runs a Windows program. In **{prog_name}** specify the path completed by the application and the additional parameters. Use the parameter **{flags}** to indicate the status of the main window of the application. If the complete path is not specified in **{prog_name}** the application will be sought in the following directories:

1. Directory in which Logoview NT is located
2. Directory of Windows System32
3. Directory of Windows
4. Directories specified in the path environment variable

If **{type=0}** is specified EXEC can run only practicable programs. If **{type=1}** is specified EXEC can also open documents. Depending on the file extension the function can also understand the program that must be run and runs it by using the required document as a parameter. This is possible only if the type of document has been correctly registered in the Windows registry.

If **{type=1}** is specified it is not possible to add parameters for running the program; **prog_name** must contain only the name of the program or the document to be implemented.

♦Return value

Error codes returned as **{err}**:

0=	Ok	
-1	=	Runnable non found
-2	=	Memory exhausted

EXIT WHILE

♦ **Syntax**

EXIT WHILE

♦ **Description**

This instruction enables a **WHILE** instruction to be interrupted even if the condition is not fulfilled. Implementation comes from the instruction after the next **END WHILE**.

FCLOSE

♦ **Syntax**

FCLOSE INT **handle**

♦ **Parameters**

handle: Index of a file opened with FOPEN

♦ **Description**

Closes a file opened with FOPEN.

FILE COPY

♦ Syntax

FILE COPY STR **name**, STR **dest**, INT **err**

♦ Parameters

name:	Source file name
dest:	Destination
err:	Error code

♦ Description

With this function a file can be copied from one file to another or from one sub directory to another; the copied file will have the same file name. Note that this instruction interrupts the multithreading of Logoview NT (acquisition from the field, alarms update, etc.) but suspends running all the events until the operation is completed. In order not to interfere with the running of the events, in the event of copying operations onto particularly slow instruments a batch file should be written and run by Logoview NT by an EXEC.

♦ Return value

Return value in **{err}** :

0	=	OK
1	=	Reading error
2	=	Writing error
3	=	The source file does not exist
4	=	File cannot be copied onto itself.

♦ Example

Suppose that we wish to copy the file ’test.structure’ in the sub directory GMONIT of drive C into the WORK subdirectory of drive A and to obtain the return value into the “ERR” variable:

```
FILE COPY "C:\GMONIT\Prova.scr","A:\WORK",#ERR
```

FILE OF

♦ **Syntax**

FILE DEL STR **name**, VAR **err**

♦ **Parameters**

name: Name of file to be deleted
err: Error code

♦ **Description**

This function enables a file to be deleted. Note that using this instruction does not interrupt the multithreading of Logoview NT (acquisition by field, alarms update, etc.) but suspends all events until the operation is completed.

♦ **Return value**

Return value in **{err}** :

0= OK
2= Unknown file
13 = Accessed denied

♦ **Example**

Cancel file “Test.scr”

FILE OF "Test.scr"

FILE REN

♦ **Syntax**

FILE REN STR **old name**, STR **new name**, VAR **err**

♦ **Parameters**

old name:	Old file name
new name:	New file name
err:	Error code

♦ **Description**

This function enables a file name to be changed. Note that using this instruction does not interrupt the multithreading of Logoview NT (acquisition by field, alarms update, etc.) but suspends all events until the operation is completed.

♦ **Return value**

Return value in **{err}** :

0	=	OK
2	=	Unknown file
13	=	Accessed denied
18	=	Cross Device Link (Rename onto different disks)

♦ **Example**

We rename onto disk A from *'Old test.scr'* to *'New test.scr'*.

FILE REN "A: 'Old test.scr. ","A: 'New test.scr'"

FILEDLG

♦ **Syntax**

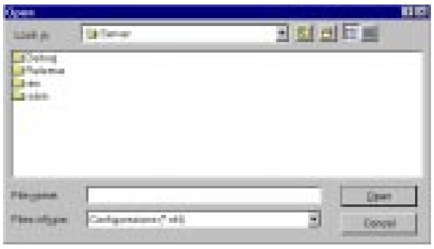
FILEDLG STR **title**, STR **filter**, STR **ext**, STR **idir**, INT **flags**, VAR STR **file**, VAR INT **res**

♦ **Parameters**

title: Window title
filter: Filter on files extension
ext: Default extension
idir: Initial directory
flags: Flags combined with OR "|" character
1 = Window for open (otherwise save)
2 = FILEMUSTEXIST (Only open) Only existing files can be selected.
4 = PATHMUSTEXIST Only existing directories can be selected
8 = CREATEPROMPT (Only open) If the file does not exist the user is asked if he wishes to create one.
16 = NOCHANGEDIR Restores initial directory after closing the window.
32 = NONETWORKBUTTON Eliminates the network button
file: File default name when window opens and name is keyed in by the user on closing
res: Error code

♦ **Description**

This function enables the standard Windows window for files to be opened.



The types of file that can be managed by the window are specified by the **{filter}.** The filter string is made up as follows:

description_type_file_1 | **.extension_1* | *description_type_file_2* | **.extension_2* | ...
description_type_file_No. | **.extension_ No.* ||

In the **{file}** field the default file name is set that is suggested to the user when the window is opened. The name that is keyed in by the user is entered when the window is closed. The name obtained at the exit will always have a path, e.g. "c:\temp\backup.dbf". This instruction enables a choice to be made between two types of window: the 'Open' window and the 'Save' window. Both types behave in the same way. The only differences lie in the message in the button that confirms the operation and the warning messages that appear when a file already exists. Window "Open":

- In the button the message 'Open' appears.
- If the file does not exist and flag **2** is specified, the user is informed of the error and closing of the window is prevented.

'Save' window:

- In the button the message ’Save’ appears.
- If the file exists the user is asked if he wishes to overwrite it. No message is displayed if the file does not exist.

♦ Return value

Return value in **{err}**:

0	=	OK
otherwise	=	Error

♦ Example

We open a window that enables a dbf file to be selected and copied onto floppy disk (A:).

```
LET $fname=”file.dbf”
FILEDLG “Open DBF file”, “File DBF | *.dbf | All files| *.* ||”, “dbf”, “C :\DB\”, 1+2, $fname,
:res
FILE COPY $fname,”A:\”,:res
```

FIND REC

♦ **Syntax**

FIND REC INT **ar**, VAR **s**, INT **g**, INT **m**, INT **a**, INT **time**, INT **min**, INT **sec**

♦ **Parameters**

ar:	File number
s:	Search result
g:	Day of month
m:	Month
a:	Year
time:	Time
min:	Minutes
sec:	Seconds

♦ **Description**

A search is made inside the record file **{ar}** with the closest date to the one set as a parameter.

As for the other functions that refer to the files, the file must have been first defined and opened.

The file is one of the 20 defined in the *History Files* section of the window *Formats configuration*. It can be accessed by the button on the main toolbar or by the equivalent menu command.

For the description of the file characteristics and the creation of a record see chapter 9: ’Formats’ of the PRIMER of Logoview NT.

The function enables the record to be accessed whose entry date is closest to the one specified as a parameter.

♦ **Return value**

Result in **{s}** :

0	=	Ok record found
1	=	Record non found

♦ **Example**

We wish to read a record of file ALARMS.ARC and display the message found.

Event 10:

```
SEEK REC. 1, #TRIG, DAY MONTH, YEAR, TIME,
SECONDS
READ REC.  1,#TRIG,#STATUS
RETURN
```

FIND WINDOW

♦ **Syntax**

FIND WINDOW STR **title**, INT **op**, VAR INT **handle**

♦ **Parameters**

title: Title of window or screen name

op: Search type

0=Logoview window, title datum
1=Logoview window, screen name datum
2=Main window, Windows applications, title

handle: Handle of sought window, 0 if not found

♦ **Description**

Searches for a window with a title datum or one that displays a frame datum.
op = 0: instruction seeks Logoview window whose title starts with the string given in the **title** parameter. Search occurs only between the main Logoview windows.
op = 1: instruction seeks Logoview window in which the previous frame is displayed as **title** parameter.
op = 2: the instruction checks all the main windows of current Windows operations for a main program window whose title starts with the string given in **title**.
The comparison between the strings is not case-sensitive.
The returned handle can be used with instructions that request a window handle as a parameter, e.g. SET WINDOWPOS.

WARNING: handle variable **MUST** be of the **DWORD** type. Any other type may cause faults in the instructions that use the handle.

♦ **Return value**

Result in **{handle}** :

0 = The sought-after window does not exist.
Otherwise = Window handle

♦ **Example**

Seeks main Logoview window and reduces it to an icon:

```
Event 10:
   FIND WINDOW “Logoview –“,2,:hwnd
   SHOW WINDOW :hwnd,6
```

FLUSH FORM

♦ **Syntax**

FLUSH FORM INT **form**, INT **op**, VAR INT **err** [, STR **printer**]

♦ **Parameters**

form: Printout form number

op: Operation
- 0 = Print on default printer
- 1 = Print on printer given as additional parameter
- 2 = Print on printer selected by operator
- 3 = Delete buffered pages

err: Variable for operation result
- 0 = OK
- -1 = Incorrect parameters
- -2 = Temporary print file cannot be created
- -3 = Required printer cannot be opened
- -4 = Printing cannot be started
- -5 = Temporary file cannot be opened in order to print current page
- -6 = Impossible to read from temporary page or from damaged file
- -7 = Current page cannot be printed
- -8 = Temporary print file cannot be written
- -9 = Unknown error

printer: Name of printer in the event of (op = 1)

♦ **Description**

This instruction enables the pages in print format **{form}** to be printed that have been stored but have not yet been printed. The format number is set in the configuration window of the print formats. If **{op}** is the same as 3 the non-printed pages will be deleted without being sent to the printer.

FONT

♦**Syntax**

FONT STR **name**

♦**Parameters**

name: File name of characters font

♦**Description**

If this function is used it is possible to load a new font onto the memory from the *Font* section of the *General Characteristics Window.* The font that is loaded is used with the DISPLAY instruction.

If the font does not exist an error message is displayed.

FOPEN

♦ **Syntax**

FOPEN STR **file,** INT **code,** VAR INT **handle**

♦ **Parameters**

file: Index of a file opened with FOPEN

code: Method of opening file:

0 = Read

1 = Write (if the file exists it destroys it)

2 = Write on existing file

3 = Append

handle: Index of file to use with instructions F...

>=0 = Handle of file

-1 = File cannot be opened

-2 = No more indices are available (max. 20)

♦ **Description**

Opens text file to use with all **F...** functions.

This type of function is designed for reading and writing text files. The returned **{handle}** must be used only with **F...** functions and must not be confused with the one used **DB ...** functions.

FREAD

♦ **Syntax**

FREAD INT **handle**, VAR STR **dest**, INT **nchar**, VAR INT **err**

♦ **Parameters**

handle:	Index of a file opened with FOPEN
dest:	String variable into which to enter the read characters
nchar:	Number of characters to read
err :	Result of operation

♦ **Description**

Reads a certain number of characters from the file with the index **{handle}** and puts them in the string variable **{dest}**.

♦ **Return value**

Return value in **{err}**:

-1	=	Incorrect handle
otherwise		number of bytes read

FREADLN

♦Syntax

FREADLN INT **handle**, VAR STR **dest**, VAR INT **err**

♦Parameters

handle: Index of a file opened with FOPEN
dest: String variable into which to enter the read line
err: Result of operation

♦Description

Reads a line from the file with an index **{handle}** and inserts it into the variable **{dest}**. A line cannot exceed 512 characters. The lines are delimited by the combination cr lf.

♦Return value

Return value in **{err}**:

-1 = Incorrect handle
otherwise number of bytes read

FSEEK

♦ Syntax

FSEEK INT **handle**, INT **op**, VAR INT **stat**

♦ Parameters

handle : Index of a file opened with FOPEN

op: Position

0	=	Start file
1	=	End of File
3	=	Returns status

stat: File status

♦ Description

Positions file pointer with index **{handle}** in the position indicated in **{stat}**; can also return current file status if **{op}** = 3 without varying position. All subsequent operations (**FREAD**, **FREADLN** and **FWRITE** are carried out from the point shown with this instruction). The instruction can be used only for positioning at the start or the end of the file.

♦ Return value

Return value in **{stat}**:

0	=	Ok
1	=	Eof
2	=	Error

FWRITE

♦ Syntax

FWRITE INT **handle**, STR **src**, VAR INT **err**

♦ Parameters

handle: Index of a file opened with FOPEN
src: String to be written
err: Result of operation

♦ Description

Writes string **{src}** into the file with index **{handle}**.

♦ Return value

Return value in **{err}** :

< 0 = Error
otherwise number of written bytes

GET DATE

♦ **Syntax**

GET DATE VAR **day**, VAR **month**, VAR **year**

♦ **Parameters**

day: Variable where day of month is returned
month: Variable where month is returned
year: Variable where year is returned

♦ **Description**

This instruction reads the date of the system at a precise time that is divided into 3 variables: **{day}, {month} and {year}**.

GET REC

♦ **Syntax**

GET REC INT **arc**, VAR **status**, VAR **pos**, INT **flag**

♦ **Parameters**

arc:	File number		
status:	Error code		
pos:	Position inside file or record number.		
flag:	**0**	=	Reading position
	1	=	Writing position
	2	=	Record number in file
	3	=	MAX. number of records in file (only circular files)

♦ **Description**

This instruction enables a certain number of data to be obtained on a given history file. The parameter **{flag}** enables the date to be specified that one wishes to obtain:

0: Current position of file reading

1: Current position of writing on file

2: Number of records inside file

3: Maximum number of records that file can contain; this last parameter is valid only for circular files.

This information is then stored by the instruction in the variable **{pos}**.

♦ **Return value**

Return value in **{err}**:

0	=	OK
otherwise		error

♦ **Example**

```
GET REC 0,#stato,@nrec,2
```

GET TIME

♦ **Syntax**

GET TIME VAR **time**, VAR **minutes**, VAR **seconds**

♦ **Parameters**

time: Variable in which system time is returned
minutes: Variable in which system minutes are returned
seconds: Variable in which system seconds are returned.

♦ **Description**

This instruction gives the system time at a given moment, divided into 3 variables: **{time}, {minutes} and {seconds}**.

GET WNDSTATUS

♦ **Syntax**

GET WNDSTATUS INT **hwnd**,VAR INT **status**

♦ **Parameters**

hwnd: Window handle -1 uses frame assigned to the event
status: Window status

♦ **Description**

Returns present window status. The parameter **hwnd** must be a handle of the window that is returned with the instruction FIND WINDOW. Alternatively, -1 can be specified to intervene on the frame assigned to the event. In this case the event must be activated by a graphic element of a frame such as a button in order to operate correctly.

♦ **Return value**

Result in **{status}** :

-1: **hwnd** does not contain a window handle
0: Normal
1: Minimized
2: Maximized

GET WNDRECT

♦ **Syntax**

GET WNDRECT INT **hwnd**,INT **type**,VAR INT **x**,VAR INT **y**,VAR INT **dx**,VAR INT **dy**

♦ **Parameters**

hwnd: Window handle -1 uses the frame assigned to the event
type: Type of coordinates to read
0= Window coordinates
1= Internal window area coordinates (client)
x,y: Window position
dx,dy: Window dimensions

♦ **Description**

Returns window position and dimensions or those of the internal window area (customer). The parameter **hwnd** must be a window handle returned by the instruction FIND WINDOW. Alternatively, -1 can be specified to intervene on the frame assigned to the event. In order to operate effectively in this case the event must be activated by a graphic element of a frame such as a button.

♦ **Return value**

Result in **{x,y,dx,dy}** :
x,y = Window position, if **type** = 1 contains always zero.
dx,dy = Dimensions

♦ **Example**

Halves the dimensions of the frame “SCR000”:

```
FIND WINDOW “SCR000”,1,:hwnd
GET WNDRECT :hwnd,0,:x,:y,:dx,:dy
SET WINDOWPOS :hwnd,0.0,dx/y,dy/2,1
```

GR. SET

♦ **Syntax**

GR. SET

♦ **Description**

There are two ways of obtaining a display of the foreground colors. Either the defined foreground color can be can be displayed or the XOR of the foreground and background colors can be highlighted.

The instruction **GR.SET** activates the first of the two modes. After this instruction has been run all the graphic operations will run their graphic elements with the foreground color, which will overwrite the background color.

GR. XOR

♦ **Syntax**

GR. XOR

♦ **Description**

Activates the XOR logic operation for graphics operations. In other words, all the graphics operations will carry out an XOR between the background color and the foreground color.

In addition to the GR.SET function, it enables the graphic element to be displayed with a color that is made up of the composition between the background color and the color that is specifically indicated for the graphics function.

HARDCOPY

♦ **Syntax**

HARDCOPY STR **lpt** [, INT **rot**, INT **left**, INT **top**, INT **dimx**, INT **dimy**]

♦ **Parameters**

lpt: Printer name

rot: Rotation
0 = Horizontal printout (landscape)
1 = Vertical printout (portrait)

left: Margin on the left in mm.

top: Top margin in mm.

dimx: Horizontal margin in mm.

dimy: Vertical margin in mm.

♦ **Description**

This instruction enables a graphics printout to be made of the frame currently shown in close-up. If the dimensions are 0 the frame will have to be adapted to the page dimensions.

IF, ELSE, END IF

♦ **Syntax**

IF cond

.

...instructions block

.

[**ELSE**]

.

...[instructions block]

.

END IF

♦ **Parameters**

cond: conditional expression

block: list of instructions to be run

♦ **Description**

If the condition is true the instructions between **IF** and **ELSE** are run; if it is false the instructions between **ELSE** and **END IF** are run. The length of the jump between the start and end of **IF** limits the maximum length of the event.

The condition may be single or multiple. In the event of a single condition there are no great problems, except for syntactical ones. In the case of multiple conditions the single conditions between parentheses must be closed up again. For example:

```
((#VAR1>>34)&(!TR1=1))|(!TR2=0)
```

In this case the condition is checked until byte **TR2** is equal to 0 or word **#VAR1** is greater than 34 and byte **!TR1** is equal to 1.

The instructions that have to be run if the condition arises are terminated by the instruction **END IF** if the condition is monodirectional or else an **ELSE** instruction can be placed between **IF** and **END IF**.

Up to 50 other **Ifs** can be nested inside the **IF**.

INPUTDLG

♦ **Syntax**

INPUTDLG STR **title**, STR **text**, INT **type**, VAR **val**, VAR INT **res** [, INT **min** [, INT **max.**]

♦ **Parameters**

title: Window title

text: Window text

type: 0 = No restriction
1 = Accepts only numbers

val: Variable containing default value at input for display in window and value at the exit keyed in by the user.

res: 0 = Cancel or 1 = Ok

min: If current it is the minimum value that the user can enter

max: If current it is the maximum value that the user can enter

♦ **Description**

Opens a dialog window with **{title}** to ask the customer a value. The request is specified in the parameter **{text}.** The value that the user can key in may be purely numeric (type = 1) or without restriction. If the value is numeric the optional parameters (min. and max.) limit the entry interval. It is also possible to specify a default value in the parameter **{val}**. This is proposed to the user when the dialog window is opened; this value will contain the value keyed in by the user until the window is closed.

♦ **Return value**

Value keyed in by the user returned in **{val}**.

Button returned in **{res}:**

0 = The user has pressed the OK button

1 = The user has pressed the CANCEL button

INPUT SCREEN

♦ **Syntax**

INPUT SCREEN INT **bx**, VAR **ris**

♦ **Parameters**

bx: box on which to make the input (-1= start from first one available)
ris: return to exit from input code

♦ **Description**

Activate input in an interactive field. This enables the user to insert a value into the specified interactive field. It does not allow shifts to other fields.

♦ **Return value**

Button returned to **{ris}:**

0	=	ENTER
1	=	TAB
-1	=	Shift TAB
16	=	ESC
128	=	No interactive field

KILL

♦ **Syntax**

KILL INT **n**

♦ **Parameters**

n: Process number

♦ **Description**

This function’s task is to complete the active processes indicated by the parameter **{n}**.
If **{n}** has a value between 1 and 8, it stops the process being run that was defined by that value. On the other hand, if **{n}** has a value that is the same as 0, the active processes will be terminated except for the process to which the event being run belongs.
In this way all the specified concurrent activities will be terminated and the application will continue to run the process under examination plus any processes that have not been expressly terminated.

LET

♦ Syntax

LET VAR **v =** INT **expr**

♦ Parameters

v: destination variable of assignment
expr: value to be assigned to variable

♦ Description

Assigns variable VAR expression value **{expr}**.
If the variable is REAL the expression is calculated by means of REAL numbers, so all the INT numbers are transformed into REAL numbers before the calculation is made.
Conversely, when the variable is INT (WORD, BYTE or TRIGGER) the entire calculation is carried out by INT numbers.
Each time that the user wishes to assign a value to a variable from the program or if mathematical operations have to be carried out between numbers and variables, this function must be used.
The function accepts both the mnemonic names of the variables and the offsets.
In the case of offsets first of all the character identifying the type of variable used must be keyed in:

```
LET  #03=10
```

In the previous example we assigned the value 10 to the third WORD configured. If we had not specified the type of variable the program would have generated an error message.

♦ Example

Assign value 100 to WORD variable (#) of offset 10
LET #10 = 100
Assign value 100 to WORD variable (#) of "temp" name
LET #temp = 100

LINE

♦ **Syntax**

LINE INT **x1**, INT **y1**, INT **x2**, INT **y2**, INT **col**

♦ **Parameters**

x1,y1,x2,y2:	Line coordinates
col:	Line color

♦ **Description**

Traces a color line **{col}** on the work videoscreen. The coordinates **{x1,y1,x2,y2}** of the top left-hand corner of the monitor are expressed in PIXEL. The color is one of those defined in the *Standard event colors window* .

Macros RGB and RGB_A can be used as parameter {col}.

LOAD AREA

♦ **Syntax**

LOAD AREA INT **x1**, INT **y1**, INT **x2**, INT **y2**, STR **name**, INT **flag**

♦ **Parameters**

x,y: Coordinates on which to position the image

name: File name of load area

flag: **1** = Adapts to rectangle (x1,y1,x2,y2) only if image is greater

2 = Adapts image to rectangle and ignores proportions between **4** = Centers image in the rectangle

8 = Maintains the original dimensions unchanged, i.e. ignores (x2,y2)

16 = Adapts image to default box and then ignores parameters (x1,y1,x2,y2)

♦ **Description**

This function enables a portion of the pre-set image that is stored on the file to be displayed and positions it on coordinates **{x,y}**. The **{flag}** parameter enables the image to be adapted to the reference rectangle as required. If the complete file path is not specified in the **{name}** field the file will be sought in the application directory.

LOAD VAR

♦ **Syntax**

LOAD VAR STR **file**, VAR **err**

♦ **Parameters**

file:	Name file			
err:	Error code			
	0	=	OK	
	-1	=	File opening error. File does not exist.	
	>0	=	N. of the string in which there is an error	

♦ **Description**

Imports the variables contained in a text **{file}** into Logoview NT. The text file format is generated by SAVE VAR. If a complete path is not specified the file will be sought in the application directory.

LOCAL FONT

♦ **Syntax**

LOCAL FONT STR **name**

♦ **Parameters**

name: Name of character font

♦ **Description**

This function enables one of the character fonts of the section *General Characteristics Window* to be loaded into the memory but the use will be reserved to the single process that loads it. This is useful when different processes write on the same video page and processes have to be distinguished from one another.
If the font does not exist an error message will be displayed

LOCK

♦ Syntax

LOCK

♦ Description

Using the **LOCK** function defines the start of a sequence of instructions inside the event that are a critical point in running the application.

The indicated sequence containing a task will be the only one running whilst the other concurrent processes will be locked.

The timed or assigned trigger tasks will remain locked until the instruction **LOCK**: **UNLOCK** is given.

♦ Example

Suppose that there are two events: event A and event B and an ’instructions area’ that should be run by one event at a time without overlapping.

This is a classic example of a critical region that must be protected by a pair of functions **LOCK** and **UNLOCK**.

```
Event A :
...
LOCK
... critical code ...
UNLOCK
...
Event B :
...
LOCK
... critical code ...
UNLOCK
...
```

MCI COMMAND

♦ **Syntax**

MCI COMMAND STR **cmd**, VAR STR **ret**, VAR **err**

♦ **Parameters**

cmd:	String with command
ret:	Returned value
err:	Result of operation

♦ **Description**

Runs an MCI command. MCI is a Windows interface for controlling multimedia devices. This interface is used to pilot all types of multimedia devices: for example, reproducing files**.avi**, .**waw** etc.

For further information consult the SDK manuals of Windows.

♦ **Example**

MCI COMMAND “PLAY pluto.avi”, #VAL, #RIS

MCI GETERROR

♦ **Syntax**

MCI GETERROR INT **err**, VAR STR **msg**

♦ **Parameters**

err:	Error code returned by **MCI COMMAND**
msg:	Error code message

♦ **Description**

Returns error description string **{err}** returned by **MCI COMMAND**.

MENU COMMAND

♦ **Syntax**

COMMAND MENU STR **cmd**, INT **type**

♦ **Parameters**

cmd: Activate menu item

type: Menu localization system

0: cmd must contain description of menu position

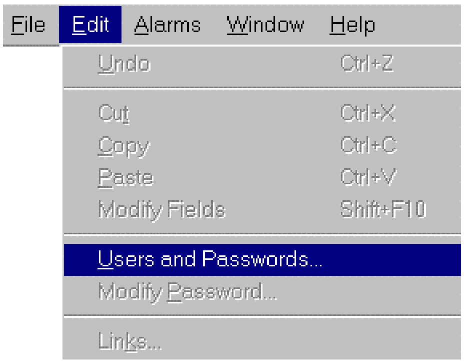
1: cmd must contain string that appears in menu

2: cmd must contain a sequence of strings separated from the character

♦ **Description**

This instruction enables the Logoview runtime menu items to be activated from an event. The item can be selected in two types, depending on the parameter **{type}**.

Type = 0: In this case the menu is identified by its position, which must be described in the string **{cmd}**. To identify an item inside a submenu specify the complete path that enables the item to be selected. Within each menu the positions are counted from the item that is in the highest position, jumping any separators. For example, if we wish to activate the item 'Users and Password' in the 'Edit' menu:



To identify this item , select the item "Edit" from the main menu (position 1) and then "Users and password" in the "Edit" submenu (position 5). To activate the item "Users and password" specify string "1 5" in **{cmd}**.

Type=1: In this case the menu item that is to be activated must be identified by the string that appears in the menu. To activate the item "Users and password" just specify string "1 5" in **{cmd}**.

The comparison between the strings is not case-sensitive. It is also sufficient specify just the first part of the string, e.g. 'Users and' because no other items start with 'Users'. This disadvantage of this system is that it depends on the language used during the development phase.

Type=2: In this case the menu item that is to be activated is identified by a sequence of strings (separated by the character '|') that reconstruct the entire menu sequence that is to be opened. For example, to select the item "Users and password" in the "Edit" menu specify the string "Edit | Users and password" **{cmd}**.

Warning: the Logoview runtime menu varies according to context. A certain menu item may be activated whilst a frame is displayed but be unavailable if the user is consulting a trend or a Titano page.

♦Example

Activate “Recognize Page' from the 'Alarms' menu:

```
COMMAND MENU “2 2”,0
COMMAND MENU “Recognize page”,1
COMMAND MENU“ Alarms|Recognized page”,2
```

MESSAGEDLG

♦ **Syntax**

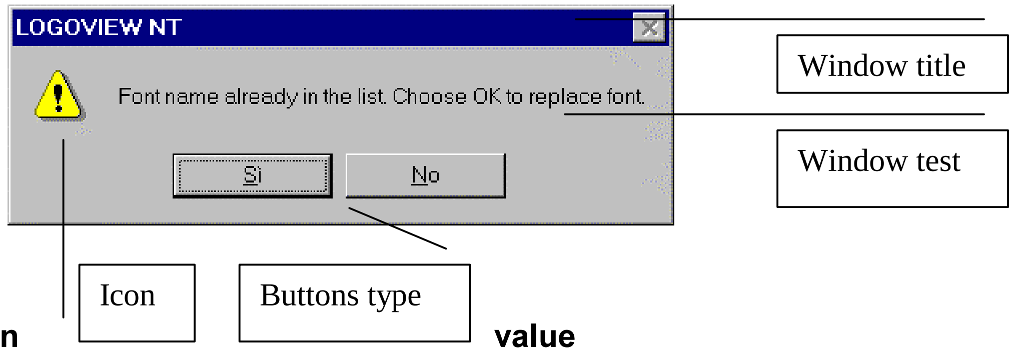
MESSAGEDLG STR **title**, STR **text**, INT **type**, INT **icon**, VAR INT **res**

♦ **Parameters**

title: Window title
text: Window text
type: Type ’Buttons’ in window
0 = Ok
1 = Ok, Cancel
2 = Yes, No
3 = Yes, No, Cancel
4 = Retry, Cancel
5 = Abort, Retry, Ignore
icon: icon type
0 = None
1 = IconQuestion (Question mark)
2 = IconStop (Stop)
3 = IconInformation
4 = IconExclamation (Exclamation mark)
res: Button pressed

♦ **Description**

Opens a window to display a message. Returns in **{res}** the identify of the pressed button. The window comprises:


LOGOVIEW NT

Font name already in the list. Choose OK to replace font.

♦ **Return value**

Identification of button pressed in **{res}** :

0 = Cancel
1 = Ok
2 = Yes
3 = No
4 = Abort
5 = Retry
6 = Ignore

MM CLOSE

♦ **Syntax**

MM CLOSE STR **obj**

♦ **Parameters**

obj: object on which to run the command

♦ **Description**

Runs the closing command on the MCI object specified as a parameter. This command is the equivalent of a closure made using the MM COMMAND instruction.

For further information on MCI commands check the SDK documentation of Windows. If special objects such as overlay or video acquisition cards are used refer to the relative manual.

MM COMMAND

♦ **Syntax**

MM COMMAND STR **obj**, STR **cmd**, VAR STR **ret**, VAR **err**

♦ **Parameters**

obj: Object on which to run the command
cmd: MCI command
ret: String variable with returned value
err: Result of operation

♦ **Description**

Runs the MCI command specified by the **{cmd}** parameter on the object **{obj}.**

This command is the most general of the MM family. It enables any MCI command to be run on a multimedia object.

For further information on MCI commands consult the Windows SDK documentation. If special items such as overlay or video acquisition cards are used see the relevant manual.

MM OPEN

♦ **Syntax**

MM OPEN STR **obj**, STR **file**, VAR **err**

♦ **Parameters**

obj: Object on which to run the command
file: File or device name
err: Result of operation

♦ **Description**

Runs the opening command of a multimedia object specified by the **{file}** parameter of the object **{obj}** in a frame. This command is the equivalent of using the MM command to open. .

For further information on MCI commands consult the Windows SDK documentation. If special items such as overlay or video acquisition cards are used see the relevant manual.

MM PLAY

♦**Syntax**

MM PLAY STR **obj**,

♦**Parameters**

obj: Object on which to run the command

♦**Description**

Runs the start command of a multimedia object specified by **{obj}** parameter current in a frame. This command is the equivalent of using the MM COMMAND to run the multimedia object.

For further information on MCI commands consult the Windows SDK documentation. If special items such as overlay or video acquisition cards are used see the relevant manual.

MM STOP

♦ **Syntax**

MM STOP STR **obj**

♦ **Parameters**

obj: Object on which to run the command

♦ **Description**

Runs the stop command for a multimedia object specified by the **{obj}** parameter in a frame. This command is the equivalent of using the MM COMMAND to stop.

For further information on MCI commands consult the Windows SDK documentation. If special items such as overlay or video acquisition cards are used see the relevant manual.

OPEN FILE

♦ ###**Syntax**

OPEN FILE STR **name**, INT **arc**, INT **type**, VAR **err** [,INT **flag**]

♦ **Parameters**

name: Name of file to be opened

arc: File number

type: **0** = Reading and writing are opened: if the file does not exist it is created

1 = Reading and writing are opened: if the file does not exist an error code is returned

2 = Read-only is opened:, if the file does not exist an error code is returned

err: Error code(0 = no error)

flag: **0** = Creates Logoview NT file (default)

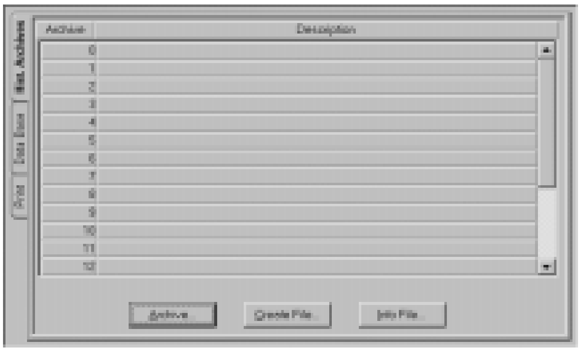
1 = Creates file compatible with LW32

♦ **Description**

Opens a file to store files. If the file does not exist a new one is created or an error code is returned.

The opened file will be used to store the records defined by the file **{arc}**.

The file is one of the 20 that can be defined in the History Files section by the formats configuration window



Archive	Description
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

and can be accessed by means of the button on the toolbar.

For the description of file characteristics and the creation of the record see the description in the chapter on files in the Formats section of the manual.

♦ **Return value**

0: No error

-1: Error whilst opening file

-2: Error whilst creating file

-3: Error in format number

-4: Error, writing (disk full?)

-5: Error, reading

-6: No space left for operations on file fields

-7: Variable does not exist
-8: Incorrect header
-9: Incorrect version number
-10: READ_ONLY file opened
-11: Writing pointer incorrectly positioned.

OPENURL

♦ **Syntax**

SCREEN STR **url**

♦ **Parameters**

url: Web resource path or address to be opened

♦ **Description**

Opens a new window that enables pages in html, i.e. in standard Web page format to be displayed. The pages may be local or on a Web server in a local network or on the Internet. Not only can individual pages be opened but once a page is opened the user can also follow different links and navigate on the Web.

PAINT

♦ **Syntax**

PAINT INT **x**, INT **y**, INT **col**

♦ **Parameters**

x,y:	Paint coordinates
col:	Color

♦ **Description**

Fills an irregular area with the color **{col}** on the videoscreen. The coordinates for the top left-hand corner of the monitor are expressed in PIXEL and must be contained in the area that is to be colored. The **{col}** parameter is a handle of the table *Standard event colors*, in which the colors used in this instruction are defined.

RGB and RGB_A macros can be used as parameter **{col}** parameters

PASSWORD WND

♦ **Syntax**

PASSWORD WND INT **op**, VAR INT **lev**, VAR STR **uid**, [, INT **flg**]

♦ **Parameters**

op: (0 - 8) Check level (0 = all)
lev:1 - 8 Password level 0 = Deleted operation
uid: Name of selected user (if lev > 0)
flg: **1** = checks level of current password:
if level >= op it doesn’t open passwords dialog.

♦ **Description**

Opens the window requesting verification or modification of a password. If the password is incorrect an error message is displayed. If **{flg}** is 1 a check is made to discover whether default password is >= **{op}**. If it is not the passwords window is opened. In other words, if **{flg}** = 1 the passwords window is opened only in an emergency whereas **{flg}** = 0 always opens a window.

♦ **Return value**

Result of operation in parameter **{lev}:**

0 = deleted operation
1-8 = password level entered

PRINT FORM

♦ **Syntax**

PRINT FORM INT **form**

♦ **Parameters**

form: Number of print form

♦ **Description**

Sends the printer the printout form **{form}** number. This number is configured in the print forms window.

N.B. This instruction does not print the form but prepares it and places it in a private queue. It does not print the form but prepares it and places it in a private queue. In the forms configuration window the number of pages can be specified that the queue may contain. Once this limit is reached, Logoview NT will print all the stored pages. The queue can also be printed immediately if the command **FLUSH FORM** is used. This forces Logoview NT to print all the pages that have been stored but which have not yet been printed.

READ PLC

♦ **Syntax**

READ PLC STR **plc**, STR **node**, VAR **dest**, INT **nvar**, INT **rem**, INT **tout**, VAR INT **ris** [,INT **flg**]

♦ **Parameters**

plc: Name of driver station from which to carry out reading.
node: Node of network hosting the driver station “” if local
dest: Destination variable
nvar: Number of variables
rem: Remote offset
tout: Timeout in seconds
ris: Result of reading
flg: if current and =

0 = If there are other requests during the timeout period,
Updates only the variables that have changed.
1 = Returns immediately if there are other requests.
Updates only the variables that have changed.
2 = Waits if there are other requests during the timeout
period. Updates all variables.
3 = Returns immediately if there are other requests.
Updates all variables.

♦ **Description**

Reads the station **{plc}** on the **{node}** of **{nvar}** variables from the remote variable **{rem}** and places the result in the local block of variables starting with the variable **{dest}**. The function waits for the result of the reading for **{tout}** tenths of a second. If the parameter is current **{flg}** the request is not passed on if others are waiting. The **{ris}** variable signal is given is the reading has been successful. If a server is used that is already being used by other clients, it is advisable to update (**{flg}** = 2 or **{flg}** = 3).

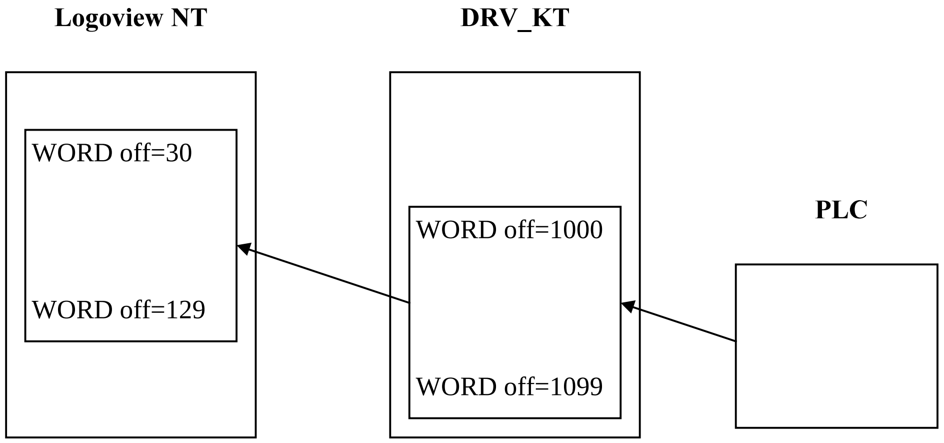
♦ **Return value**

Result of operation in **{ris}** parameter**:**

0= Ok
10 = Unknown server (check name in the window of the communications)
11= Command not supported by the specified server
12 = Channel occupied (with **{flag}** = 1 or **{flag}** = 3)
13 = Timeout
14 = Unknown error
16 = Closed channel

♦ **Example**

We wish to read 100 WORD variables starting from remote offset 1000 on “DRV_KT” driver in the local node(i.e. on the same Logoview NT host machine). These variables need to copied onto 100 local WORD variables starting with variable 30, according to this plan:



READ PLC “DRV_KT”,””, 30, 100, 1000, 10 ,#ERR

READ REC

♦Syntax

READ REC INT **arc**, VAR **st**, VAR **byte**, VAR **word**, VAR **trig**, VAR **real**, VAR **dword**

♦Parameters

arc:	File number
st:	Result of operation
byte:	Destination of BYTE variables
word:	Destination of WORD variables
trig:	Destination of TRIGGER variables
real:	Destination of REAL variables
dword:	Destination of DWORD variables

♦Description

Reads the file records in sequence **{arc}**; the fields are stored starting with the variables **{byte, word,..}**; this is because there may be variables of all types in the record and Logoview NT must know the exact location of variables read by the disk and must specify the address for each type.
The BYTE variables are thus located starting with the address of the **{byte}** variable; the WORD variables starting with the address of **{word}** and so on, even for TRIGGER. If a certain type of variable has not been used to define the record any parameter can be specified as it will not be used.
As for other functions that refer to the files this function also requires that the files be defined before they are used. To do so, access the *section History Files* in the window Configuring Formats.

For methods of describing file characteristics and creating the record see the description in chapter 9, 'Forms' of Logoview NT PRIMER.

The time and date of the read record are stored in the system variables: **R_HOUR**, **R_MINUTES**, **R_SECONDS**, **R_DAY**, **R_MONTH**,**R_YEAR**.

♦Return value

Result of operation in the parameter **{st}:**

0=	Ok there are still records	
1:	=	Last record read
2:	=	Reading error beyond EOF
3:	=	Read record does not contain data
-1:	=	Error during file opening
-2:	=	Error during file creation
-3:	=	Error on file number
-4:	=	Error, writing (disk full?)
-5:	=	Error, reading
-6:	=	No memory for file field operations
-7:	=	Variable does not exist
-8:	=	Incorrect header
-9:	=	Incorrect version number
-10:	=	READ_ONLY file open

♦ Example

We wish to read a record from the file “ALARM.ARC” and to display the message found.

```
Event 10:
SEEK REC 1,#TRIG, DAY, MONTH, YEAR, TIME,
MINUTES SECONDS
READ REC  1,#TRIG,#STATUS
RETURN
```

RESET BOX

♦ **Syntax**

RESET BOX

♦ **Description**

Deactivates all reference boxes. After this function is run all subsequent graphic operations will refer to absolute coordinates, i.e. the coordinates referring to the top left-hand corner of the screen.

RESET SCREEN

♦ **Syntax**
RESET SCREEN

♦ **Parameters**
none

♦ **Description**

Eliminates the effect of the **SET SCREEN**; like the **SET SCREEN** this instruction is cumulative. If it is invoked several times the effect of a multiple **SET SCREEN** can be eliminated.

RESET SEM

♦**Syntax**

RESET SEM INT **s**

♦**Parameters**

s: Semaphore number (0 - 31)

♦**Description**

Zeroes semaphore t **{s}** by starting up again TASKS on this semaphore .

This function complements **SET SEM** and enables a task to run concurrently within the framework of the instructions comprised between a **SET SEM** and a **RESET SEM.**

♦**Example**

```
Event 0:
SET SEM SEMAPHORE 1
LEGGI PLC “DRV_KT”,””, 30, 100, 1000, 10 ,#ERR
RESET SEM SEMAPHORE 1
FINE
```

RESET VAR

♦ **Syntax**

RESET VAR VAR **first**, INT **num [**, **number**, INT **flag]**

♦ **Parameters**

first: First variable to zero
num: Number of variables to zero
number: Initialization value of variables (0 default) must be compatible with first type of variable.
flag: 0 = or absent does not update frames
1 = updates any animations on frames

♦ **Description**

The instruction **RESET VAR** initializes the at **{number}** the **{num}** starting with **{first}** variable. The operation is practically instantaneous.

♦ **Example**

Resets a block of 1000 variables trigger (?) at 40, starting with offset 50 variable and updates the currently displayed frames (flg = 1)
RESET VAR #50, 1000, 40, 1

RESTART

♦ **Syntax**

RESTART INT **n**

♦ **Parameters**

No: Process number

♦ **Description**

Restarts processes suspended with STOP from the point in which they were interrupted. If {n} is zero all the suspended processes will restart. On the other hand, if {n} falls between 1 and 8 it only restarts the specified process.

♦ **Example**

```
Event 0:
*SUB 18
•  Starting with event 1 process 2 is initialized.
TASK 2,1
EVENT 1
STOP 2
DEL ALL
•  Process two suspended
WHILE #nm  < 10
   LET #nm=#nm+1
END WHILE
RESTART 2
END
```

RETURN

♦ **Syntax**

RETURN [**esp**]

♦ **Description**

Returns from an event call with instruction SUB or CALL. This function may generate errors when the program flow meets it before a jump has been made using the instruction SUB or CALL. The **RETURN** can return a result. If the event has been called up by a CALL the result can be used by the caller. Otherwise, it will be lost. The result is optional because returns 0 (numeric result) or string 0 (string result) are not specified. Specially attention must be devoted to the type of returned result because if the caller expects a number and a string is returned (or vice versa) unexpected results may occur .

An event must always finish with an **END** or a **RETURN**; these instructions are interchangeable.

SAVE VAR

♦ **Syntax**

SAVE VAR STR **file**, VAR INT **err**, VAR **start**, INT **nvar**, INT **flg**

♦ **Parameters**

file: Name file
err: Error code
start: First variable to be saved
nvar: Number of variables
flg: = **1** attach to existing file
= **2** save instead of the name of the offset of the variable

♦ **Description**

Salves a group of variables in a file. If the complete file path is not indicated it will be created inside the application directory. If **{flg}** = 2 Logoview NT is specified it will proceed to save the offset of the variables in the file instead of the name. This speeds up the loading operation (LOAD VAR).

♦ **Return value**

Result of operation in **{status}** parameter**:**

0	=	OK
-1	=	File opening error
-2	=	File writing error

###

SCREEN

♦ **Syntax**

SCREEN STR **name**, STR **title** [, INT **replace**]

♦ **Parameters**

name: Name of file to be loaded

title: Title of new frame

replace: If current and = **1** replaces the frame in the active window without opening another

♦ **Description**

Loads the frame defined by the string **{name}** onto the screen.

When this instruction is run the frame indicated by the string **{name}** is loaded onto the videoscreen and displayed. If the parameter **{replace}** is current and = 1 the frame will replace the frame current in the active window without opening another. Otherwise, the frame will be loaded into a new window if it is possible to create one.

This instruction can be used if a trend window is to be opened. To do so, it is sufficient to specify the title of the trend in **{name}** For the trends **{title}** is not used because the window already has its own name. The trends are always opened in a separate window.

SCREEN NAME

♦ **Syntax**

SCREEN NAME STR **name**

♦ **Parameters**

name: name of frame

♦ **Description**

Returns name of current frame.

♦ **Return value**

Result of operation in **{name}** parameter. As in the **CH...** functions this function must also be used in an event that is linked to a frame.

SCROLL WINDOW

♦ **Syntax**

SCROLL WINDOW INT **hwnd**, INT **posx**, INT **posy**

♦ **Parameters**

hwnd: Window handle , -1 uses the frame assigned to the event

posx,posy: New coordinates inside the window’s internal area

♦ **Description**

Scrolls the window given as a parameter in order to display the coordinates posx,posy in the top left-hand corner. The aim of this instruction is to simulate the action of the scroll bars of a window. It can be used for programmed piloting of the largest frames of the window in which they frames are displayed. This instruction works only with some types of windows such as Titano frames or masks. It does not function with Trend lines or with other types of window. The parameter **hwnd** is a window handle that is returned by the instruction FIND WINDOW. Alternatively, -1 can be specified in order to modify the frame assigned to the event. In this case, the event must be activated by a graphic element of a frame such as a button if it is to operate correctly.

♦ **Example**

Scrolling down frame “SCR000” to coordinates 200.0:

```
FIND WINDOW “SCR000”,1,:hwnd
SCROLL WINDOW :hwnd,200.0
```

SEEK REC

♦ **Syntax**

SEEK REC INT **arc**, VAR **status**, INT **pos**, INT **flag**

♦ **Parameters**

arc:	File number		
status:	Error code		
pos:	Position inside file		
flag:	**0**	=	Sets next reading
	1	=	Sets next writing

♦ **Description**

This instruction enables the user to position himself precisely inside an file in order to read or write accurately. The **{flag}** parameter enables the reading or writing position to be modified if required. The reading position can be modified in any type of file but the writing position can be modified only in NON-CIRCULAR files and NOT at fixed periods.

N.B. It is not possible to modify the writing pointer in a CIRCULAR file or an AUTOMATIC WRITING file.

♦ **Return value**

Result of operation in **{status}** parameter**:**

0	=	Ok
1	=	position beyond end of file

♦ **Example**

SEEK REC 0, #status, 10533, 0

SET BOX

♦ **Syntax**

SET BOX INT **n**

♦ **Parameters**

n: Number of box to be activated

♦ **Description**

Activates the box defined by **{n}**. From this moment all the coordinates of the graphic instructions refer to the active box.

SET BUTTON

♦ **Syntax**

SET BUTTON INT **nbx,** INT **status**

♦ **Parameters**

nbx: Number of box in which the button is located that is to be activated or deactivated.

status: **1** = Activate, **0** = Deactivate

♦ **Description**

Activates or deactivates the button in the box **{nbx}**. By default, this instruction modifies the frame assigned to the event. The instructions RESET SCREEN and SET SCREEN enable buttons to be activated/deactivated on any open frame. Warning: the parameter $ALL of the SET SCREEN is ignored by the SET BUTTON.

♦ **Example**

Deactivates the button on box 5 of the square “SCR001”; activates the button on box 6 of the square “SCR002”

```
SET SCREEN “SCR001”
SET BUTTON 5,0
RESET SCREEN
SET SCREEN “SCR002”
SET BUTTON 6,1
RESET SCREEN
```

SET SCREEN

♦ **Syntax**

SET SCREEN STR **scr**

♦ **Parameters**

scr: Name of frame or
$ALL = All frames
$NONE = No frame

♦ **Description**

Enables drawing on a specific frame **{scr}**. It is also possible to enable the operation on all frames with **{scr}** =$ALL or none **{scr}** = $NONE. The events generated by an object in the frames already carry out an implicit **SET SCREEN** once they have been run; it is therefore possible to draw from these events without any further operation. On the other hand no frame has been assigned to the events run by a timer or a trigger so that this instruction must be used to set the frame for graphic operations.

N.B. The **SET SCREEN** instruction is cumulative. It can be invoked several times to draw on several different frames. The parameter **{src}** = $ALL can be used to draw on all the open frames. To invalidate the effects of the last **SET SCREEN** it is sufficient to invoke a **RESET SCREEN**.

SET SEM

♦ **Syntax**

SET SEM INT **s**

♦ **Parameters**

s: Semaphore number

♦ **Description**

Check that the semaphore **{s}** is set. In this case the TASK waits until the semaphore **{s}** is reset by another process by the instruction **RESET SEM**. When the semaphore is reset in such a way that the TASK can proceed the instruction **SET SEM** sets the task again in order to block access to other tasks. This instruction defines the critical areas that must be run by only one task at a time. 32 semaphores are available, so **{s}** must be assigned a number between 0 and 31.

Let us suppose that we have an event that manages reading of certain variables by the PLC that must be detected by two distinct tasks. To prevent both tasks from acquiring data simultaneously a flag is positioned (the instruction **SET SEM**) so that the concurrent task is warned that data acquisition is already in progress.

When the event linked to one of the two tasks reaches the instruction **SET SEM {s}** n, semaphores status is tested. If it has not already been set, the instructions can be run further by indicating that processing these data are in progress and that semaphore **{items}** is set.

On the other hand, if the semaphore has already been set, the task goes into wait until the semaphore is reset by the concurrent task. The flag is reset by the instruction **RESET SEM {s}**.

♦ **Example**

```
Event 5:

SET SEM 1
READ PLC “DRV_KT”,””, 30, 100, 1000, 10 ,#ERR
IF #ERR != 0
   *error management
END IF
RESET SEM 1
END
```

SET WINDOWPOS

♦ **Syntax**

SET WINDOWPOS INT **hwnd**, INT **x**, INT **y**, INT **dx**, INT **dy**, INT **flags**

♦ **Parameters**

hwnd: Window handle, -1 use frame assigned to the event
x,y: Window position
dx,dy: Window dimensions
flags: Flags:
1=Ignore position, change only dimensions
2=Ignore dimensions, change only position

♦ **Description**

Modifies position and dimensions of window given as parameter. The position coordinates refer to the area inside the window on which it depends. For example, in Logoview frame windows the frames coordinates refer to the inside area of the main Logoview window. The parameter **hwnd** must be a window handle returned by the instruction FIND WINDOW. Alternatively, the user can specify -1 to modify the frame assigned to the event. In this case, the event must be activated by a graphic element of a frame such as a button if it is to operate correctly.

Example
Positioning “SCR000” at the coordinates 0.0:

```
FIND WINDOW “SCR000”,1,:hwnd
SET WINDOWPOS :hwnd,0.0,0.0,2
```

SHOW WINDOW

♦ **Syntax**

SHOW WINDOW INT **hwnd**, INT **code**

♦ **Parameters**

hwnd: Window handle , -1 use the frame assigned to the event

code: Operation code:

0= Hide. Hides the window .

1= Show Normal. Activates and shows the window. If the window is minimized or maximized the previous position and dimensions are restored.

2= Show Minimized. Activates the window and minimizes it.

3= Show Maximized. Activates the window and maximizes it.

4= Show No Activate. Shows window but does not activate it. the current active window remains active.

5= Show. Activates the window and shows it in its current position.

6= Minimize. Minimizes the window and makes the next window active.

7= Show Minimized no activate. Shows and minimizes the window without modifying the active window.

8= Show No Activate. Shows the window without modifying the active window.

9= Restore. Activates and shows window. If it is minimized or maximized the previous position and dimensions are restored. When a minimized window is restored this flag should be used.

10= Bring window to top. Brings window to top. This flag brings the window to the top but if it is minimized it stays minimized. To reopen it, specify flag **9**.

11= Close Window. Close window by destroying it. To reduce it to an icon, use flag **6**.

12= Disable Window. Disables window to prevent it from accepting input from the user.

13= Enable Window. Enables the window and makes it available for the user.

♦ **Description**

Changes window status according to value of parameter flags. This instruction can, for example, be used to reduce a window to an icon or to maximize it. It can also be used to close a window (parameter **11**) as an alternative to CLOSE SCREEN. The parameter **hwnd** must be a window handle that is returned by the instruction FIND WINDOW. Alternatively, -1 can be specified to modify the frame assigned to the event: in this case, the event must be activated by a graphic element of a frame such as a button if it is to operate correctly.

♦ **Example**

Find main Logoview window and reduce it to an icon:

```
FIND WINDOW “Logoview –“,2,:hwnd
SHOW WINDOW :hwnd,6
```

SHUTDOWN

♦ **Syntax**

SHUTDOWN INT **type**

♦ **Parameters**

type: Type Shutdown :
0 = Close Logoview
1 = Close Logoview and shut down windows

♦ **Description**

Closes current application and any files that may be opened.

SOLID BOX

♦ **Syntax**

SOLID BOX INT **x1**, INT **y1**, INT **x2**, INT **y2**, INT **col**

♦ **Parameters**

x1,y1,x2,y2: Rectangle coordinates
col: Color of edge and area of rectangle

♦ **Description**

Traces a box that is completely colored **{col}** on the videoscreen. The coordinates of the top left-hand corner of the monitor are expressed in PIXEL. The number of the color is configured in the *Standard event colors* table.
Macros RGB and RGB_A can also be used as parameters {col} for this function.

SOLID ELLIPSE

♦ **Syntax**

SOLID ELLIPSE INT **xc**, INT **yc**, INT **rx**, INT **ry**, INT **col**

♦ **Parameters**

xc,yc: Center coordinates
rx,ry: Ellipse axes
col: Color

♦ **Description**

Traces a solid color ellipse **{col}**, center **{xc,yc}** and axes **{rx,ry}.**
The angle coordinates are in the top left hand corner of the monitor and are expressed in PIXEL. The color **{col}** is defined in the *Standard event colors table.*
Macros RGB and RGB_A can be used as parameters {col}.

STOP

♦ **Syntax**

STOP INT **n**

♦ **Parameters**

n: Process number to be suspended

♦ **Description**

If **{n}** is zero all the processes are suspended except the one that has actually invoked the instruction. If **{n}** is between 1 and 8 only the specified process is suspended. The suspended processes can be restored by RESTART. Note that the instructions **STOP**, **RESTART** can only suspend user processes, i.e. those from 1 to 8. To suspend trigger or timed processes use **LOCK** and **UNLOCK**.

STRCASE

♦ **Syntax**

STRCASE VAR STRING **dest**, INT **case**

♦ **Parameters**

dest:	destination string
case:	0=Uppercase 1=Lowercase

♦ **Description**

Transforms the string **{dest}** into upper case (**{case}**=0) or lower case (**{case}**=1).

STRCAT

♦ Syntax

STRCAT VAR STRING **dest**, STR **src**, [,INT **nchr**]

♦ Parameters

dest: Destination variable to which to chain the string
src: String to chain
nchr: Optional number of src characters

♦ Description

Chains the string **{src}** to variable string **{dest}**.
If the parameter **{nchr}** is current specify the number of characters to take from the string **{src}**.

STRDATE

♦ **Syntax**

STRDATE VAR STRING **dest**, INT **day**, INT **month**, INT **year**

♦ **Parameters**

dest:	Destination
day:	Day of formatting date
month:	Month of formatting date
year:	Year of formatting date

♦ **Description**

Formats a date with the format set by default in Windows and stores it in **{dest}**.

STREXTR

♦ Syntax

STREXTR VAR STRING **dest**, STR **src**, INT **first**, INT **nchr**

♦ Parameters

dest: Variable in which extracted characters are entered
src: String from which characters are extracted
first: First src character from which extraction starts
nchr: No of characters to be extracted

♦ Description

Copies characters of the string **{src}** starting with the character **{first}**.to **{dest}** **{nchr}**

STRPRINT

♦ **Syntax**

STRPRINT VAR STR **dest**, STR **format** [, PAR **par1**, ... , PAR **par16**]

♦ **Parameters**

dest: Destination string
format: Formatting string
par1: First optional parameter
:
par16: Sixteenth optional parameter

♦ **Description**

This instruction enables a printout that is formatted as a string. It is like a **DISPLAY** in which the result is stored in a variable rather than appearing on the screen. The formatting codes of the parameters are the same as **DISPLAY**.

STRREPEAT

♦ **Syntax**

STRREPEAT VAR STRING **dest**, STR **src** ,INT **rep**

♦ **Parameters**

dest:	Destination string
src:	Source string
rep:	Number of repetitions

♦ **Description**

Duplicates **{rep}** times the string **{src}** and stores it in **{dest}**. For example, **STRREPEAT** $A,"AB",4 stores in $A la string "ABABABAB".

SUB

♦ **Syntax**

SUB INT **e** [, PAR **par1**, ... , PAR **par15**]

♦ **Parameters**

e:	Event number
par1:	Variable or value given as parameter value
:	
par15:	Variable or value given as parameter value

♦ **Description**

Runs a subroutine starting from the event **{e}**. Also in this case the event number **{e}** can be replaced by a variable that contains the number. In this case the value of the variable must be kept under control that gives the number of the event to be run because if a value is outside the range of the defined events it could cause the system to crash if it is run. When an event with the **SUB** is run if the called up event ends with the instruction **RETURN**, at the end of the event the program flow returns to the instruction following the calling sub. Otherwise, the flow continues until it finds an **END** or a **RETURN**.

Be careful that no orphan calls or made, i.e. calls without the corresponding **RETURN**. Otherwise a system emergency may arise due to lack of stack. The maximum number of calls of the nested **SUB** function is 50.

TAB EDIT

♦ **Syntax**

TAB EDIT STR **table**, INT **line** , INT **column**

♦ **Parameters**

table:	Name of table in current file
line :	Line containing editing box
column :	Column containing editing box

♦ **Description**

Enables editing of a specific box of a table to be edited.

This instruction can be run only by events to which a frame has been assigned that are run by, for example, buttons, animations and interactive fields.

If no frame has been assigned (as in the case of events run by timer) the function returns an error. This behavior is identical to that of **CH...** functions.

TASK

♦ **Syntax**

TASK INT **n**, INT **e**

♦ **Parameters**

n: Process number (2-8)
e: Event number

♦ **Description**

This instruction runs the event **{e}** as a parallel process, characterized by the number **{n}**. Logoview NT can manage up to 8 processes in parallel that have been activated by the flow of events. A process is a set of programming sequences that can be run in parallel to others. This enables more than one flow of Logoview NT instructions can be run inside the same application.

♦ **Example**

Event 0:
**From event 1 process 2 is initialized.*
TASK 2,1

TIMER

♦ Syntax

TIMER INT **flg**, INT **tmr**, VAR REAL **cont**

♦ Parameters

flg: = **0** Starts up time meter
= **1** Stops meter and returns value {cont}
tmr: timer index (0-31, or -1)
cont: Contains recorded time.

♦ Description

This instruction enables partial time to be counted. If this instruction is first run with **{flg}**=0 the count is initialized. If the user wishes to know how much time has elapsed from when the zeroing instruction was started he moves onto instruction **{flg}**=1 and a REAL type variable so that Logoview NT can indicate the time. Once the meter has been zeroed it continues to accumulate seconds. It is therefor possible to have more partial time by repeatedly calling up the **TIMER** with **{flg}**=1.
32 (0-31) timers are available plus a timer for each TASK (**{tmr}**=-1) and the time returned in **{cont}** is that of the meter assigned to the requested timer.

♦ Example

```
LET  #key=0
TIMER 0,@TMP
WHILE  (#key < 100)
   LET #key = #key + 1
END WHILE
TIMER 1,@TMP
•  in TMP is the time taken to complete the loop
END
```

TRIGGER

♦ **Syntax**

TRIGGER VAR **trig**, INT **v**

♦ **Parameters**

trig:	Trigger variable
v:	Trigger status

♦ **Description**

Modifies the status of the trigger variable **{trig}**; if **{trig}** is different from **TRIGGER** during runtime an error is reported. Parameter **{v}** is an INT-type expression that is always changed into a binary value (0.1). In addition, if the status of variable **{trig}** is changed (i.e. from 0 to 1, or from 1 to 0) the corresponding event is started.

♦ **Example**

Modifies the trigger variable of offset 12 with value 1.

TRIGGER #12,1

UNLOCK

♦ **Syntax**

UNLOCK

♦ **Description**

If the **UNLOCK** function is used it defines the end of a sequence of instructions within an event that make up a critical point in running an application.

When this instruction is run the concurrent processes will be run that were previously suspended by the instruction **LOCK** and all the events that make it up.

WAIT

♦ **Syntax**

WAIT INT **sec**

♦ **Parameters**

sec: Number of tenths of a second

♦ **Description**

Suspends the process that calls up the function for **{sec}** tenths of a second. This function is especially useful when the interspacing of different event instructions has to be slowed down.

Both the timed events and the triggered events work in the multitask mode and are therefore run in parallel to the main program. Using one of these functions in one of the processes does not interfere in any way with the other programs that are running concurrently and which may be in the process of being processed.

WHILE

♦ **Syntax**

WHILE INT **cond**

♦ **Parameters**

cond: numeric expression

♦ **Description**

Repeats the instructions between **WHILE** and **END WHILE** until the expression **{cond}** is true. This is a WHILE loop in all ways and any instruction can be inserted in it.
The length of the jump between the start of the loop and the end is limited by the maximum length of the event. The status may be single or multiple. There are no great problems, except for syntactic problems, in the case of a single condition. On the other hand, for multiple conditions the single statuses must be enclosed between parentheses. For example:

WHILE ((#VAR134)&(!TR1=1))|(!TR2=0)

In this case the event will continue to revolve inside the loop until byte **!TR2** is equal to zero or word **VAR1** is greater than **34** and byte **!TR1** is equal to **1**.
Closing the loop must be indicated by the instruction **END WHILE**.
Up to **50** other loops can be nested within the loop . If the expression at the start of the loop is false the loop is **not** run.

Example

We increase variable x to 100 and decrease it to 0°.

```
Event 10
...
LET  #x=0
*increase
WHILE  #x <= 100
   LET  #x = #x + 1
END WHILE
*decrease
WHILE  #x > 0
   LET  #x = #x - 1
END WHILE
...
RETURN
```

WINHELP

♦ **Syntax**

WINHELP STR **hlpfile**, INT **cmd**, INT **data**

♦ **Parameters**

hlpfile: Help file name. If it does not contain the path the path in which the application is located will be taken as a default value.

Cmd: Help command

0 : Opens itemspecified by ID given in <data>
1 : Opens itemspecified by ID given in <data> as POPUP window
2 : Opens contents table of Help file
3 : Opens main page of Help file

Data: Help data

♦ **Description**

Opens Help file **{hlpfile}** using Windows Help system. To display a item inside the Help file use **cmd=0** and specify the numeric ID of the requested item in **date.** The numeric ID must be defined during the Help writing phase, otherwise the required item will not be called up.

WRITE PLC

♦ Syntax

WRITE PLC STR **plc**, STR **node**, VAR **source**, INT **nvar**, INT **rem**, INT **tout**, VAR INT **ris** [,INT **flg**]

♦ Parameters

plc:	Name of driver station from which to run reading
node:	Node of network hosting driver station
source:	Variable source
nvar:	Number of variables
rem:	Remote offset
tout:	Timeout in seconds
ris:	Result of reading
flg:	= **0** or absent, waiting for Timeout = **1** return immediately if there are other messages pending.

♦ Description

Writes **{nvar}** starting with variable **{org}** on the **{plc}** station located in the **{node}** starting with the remote variable **{rem}**. The function waits for the result of the reading for **{tout}** tenths of a second. If the parameter **{flg}** is current the request is not forwarded if other requests are waiting. The variable **{ris}** reports whether the reading has been successful.

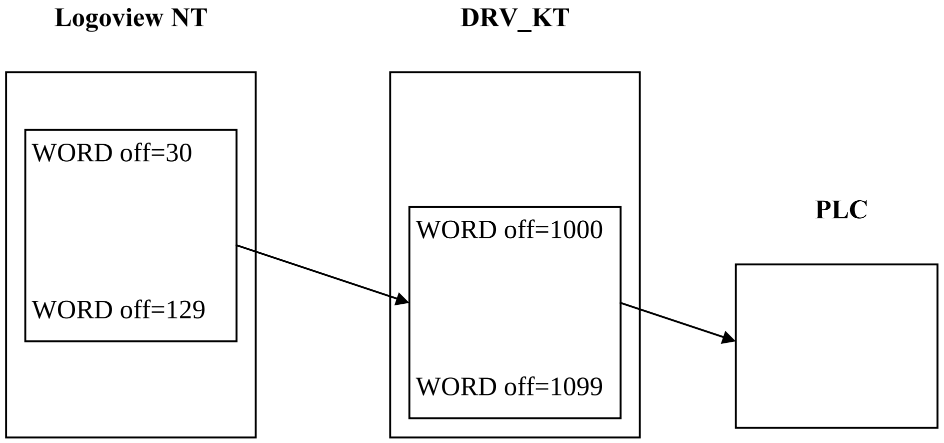
♦ Return value

Result of operation in parameter **{ris}:**

0= Ok
10 = Server unknown (check the name of the communications window)
11= Command not supported by specified server
12 = Occupied channel (with **{flag}** = 1 or **{flag}** = 3)
13 = Timeout
14 = Unknown error
16 = Closed channel

♦ Example

We wish to write a block of 100 WORD variables starting from local offset 1000 on “DRV_KT” driver in the local node (i.e. on the same Logoview NT host machine). These variables need to be written starting with the 100° remote offset, according to this plan:



WRITE PLC “DRV_KT”,””, 30, 100, 1000, 10 ,#ERR

WRITE REC

♦ **Syntax**

WRITE REC INT **ar**, VAR **err**

♦ **Parameters**

ar: File number
err: Error code

♦ **Description**

Tags on a record to the file assigned to the file **{ar}**.

As for other functions that refer to the files this also requires the files to be defined before they are used.

To do so, it is sufficient to accede to the *History Files section* of the *Formats Configuration window.*

For details on methods of describing file characteristics and creating the record, see the description in chapter 9:'Formats' of Logoview NT PRIMER.

♦ **Return value**

Result of operation in parameter **{err} :**

0 Ok		
-1	=	Error during file opening
-2	=	Error during file creation
-3	=	Error on format number
-4	=	Error, writing (full disk, etc.)
-5	=	Error in reading
-6	=	No memory for inf. in the file fields
-7	=	Variable does not exist
-8	=	Incorrect header
-9	=	Incorrect version number
-10	=	READ_ONLY file open

♦ **Example:**

During a trigger set or reset linked to an alarm we wish to write a record in the ALARMS.ARC file and test: if necessary display alarm.

```
Event 115:
WRITE REC.  1
IF  #VIS=1
    IF  #ORDIN=1
      LET  #Y=461
    END IF
    * displays alarm ...
END IF
RETURN
```

Reference alarms

Limits

These are the limits of the database section of Logoview NT:

Description	Limit
Data File Dimensions (DBF)	1.000.000.000 bytes
Max. width: field	254 characters
No of numeric field digits	19
Number of fields	256
Number of di tags by index	47
Record dimensions	65500 (64k)

Return codes

The following codes can be returned by the Database functions:

Value	Meaning
0	In general, a 0 result indicates that the instruction is successful.
1	Search key has been found.
2	Search has been unsuccessful and database index is positioned on the record after the one sought.
3	End of file (EOF)
4	Beginning of file (BOF)
5	Key has not been found
10	Tag should be in reverse order
20	Tag should have single keys or attempt has been made to add a duplicate key.
50	Part of the file has been LOCKED by another user.
60	File cannot be created
70	File cannot be opened
80	Tag has not been found
110	There are no active transactions
120	There are active transactions
140	No authorization for requested operation
150	Activated connection already exists
170	Attempt to create log file with already existing log file
180	No logging active

Error codes

The following error codes are returned by the database functions:

General errors during access to disk

Value	Meaning
-10	Error during file closing
-20	Error in creation of file. For example, an incorrect name has been specified, the file was already open or there are problems with the disk.
-30	Error in determining the length of a file.
-40	Error during modification of file length.
-50	Error during locking of a file

-60	*Error in opening of a file. The most common problem is that an attempt is made to open a file that does not exist. Another problem is the attempt to open more files than those that have been configured in CONFIG.SYS (Logoview allows up to 64 files to be open, 40 of which are database).*
-61	*It is not possible to open the file (access denied)*
-62	*Opening of file not valid*
-63	*Maximum number of open file handles has been reached*
-64	*Specified file has not been found*
-69	*Attempt to open a second file instance not authorized*
-70	*Error during file reading. It could be caused by a* ***GO*** *database on a non-existent record.*
-80	*Error during file deletion*
-90	*Error during file renaming*
-100	*Error during positioning in a file point*
-110	*Error during* unlocking *of a file.*
-120	*Error during writing of a file. Occurs when disc is full.*

Specific errors on DBFs

Value	*Meaning*
-200	*The specified file isn't a DBF. It occurs only when the 'header (and probably the data) of the DBF file has been damaged.*
-210	*Incorrect field name. The field name given to an instruction like the GETFIELD database or the PUTFIELD database does not exist inside the database .*
-220	*Type of field not recognized..*
-230	*Found record is too large.*
-240	*Attempt was made to add a record beyond the end of the file*
-250	*Attempt was made to seek a non-numeric tag*

Specific errors on index files (.CDX)

Value	*Meaning*
-300	*No key was found in the tag. This error occurred when a key corresponding to a record should be a tag but was not found.*
-310	*The index file is damaged. This error is generated when an inconsistency is discovered inside a CDX. file.*
-330	*Name of incorrect TAG. The name of an incorrect tag has been specified in a TAGSELECT database instruction. Check that the index file with the tag to be selected has been opened.*
-340	*Attempt to add a record or create an index file has duplicated a tag.*
-350	*Information in tag structure incorrect.*

Errors in evaluation of database expressions

Value	*Meaning*
-400	*Comma "," or closed parenthesis ")" was expected. For example, the expression "SUBSTR(A" seems to have generated this type of error.*
-410	*The expression is not complete. For example, the expression "FIELD_A+" would not be complete because something is missing after "+".*
-420	*The specified file date was not opened.*
-422	*The second and third paragraphs of an* ***IIF()*** *must have exactly the same length.*
-425	*The second and third paragraphs of an* ***SUBSTR()*** *or* ***STR()*** *must be constants.*
-430	*The number of parameters is incorrect.*
-440	*The expression is too long or complicated to be evaluated.*

-450	Closed parenthesis ")" missing. Be careful that each open parenthesis is matched by a closed parenthesis.
-460	A sub-expression type does not match the type requested by an operator. For example, in the expression "33 .AND .F.", "33" is numeric whereas the operator ".AND" requires a logic type..
-470	Function not recognized. Name of non-existent function has been found.
-480	Operator not recognized. Incorrect character found instead of operator.
-490	A sequence of characters has not been recognized as being a valid constant, field name or function.
-500	String of terminator character missing '.
-510	Expression invalid for use in a tag.

Optimization errors

Value	Meaning
-610	Error, general, during optimization routine
-620	Error during suspension of optimization
-630	Error during flushing of buffered information

Errors in relations

Value	Meaning
-710	Error, general, in relations routines
-720	No connected record has been found
-730	Referenced relation does not exist or has not been initialized.

Serious errors

Value	Meaning
-910	Internal error. Internal error contains incorrect value.
-920	No more memory available.
-930	Incorrect parameter has been given to a function.
-935	Unexpected input parameter
-950	Unexpected result. Function has returned unexpected error.
-960	Failure during structure control
-970	Structure of internal data damaged or not initialized

Errors in instruction parameters

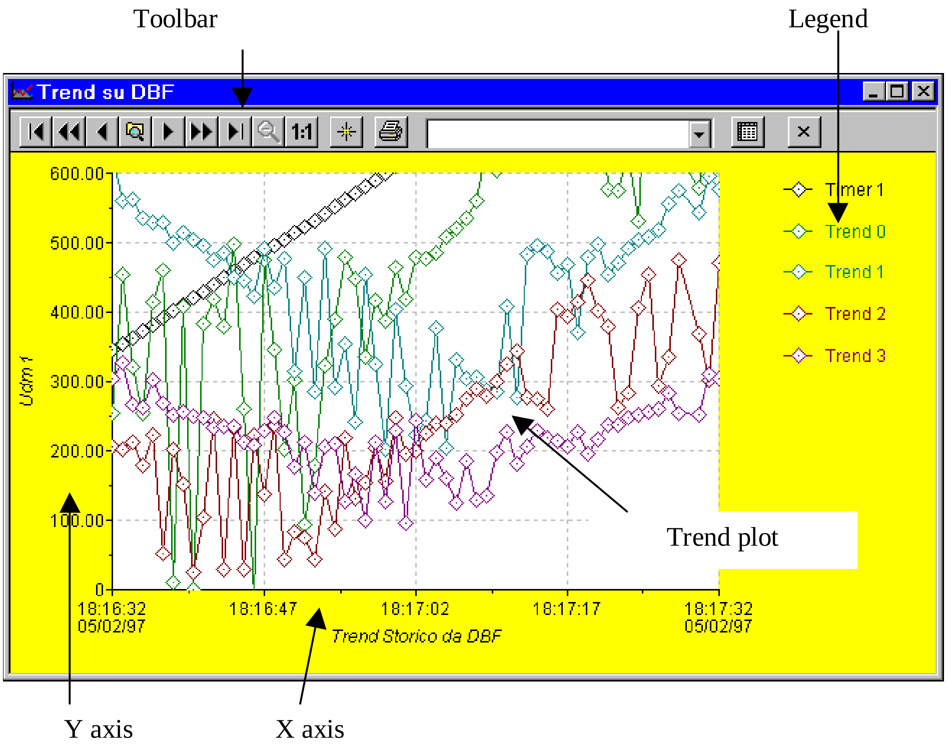
Value	Meaning
-16385	Incorrect description index. In a CREATE or OPEN database a description record index has been specified that does not exist.
-16386	No record. In a CREATE or OPEN database a description index of a record has been specified that contains no field.
-16387	No more handles available (maximum number of handles =40)
-16388	Incorrect handle . An instruction has been given a handle that does not refer to a database or to an open index file.
-16389	Incorrect field name. An instruction has been given a field name that does not exist.
-16390	Incorrect tag name. An instruction has been given a tag name that is not one of the open index files. Make sure that the index file with the required tag has been opened.
-16391	QUERYSET database has not been called up. Before calling up one of the QUERY databases, call up QUERYSET database.

CONTENTS

1.1 *"RUNTIME"* and Historical Trends

This chapter explains in greater detail how historical line trends are displayed and how their configurations are modified during runtime.

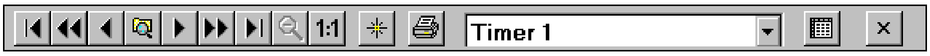
A historical trend is a graph made up of a series of values memorized in files or the database. During runtime the window display therefore comprises parts that were already displayed during the development phase inside the window activated by the *Modify* command, Chapter **Errore. L'origine riferimento non è stata trovata.** and also a different series of commands and options from those of a real time line trend.



The configuration of some trend characteristics can be modified directly during runtime by all operators but if the application has been protected the variables channels can be selected only by those holders of passwords that allow them to make modifications.

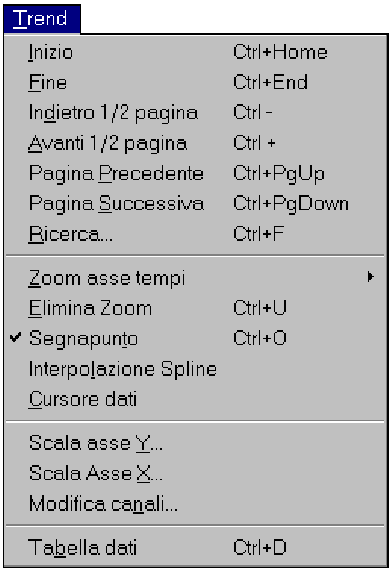
During runtime the line trend is provided with a command menu and a series of icons inside the toolbar that enable the required modifications to trend configuration and control to be made.

1.1.1 Toolbar



The historical trends toolbar consists of icons inside the trends tool bar in real time (Chapter 1.2.1). On the other hand, other icons are specific and have a matching command in the relative menu.
The following chapter explains both the use of the command and of the corresponding icon that can be activated by the mouse cursor.

1.1.2 TREND menu



1.1.2.1 *Start* ⏮

Run this command to place the trend in its starting position.

1.1.2.2 *Finish* ⏭

Run this command to place the trend in its final position.

1.1.2.3 *Back by half a page* ◀

Run this command to place the trend half a page back inside the display window.

1.1.2.4 *Forward by half a page* ▶

Run this command to place the trend half a page forward inside the display window .

1.1.2.5 *Previous page* ⏪

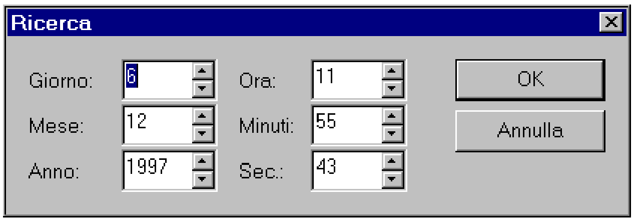
Run this command to place the trend on the previous page inside the display window .

1.1.2.6 *Next page* ⏩

Run this command to place the trend on the next page inside the display window .

1.1.2.7 *Search* 🔍

Run this command to open the dialog window into which to enter the data to be sought inside the trend.



Ricerca

Giorno: 6
Mese: 12
Anno: 1997
Ora: 11
Minuti: 55
Sec.: 43

1.1.2.8 *Time axis zoom*

Run this command to open a cascade menu from which to set the percentage by which the display of trend plot area should be magnified.

1.1.2.9 *Eliminate zoom* 1:1

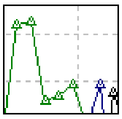
Run this command to eliminate zoom and return to an actual-size 1:1 scale display of the trend plot.

1.1.2.10 *Restore Zoom*

Run this command to restore the previous zoomed display of a trend plot area. If the command is used after the zoom has been restored, the display will return to the 1:1 scale display.

1.1.2.11 *Point marker*

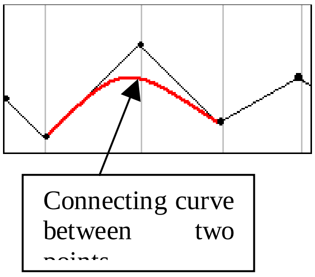
Run this command to display the different trend channels and the point markers of recorded variable values.



1.1.2.12 *Spline interpolation*

If this command is run the graph points inside the trend window are not joined together by a straight line.

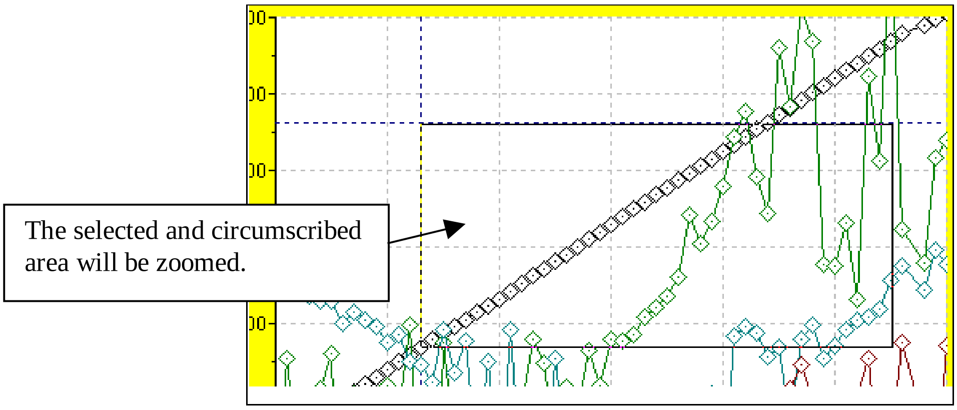
LOGOVIEW NT uses a quadratic equation (second degree equation) to calculate the interpolation of two and plot the ideal connecting curve.



1.1.2.13 *Data cursor*

Run this command to display the data cursor on the trend plot area in order to more effectively highlight the relevant points.

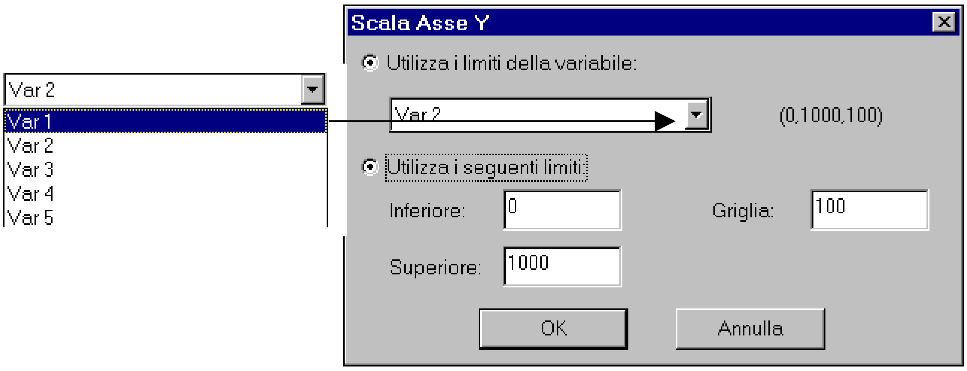
To activate the zoom by means of the data cursor, place the mouse cursor on the relevant plot area mark out the area to be zoomed with the cursor and then press the left-hand mouse key.



The selected and circumscribed area will be zoomed.

1.1.2.14 *Y axis scale*

Run this command to open the dialog window inside which the Y axis scale configuration can be modified.



Scala Asse Y

● Utilizza i limiti della variabile:

Var 2 (0,1000,100)

● Utilizza i seguenti limiti:

Inferiore: 0

Griglia: 100

Superiore: 1000

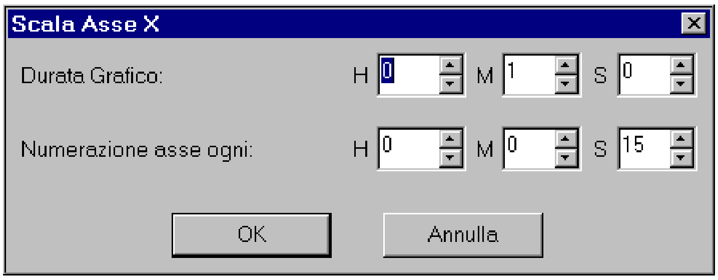
The window is subdivided into two areas from which to select the types of limit to assign to the Y-axis scale.

Use variable limits: use this option to select from a curtain menu the variable channel to be used as a scale for the Y axis.

Use following limits: select this option to key in the scale limits and the value to be assigned to the Y-axis grid.

1.1.2.15 *X axis scale*

Run this command to open the dialog window inside which the X axis scale configuration can be modified.



Scala Asse X

Durata Grafico: H 0 M 1 S 0

Numerazione asse ogni: H 0 M 0 S 15

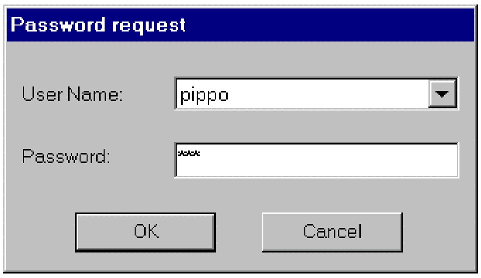
The window is subdivided into two areas from which to select the types of limit to assign to the X-axis scale..

Graph duration: inside these boxes set the duration of the graph that displays the sampled trend data. The duration is expressed in hours, minutes and seconds.

Number axes every: inside these boxes the division of the scale of the X axis is set in accordance with the data sampled by the trend. The division is expressed in hours, minutes and seconds.

1.1.2.16 *Modify channels*

Run this command to open a dialog window in which to modify configuration of the channels of the variables monitored in the trend.
If the trend has been protected, when the command is activated, a window opens that requests the operator’s password.

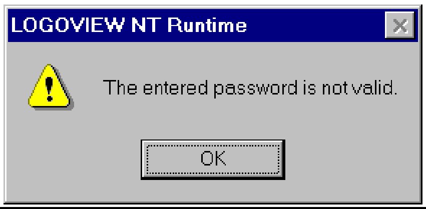


Password request

User Name: pippo

Password: ****

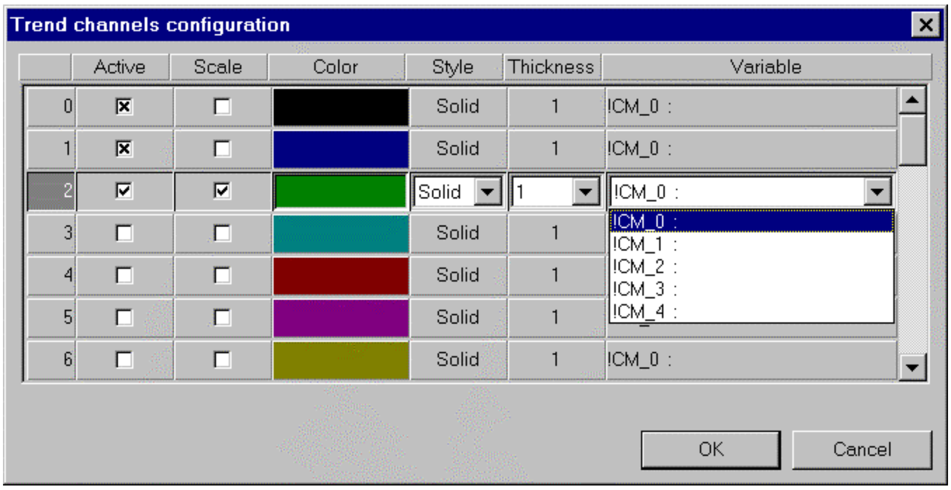
*If the operator keys in an incorrect password **LOGOVIEW NT** displays the following error message.*



LOGOVIEW NT Runtime

The entered password is not valid.

After the correct password has been keyed in the dialog window *“Trend Channel configuration"* is displayed, inside which the configuration of the trend channels for the monitored variables can be modified.



Trend channels configuration

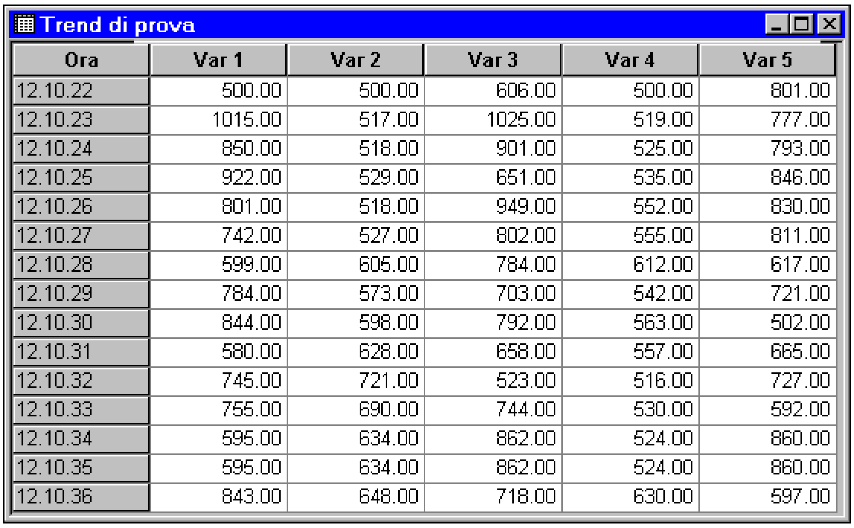
	Active	Scale	Color	Style	Thickness	Variable
0	☒	□	⬛	Solid	1	!CM_0 :
1	☒	□	🟦	Solid	1	!CM_0 :
2	☑	☑	🟩	Solid	1	!CM_0 :
3	□	□	🟦	Solid	1	
4	□	□	🟥	Solid	1	
5	□	□	🟪	Solid	1	
6	□	□	🟨	Solid	1	!CM_0 :

!CM_0 :
!CM_1 :
!CM_2 :
!CM_3 :
!CM_4 :

For full details on the configuration window, see Chapter **Errore. L'origine riferimento non è stata trovata.** *Trend Channel configuration.*

1.1.2.17 *Data table*

Run this command to display all the channel values that make up the trend. These values cannot be altered: they are only for consultation. To modify them, copy them onto the system's clipboard and transfer them to other applications such as Excel.



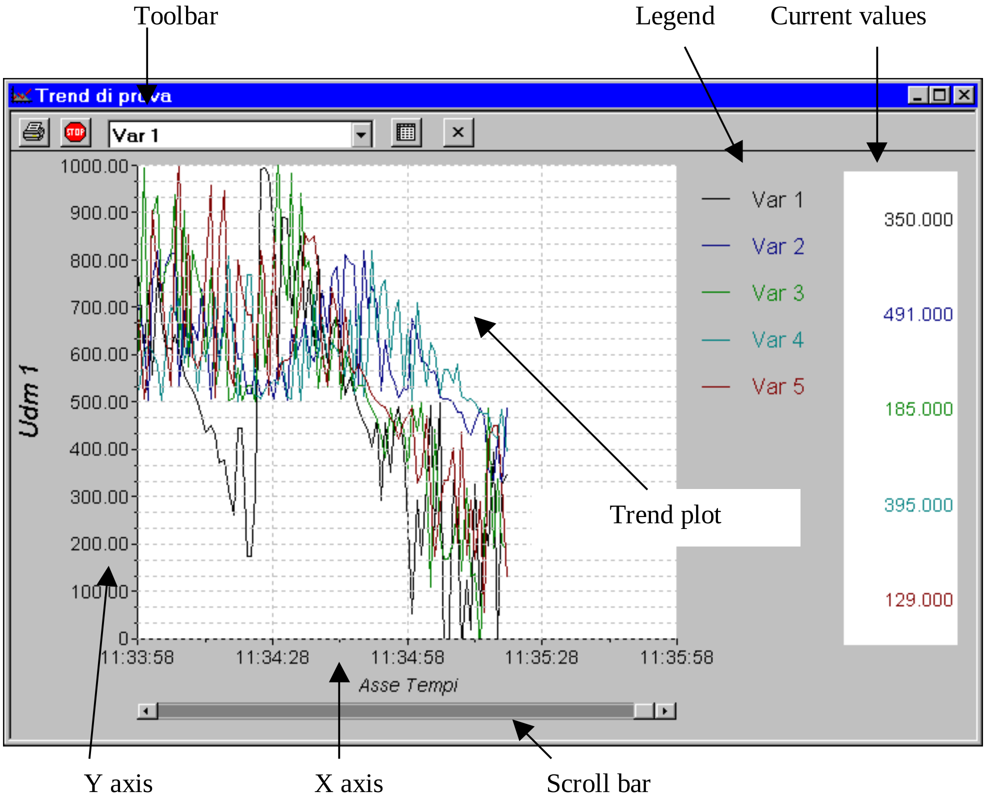
Trend di prova

Ora	Var 1	Var 2	Var 3	Var 4	Var 5
12.10.22	500.00	500.00	606.00	500.00	801.00
12.10.23	1015.00	517.00	1025.00	519.00	777.00
12.10.24	850.00	518.00	901.00	525.00	793.00
12.10.25	922.00	529.00	651.00	535.00	846.00
12.10.26	801.00	518.00	949.00	552.00	830.00
12.10.27	742.00	527.00	802.00	555.00	811.00
12.10.28	599.00	605.00	784.00	612.00	617.00
12.10.29	784.00	573.00	703.00	542.00	721.00
12.10.30	844.00	598.00	792.00	563.00	502.00
12.10.31	580.00	628.00	658.00	557.00	665.00
12.10.32	745.00	721.00	523.00	516.00	727.00
12.10.33	755.00	690.00	744.00	530.00	592.00
12.10.34	595.00	634.00	862.00	524.00	860.00
12.10.35	595.00	634.00	862.00	524.00	860.00
12.10.36	843.00	648.00	718.00	630.00	597.00

1.2 *"RUNTIME"* and Real Time Trends

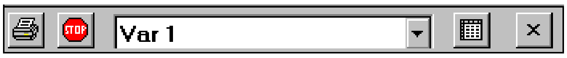
This chapter explains in greater detail how real time line trends are displayed and how their configurations are modified during runtime.

The Line trend window comprises parts already displayed during the development phase inside the window activated by the *Modify* command Chap. **Errore. L'origine riferimento non è stata trovata.**



The configuration of some trend characteristics can be modified directly during runtime by all operators but if the application has been protected (see Chapter **Errore. L'origine riferimento non è stata trovata.**), the variables channels can be selected only by those holders of passwords that allow them to make modifications
During runtime the line trend is provided with a command menu and a series of icons inside the toolbar that enable the required modifications to trend configuration and control to be made.

1.2.1 Toolbar



1.2.1.1 *Print icon*

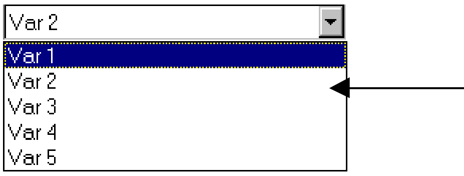
Click on this icon with the mouse to open the *Print* dialog window, from which print type and the trend area to be printed can be selected.

1.2.1.2 *Stop / Start icon* 🛑

Click on this icon with the mouse to temporarily stop the displayed graph in order to carry out a specific check. However, during the temporary arrest, storage of the values of the monitored data continues.
When the operator reactivates the graphic display, all the stored data will be set out in the trend.

1.2.1.3 *Graph channels*

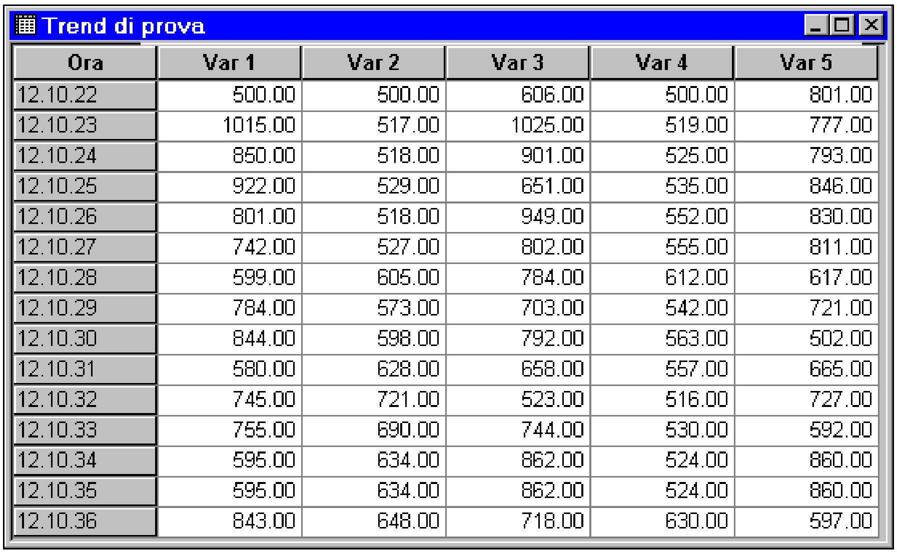
From this curtain menu, the operator can select the variable channel to be used as the scale for the Y axis.



Selecting a channel from the menu changes the scale reference of the trend’s Y axis. Up to 32 channels may be current.

1.2.1.4 *Values table icon*

Click on this icon with the mouse to open the table displaying all the values of the trend channels.
Run this command to display all the channel values that make up the trend. These values cannot be altered: they are only for consultation. To modify them, copy them onto the system's clipboard and transfer them to other applications such as Excel.



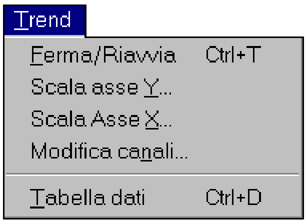
Trend di prova

Ora	Var 1	Var 2	Var 3	Var 4	Var 5
12.10.22	500.00	500.00	606.00	500.00	801.00
12.10.23	1015.00	517.00	1025.00	519.00	777.00
12.10.24	850.00	518.00	901.00	525.00	793.00
12.10.25	922.00	529.00	651.00	535.00	846.00
12.10.26	801.00	518.00	949.00	552.00	830.00
12.10.27	742.00	527.00	802.00	555.00	811.00
12.10.28	599.00	605.00	784.00	612.00	617.00
12.10.29	784.00	573.00	703.00	542.00	721.00
12.10.30	844.00	598.00	792.00	563.00	502.00
12.10.31	580.00	628.00	658.00	557.00	665.00
12.10.32	745.00	721.00	523.00	516.00	727.00
12.10.33	755.00	690.00	744.00	530.00	592.00
12.10.34	595.00	634.00	862.00	524.00	860.00
12.10.35	595.00	634.00	862.00	524.00	860.00
12.10.36	843.00	648.00	718.00	630.00	597.00

1.2.1.5 *Close icon*

Click on this icon with the mouse to close the displayed line trend.

1.2.2 TREND menu

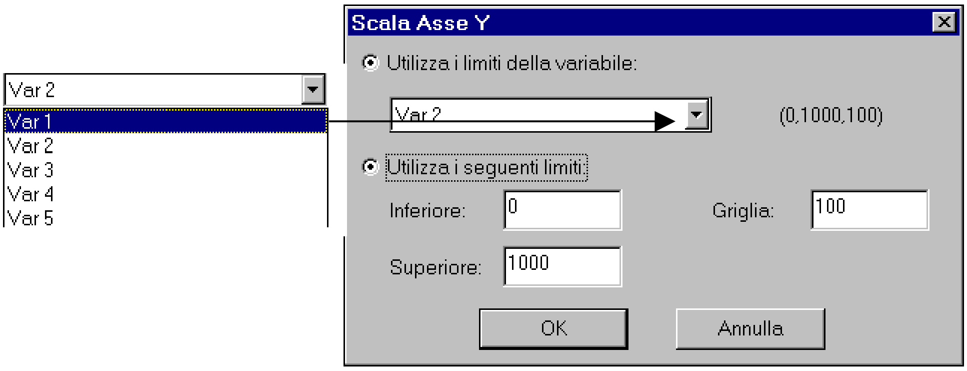


1.2.2.1 *Stop / Re-start*

This command enables the operator to temporarily stop the progress of the displayed graph in order to carry out a specific check. However, during the temporary arrest, storage of the monitored variables continues.
When the operator restarts the display of the graph, all the stored data are set out in the trend.

1.2.2.2 *Y axis scale*

Run this command to open the dialog window inside which the Y axis scale configuration can be modified.



Scala Asse Y

● Utilizza i limiti della variabile: Var 2 (0,1000,100)

● Utilizza i seguenti limiti:

Inferiore: 0 Griglia: 100

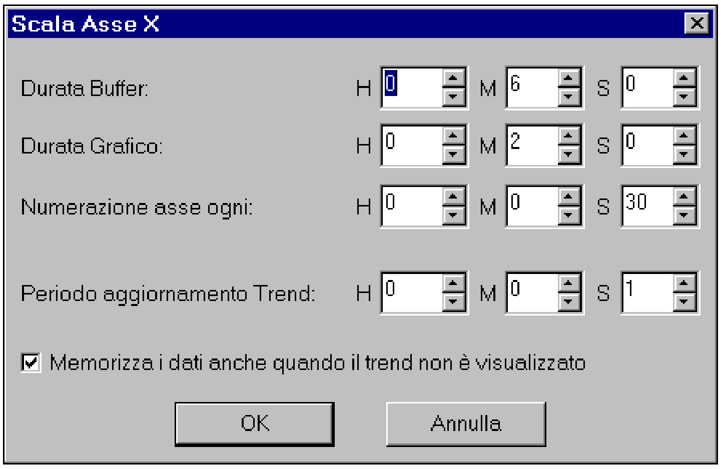
Superiore: 1000

The window is subdivided into two areas from which to select the types of limit to assign to the Y-axis scale.

Use following limits: select this option to key in the scale limits and the value to be assigned to the Y-axis grid.

1.2.2.3 *X axis scale*

Run this command to open the dialog window inside which the X axis scale configuration can be modified.



Scala Asse X

Durata Buffer: H 0 M 6 S 0

Durata Grafico: H 0 M 2 S 0

Numerazione asse ogni: H 0 M 0 S 30

Periodo aggiornamento Trend: H 0 M 0 S 1

☑ Memorizza i dati anche quando il trend non è visualizzato

The window is subdivided into two areas from which to select the types of limit to assign to the X-axis scale..

Buffer duration: inside these boxes set the duration of the buffer in which the sampled trend data are stored. The duration is expressed in hours, minutes and seconds and may be greater than the displayed graph duration.
A buffer is a temporary memory support for data exchange with a different type of support. It is often used to compensate for differences in speed between the different devices that manage the data.

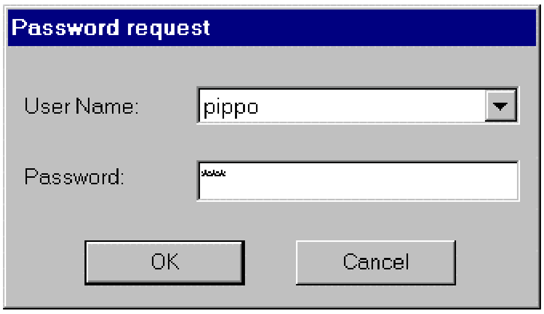
Graph duration: inside these boxes set the duration of the graph that displays the sampled trend data. The duration is expressed in hours, minutes and seconds.

Frequency of trend updating: inside these boxes the frequency of sampling of the variables configured for the trend is set. Frequency is expressed in hours, minutes and seconds and represents the time that passes between one sampling and the next of the values of runtime variables.

Store data even when the trend is not displayed: this command enables the line trend to sample and store data even when the graph is not displayed. This prevents data being lost during cycles and enables an immediate display of values to be obtained when the graph is opened during runtime.

1.2.2.4 *Modify channels*

Run this command to open a dialog window in which to modify configuration of the channels of the variables monitored in the trend.
If the trend has been protected, when the command is activated, a window opens that requests the operator's password.

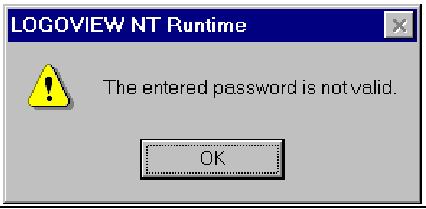


Password request

User Name: pippo

Password: ***

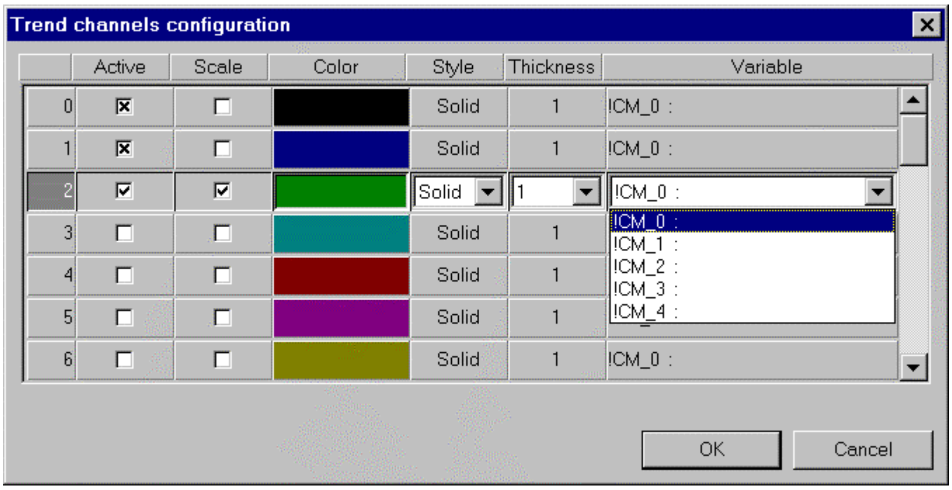
*If the operator keys in an incorrect password **LOGOVIEW NT** displays the following error message.*



LOGOVIEW NT Runtime

The entered password is not valid.

After the correct password has been keyed in the dialog window *"Trend Channel configuration"* is displayed, inside which the configuration of the trend channels for the monitored variables can be modified.



	Active	Scale	Color	Style	Thickness	Variable
0	☒	□	⬛	Solid	1	!CM_0 :
1	☒	□	🟦	Solid	1	!CM_0 :
2	☑	☑	🟩	Solid	1	!CM_0 :
3	□	□	🟦	Solid	1	
4	□	□	🟥	Solid	1	
5	□	□	🟪	Solid	1	
6	□	□	🟨	Solid	1	!CM_0 :

!CM_0 :
!CM_1 :
!CM_2 :
!CM_3 :
!CM_4 :

For full details on the configuration window, see Chapter **Errore. L'origine riferimento non è stata trovata.** *Trend Channel configuration.*

1.2.2.5 *Data table*

Run this command to display all the channel values that make up the trend. These values cannot be altered: they are only for consultation. To modify them, copy them onto the system's clipboard and transfer them to other applications such as Excel.

Programming manual Copyright LSI srl 1991-1997

The information contained in this document may be changed without prior notice.

V1.0 of November 1997

Contents

1. Creating and running programs.

The LOGOVIEW programming language is called LPE. ('Programming and events language'). Creating an LPE application package falls into three phases:

- **Writing the application package by means of the events editor (to open the window or windows, press the button at the top of the main toolbar);**
- **Configuration of the contact points with the graphic (box) and the events.**
- **Running the application package. If the application package does not function in the required manner, it must be modified. If necessary, re-check the boxes and run again.**

This chapter explains the above three pages with a simple example. When following the example, note that each instruction that requires an action to be carried out by the user is numbered.

The programs that are exemplified in the first chapters do not include all error management. The programs are thus shorter and easier to understand but when running these programs do not key in the type of value required as running the application package could generate errors. For this reason, in view of the teaching purpose of the illustrated examples, the user must carefully follow the instructions set out below. When the user develops his own programs he must add additional instructions for error management. A subsequent chapter will illustrate in detail the methods for error management.

1.1 Creating a new event

Apart from the word ’application package, the word ’event’ often appears. The terms application package and event will from now on often be used as synonyms even if they are normally not synonyms: a real application package often comprises many events whilst events normally comprise a single event, as we shall see.

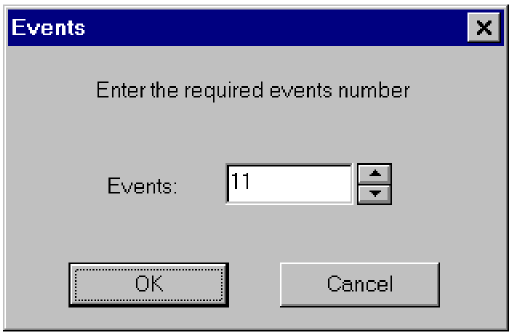
1.2 What is an event?

An event is a sequence of LPE instructions. In programming in general, the term ’procedure’ is used to identify a sequence of instructions so ’event’ can be considered to be a synonym of ’procedure’. At the start, Logoview automatically runs event number 0. The other events are run only if circumstances cause them to be run. For example, if a timer reaches the set count or if a trigger variable changes status. Within an event Logoview starts to run instructions from index 1 and then carries out all the subsequent instructions until the end of the event is reached. Each event must finish with an END or RETURN instruction.

1.3 Creating an event

Creating a new event and assigning a name:

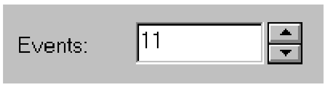
after opening the events editor, click on the file icon on the events editor toolbar. After this, LOGOVIEW NT will open the dialog window shown below:



Events

Enter the required events number

Events: 11

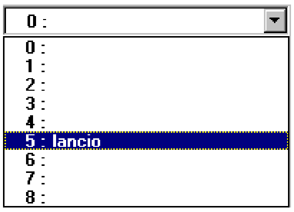


Events: 11

Enter the number of events to be configured into the box either by keying in the number or by clicking on the increase or decrease arrows ▲▼ at the side of the text box.

After pressing OK the events will have been configured.

There is no need to configure all events at once: they can be configured as and when required. It is nevertheless wise to divide events into blocks containing homogenous events, i.e. events that treat the same argument , as is the case with Wizard automatic composition.

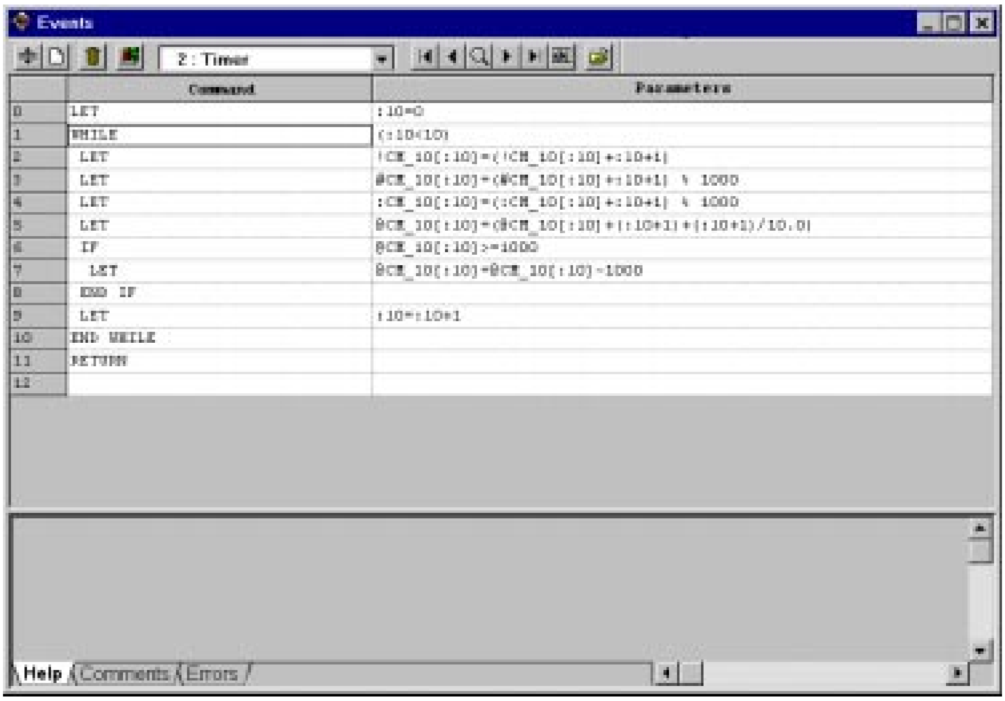


0 :
0 :
1 :
2 :
3 :
4 :
5 : lancio
6 :
7 :
8 :

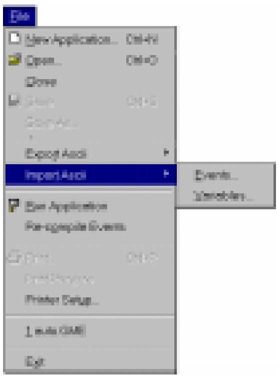
Use the combined box on the control bar of the events editor to see if the configured events are current. Press ▼ to open the list of configured events. As can be seen from the image alongside, the list contains all the configured events. They are marked by a order number and two dots. The current event in which work is possible can also be selected from the list. After configuring the events they can also be assigned a name to make identification by the developer easier. The names of the events must begin with a letter and can contain both letters and numbers. Spaces or characters outside the range [0-9,A-Z,a-z] are not permitted.

A name should be chosen that describes the action of the event in such a way that the event can be more easily identified over time.

1.4 In the program editor



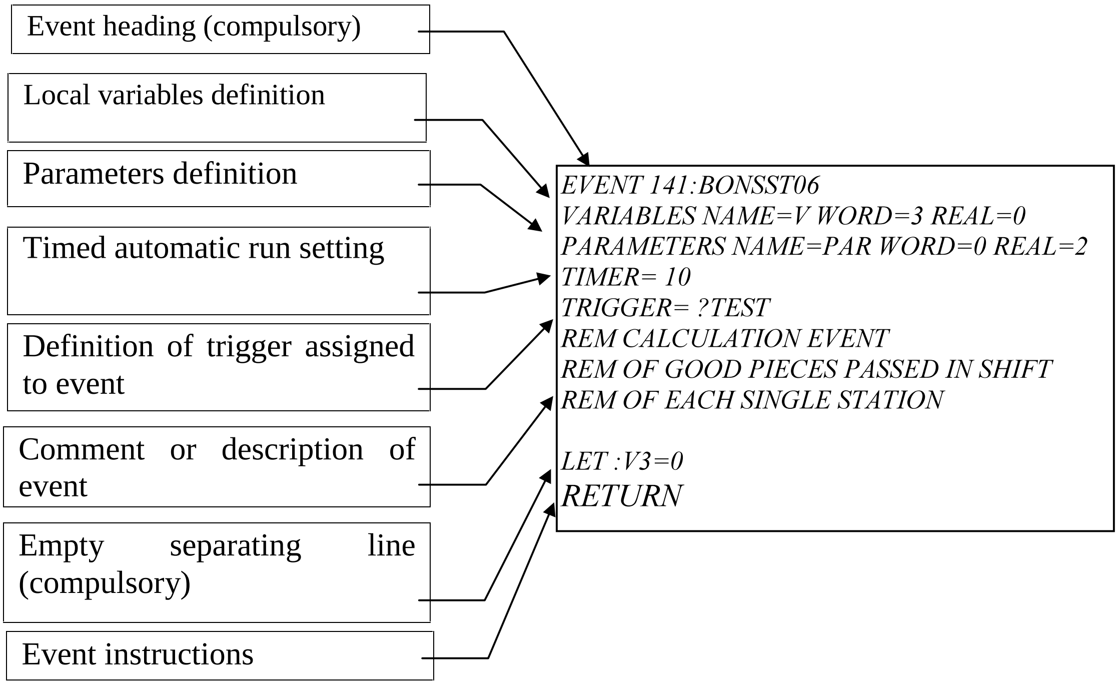
	Command	Parameters
0	LET	:10=0
1	WHILE	(:10<10)
2	LET	!CH_10[:10]=(!CH_10[:10]+:10+1)
3	LET	#CH_10[:10]=(#CH_10[:10]+:10+1) % 1000
4	LET	:CH_10[:10]=(:CH_10[:10]+:10+1) % 1000
5	LET	@CH_10[:10]=(@CH_10[:10]+(:10+1)+(:10+1)/10.0)
6	IF	@CH_10[:10]>=1000
7	LET	@CH_10[:10]=@CH_10[:10]-1000
8	END IF	
9	LET	:10=:10+1
10	END WHILE	
11	RETURN	
12		



Apart from using the events editor described above an ASCII editor can be used to write the events outside Logoview (e.g. in Notepad). After the events have been written with the outside editor they can be imported into Logoview by means of special functions that are found in the file menu of the main menus bar. To change the events already in Logoview into ASCII use the export function in the file menu. In this way Logoview will save all the events current in an ASCII file (with the extension ’.prg’) that can be modified by your preferred editor.

Whilst importing the events in ASCII format Logoview carries out a syntactic code check. This check checks only the formal aspect of the language. No logic errors will be detected until the code is run with the runtime code. LOGOVIEW detects incorrect code lines during the syntactic check and comments on them so that no problems are caused by running the application package. After ASCII has been loaded into the errors list all the detected errors are listed at the bottom of the editor.

A different type of editor is to be used for each type of event that is to be written. When using an ASCII editor to write an event, some description lines must precede the instructions that make it up in the following manner:


Event heading (compulsory)
Local variables definition
Parameters definition
Timed automatic run setting
Definition of trigger assigned to event
Comment or description of event
Empty separating line (compulsory)
Event instructions

```
EVENT 141:BONSST06
VARIABLES NAME=V WORD=3 REAL=0
PARAMETERS NAME=PAR WORD=0 REAL=2
TIMER= 10
TRIGGER= ?TEST
REM CALCULATION EVENT
REM OF GOOD PIECES PASSED IN SHIFT
REM OF EACH SINGLE STATION

LET :V3=0
RETURN
```

Each event must start with the key word EVENT followed by the sort number, the two dot character (:) and the event label. The example defines event No. 141 with label BONSST06.
The lines that follow the key words EVENT describe the event characteristics and must immediately follow the line with EVENT. In other words, there must be no empty lines because Logoview would interpret an empty line as the end of a heading and the start of the event instructions. Each event is divided into two sections: the heading and the main body. The heading contains all the elements that describe the event characteristics, such as, for example, the label and the sort number. The main body contains all the instructions that make up the event.

The heading and the body of an event must be separated by at least one empty line. Each event must end with an END or RETURN instruction. Between one event and the next, at least one line must be left empty.

If the editor inside LOGOVIEW is used these elements are not necessary because all the settings are defined by dedicated commands.

A very simple command, like a command created for a test, consists of a single LPE event.

1.5 Example procedure

The following pages explain the following example procedure:

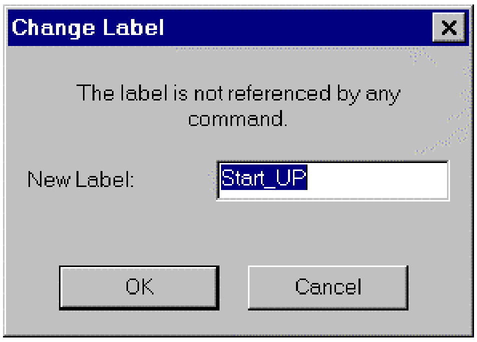
```
EVENT 1:TEST

DISPLAY  0,0,0,15, 'This is my first event'
WAIT  100
DISPLAY  0,0,0,15,"End of program"
END
```

This procedure does not run anything - it is only an example of how the key LPE words (DISPLAY, WAIT and END) are used. The procedure displays 'This is my first event' on the screen. After only a few seconds this message is replaced by the message 'End of program'. It then finishes.

1.6 Writing and reviewing procedure

Before entering the instructions that make up the procedure, the name of the procedure must be indicated after the word *EVENT*. If you are using the programming environment (development) to enter the name, press the button displaying the letters 'ab' between square parentheses ab|. This action opens a dialog window that enables the name of the event to be inserted. After the name of the event has been inserted, press OK to confirm or cancel by pressing 'cancel' in the dialog window.



Change Label

The label is not referenced by any command.

New Label: Start_UP

On the other hand, if you are using an ASCII editor, the name of the event can be entered immediately afterwards between the two points that follow the event number:

EVENT 1: ***TEST***

Any name can be used as long as the Logoview rules are complied with: do not use names that are longer than 128 characters; the name must begin with a letter of the alphabet and must not contain spaces or characters apart from alphanumeric ones.

There is no need to give events names. However, in order to make events easier to read and transfer it is always wise to assign a name to events, preferably a name that is pertinent to the task performed by the event.

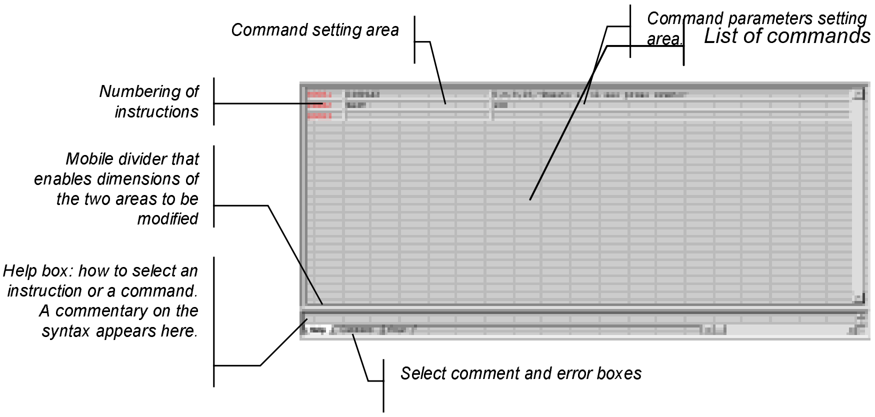
The instructions for this procedure can now be entered. Here again, there are two possible procedures, depending on whether one is operating within the LOGOVIEW development environment or whether an ASCII editor is used. In the former case, just key in the instructions one at a time, remembering to follow the simple syntactic rules and of course the grammar of the language. There are not many rules and they are simple.

- ✎ *The events must be separated by an empty line.*
- ✎ *The head and body of the event must be separated by an empty line.*
- ✎ *Each line must contain only one command or instruction.*
- ✎ *The events without instructions can be omitted. There can therefore be gaps in the numbering of the events. However, the events must be in ascending order.*
- ✎ *In order to read the code better, it is better to indent the line when using the instructions WHILE-END_WHILE or IF-THEN-ELSE*

If the above rules are followed, the event must be the same as the one shown in the figure on page 6.

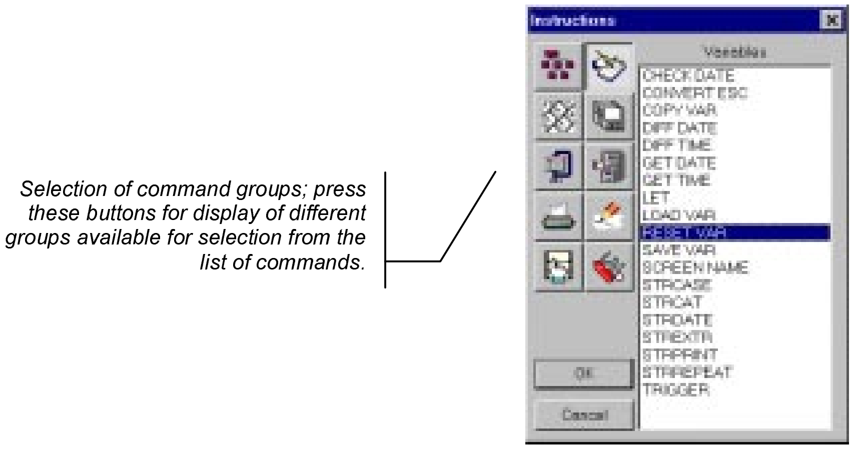
Those who prefer to use the internal editor operate within a framework that is somewhat more rigid but on the other hand they have a series of aids that make up for the greater rigidity.

First, the working environment must be explained:


Command setting area
Command parameters setting area.
List of commands
Numbering of instructions
Mobile divider that enables dimensions of the two areas to be modified
Help box: how to select an instruction or a command. A commentary on the syntax appears here.
Select comment and error boxes

To enter an instruction, proceed as follows:

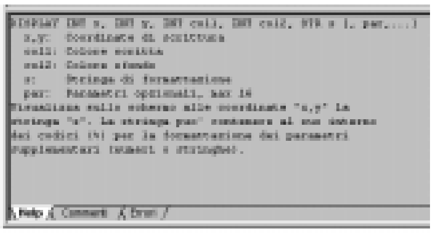
- Place the cursor on the command setting area and press the left-hand side of the mouse.
- This action makes a dialog window appear. In this window there is a list that sets out the LPE commands and functions divided by argument .



Selection of command groups; press these buttons for display of different groups available for selection from the list of commands.

- Select the group containing the instruction to be used. In the example, the DISPLAY belongs to the group of instructions that is represented by the button with a picture of a pencil .
- Select the instruction from the list, after obtaining a display. If necessary, use the slider to the side of the list to select the instruction..
- Finally, press OK

After these operations have been carried out, the selected command will appear on the first available line of the events editor.



```
DISPLAY INT x, INT y, INT col1, INT col2, STR s [, par....]
  x,y:  Coordinate di scrittura
  col1: Colore scritta
  col2: Colore sfondo
  s:    Stringa di formattazione
  par:  Parametri opzionali, max 16
Visualizza sullo schermo alle coordinate "x,y" la
stringa "s". La stringa puo' contenere al suo interno
dei codici (%) per la formattazione dei parametri
supplementari (numeri o stringhe).
```

At this point, key in the parameters that determine the behavior of the instruction. To do so, place the cursor in the right-hand side of the editor and then click. At the bottom of the editor the Help text and the command syntax will appear. If this does not occur it means that the Help box has not been selected: select it by clicking on the word Help, which can be seen at the bottom.

Key in the parameters by following the syntax. In our example, we wish to write the text *'This is my first event'* on the coordinates 0.0 of the black screen (color 0) on a white background (color 15). The parameters line should therefore have this appearance:

0,0,0,15,"This is my first event".

For each new instruction that is to be used, repeat the previous point.

1.7 Action of key words during implementation of the application package

DISPLAY - displays on the screen the text shown between inverted commas at the specified coordinates in the color given as a parameter. The text to be displayed in the first instruction is ’This is my first event’.
WAIT 100 - suspends the current event for a set number of tenths of a second. In the example, the pause is 10 seconds.
END - finishes the event and the application package.

2. *Variables and constants*

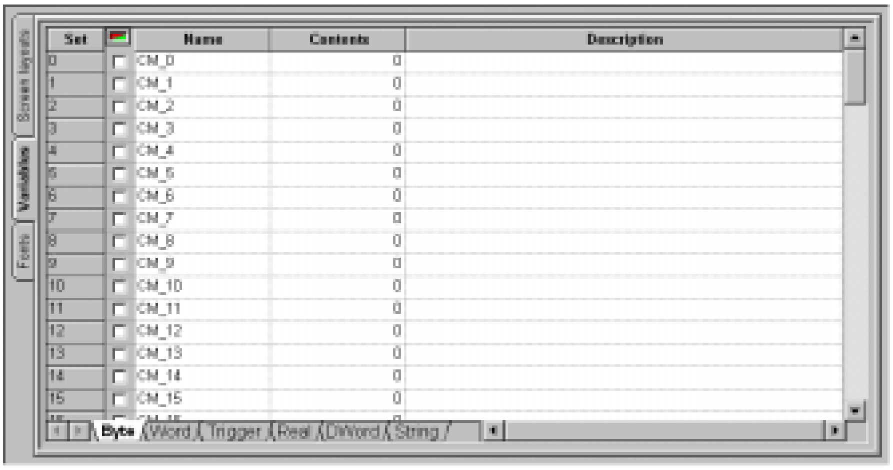
The LPE application packages can process data in different ways. For example, they can make calculations or save and call up the text strings (such as names and telephone numbers from a data file).

In all cases, the application package must be able to manage values, i.e. different types of numbers, strings, etc.

In LPE there are two methods for managing values: with variables and constants. The constants are fixed values: e.g. 1,2,3. The variables are used to store the values that can change - for example a variable called 'X' may start with the value 3 but then take on the value 7.

2.1 Defining variables

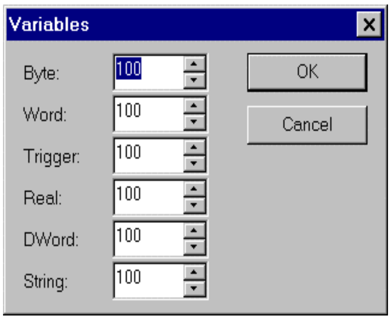
First of all, the variables must be created. They cannot be used unless they have already been created and configured by means of the variables box from the Basic Settings window.



Set		Name	Contents	Description
0	□	CM_0	0	
1	□	CM_1	0	
2	□	CM_2	0	
3	□	CM_3	0	
4	□	CM_4	0	
5	□	CM_5	0	
6	□	CM_6	0	
7	□	CM_7	0	
8	□	CM_8	0	
9	□	CM_9	0	
10	□	CM_10	0	
11	□	CM_11	0	
12	□	CM_12	0	
13	□	CM_13	0	
14	□	CM_14	0	
15	□	CM_15	0	

From this box the configured variables can be seen. Open the dialog window that is used for configuring other variables and selecting different types, assign the names and comments and insert the initialization values.

To configure the number of variables, click with the arrow on the button Set in the top left-hand corner. This opens the dialog window and allows the variables to be set.



Variables

Byte: 100

Word: 100

Trigger: 100

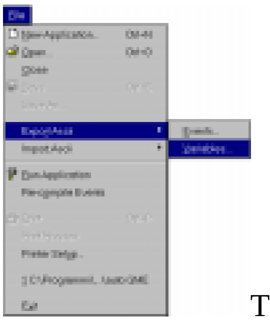
Real: 100

DWord: 100

String: 100

The window shows the fields in which to set the quantities required for each one of the variables supported by LOGOVIEW. As for all the other settings boxes, the value can be entered directly. Otherwise, use the arrow keys to the side of the settings box. Press the arrow pointing upwards to increase the value and press the arrow key pointing downwards to decrease the value. After making the required modifications press OK to confirm.

When operating within the dialog window to configure variables, the number of configured variables can be increased or decreased. Exercise great care with decreasing variables because LOGOVIEW does not check whether the eliminated variables are used in events or assigned to some object. Application package malfunctions could arise if they are used.



The variables list can also be modified by an ASCII editor, as is the case with events. In this case too, the items from the import and export menu are used. The ASCII file that is exported from LOGOVIEW has the extension [.*VAR*] and this format:

```
Byte  5

VAR1      =0 "COMMENT ON VAR 1'
VAR2      =0 " COMMENT ON VAR 2"

String  5

VAR1      ="Contents of string1" "COMMENT ON VAR 1"
VAR2      ="Contents of string2" "COMMENT ON VAR 2"
```

The format is extremely simple. The variable type is entered, followed by a space and the number of variables that the user wishes to configure. Two lines are then left empty and there then follows the list of the variables. This must be identical to the number that is entered at the start of the section. Each line contains a variable that must be set out in the following manner:

variable name, space = initial value, comment between double inverted commas.

If the variables are string variables, the initial value must also be contained within three double inverted commas. In addition, if the variable does not have a name, an accent [‘]must be entered instead of the name. Finally, if there is no comment, just leave it out. To recapitulate, a string variable without name and comment must be entered as follows:

```
`   =”Contents of string anonymous”
```

Regardless of whether the variables are configured by the ASCII editor in the manner shown above or are configured by the basic settings window, the variables are GENERAL variables. In other words, it will be seen by all the events and all the objects that make up or will make up the application package. LOGOVIEW also provides scope for different types of LOCAL variable. These are types of variable that can be seen only inside the event in which they were defined. This type of variable can be defined either in the configuration method or by the ASCII editor. The list of configured variables is listed immediately below the name of the event, in this way:

```
EVENT 1:TEST
VARIABLES NAME=LOC WORD=2 REAL=2 STRING=1
REM Test event
REM example of configuration of local variables

DISPLAY  0,0,0,15,"This is my first event"
WAIT  100
```

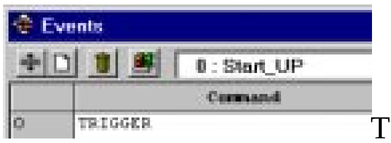
The second line means that 2 WORD-type variables, 2 REAL-type variables and 1 string-type variable are configured whose names start with LOC.

The local variables have a base name that is followed by the order number and is preceded by the symbol that represents the type. The following syntax is therefore used to refer to the first configured WORD variable:

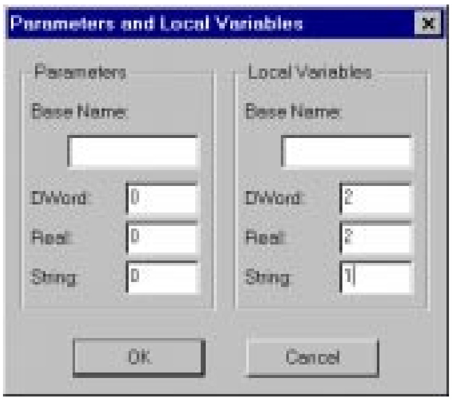
#LOC0

where the symbol # indicates that the WORD-type variable is to be used. LOC is the name given to the variables and 0 indicates the first variable.

For local variables only DWORD, REAL or STRING can be configured. When such variables are assigned to GENERAL BYTE or TRIGGER variables. Caution must be exercised as the value is cut if it exceeds the capacity of the destination variable.



To configure the local variables from the development environment, open the events editor, click on the event whose local variables need to be defined and then press the variables button . This is located in the events editor window. The dialog window will open that enables the variables to be configured.



Parameters and Local Variables

Parameters
Base Name:
DWord: 0
Real: 0
String: 0

Local Variables
Base Name:
DWord: 2
Real: 2
String: 1

In this dialog window, as can be seen from the figure, both the quantity of variables and the name to be used can be entered.

2.2 Choosing a variable

Before defining the variables, decide what type of information they must contain. There are different types of variables for different types of value. If you try to assign an incorrect value to a variable or if you try to assign a value to a variable that has not been configured an error message will be displayed.

When using a variable, specify the type and add a symbol in front of the name of the variable.

2.2.1 Numbers

Binary values can only have the numbers 0 or 1 and therefore require a TRIGGER variable, which can only have two values, 0 or 1. A ***TRIGGER*** value can have no other value. It is thus a Boolean variable. The TRIGGER variables use the symbol ?. For example:

?12, ?PROX

For small numbers, e.g. 6, use a BYTE variable. A ***BYTE*** variable is a whole figure without any sign between **0** and **255**. It therefore provides a significant memory saving if we are dealing with digits that are within the 0-255 range.

BYTE variables use the symbol !. For example:

!25 , !VAR

For average numbers, e.g. 1500, use a WORD variable. A ***WORD*** variable is a whole variable with a sign and its variation field is greater than a byte variable. Its value may vary between **-32768** and **32767.** WORD variables use the symbol **#**. For example:

#33 , #STAZ

For large whole number, e.g. 10000000, use a DOUBLE WORD (DWORD). A ***DWORD*** variable is a whole sign. It can be used between **-2147483648** and **2147483647**. It is therefore greater than a Word because it can represent a larger scale of values. The DWORD variables use the symbol ’:’ (colon). For example:

:231, :TEMP

For decimal numbers such as 2.5 use a REAL variable**.** The sign of a ***REAL*** variable is a floating point. It can therefore take on values between **-1 and 38** and **1 and 38**. This is the only type of variable that can show rational quantities. This is the only type of variable that can represent rational quantities. If decimal numbers need to be used at a certain point in the application, e.g. 1.22, a REAL variable must be used and not a whole variable like WORD or DWORD. Otherwise, unforeseeable results will be obtained. (This type of variable is illustrated in greater detail at the end of this chapter). REAL variables use the symbol @. For example:

@28, @APOT

The REAL variables must also be used for very large or small numbers that are outside the range of the DWORD variables. REAL variables manage large numbers such as +-9.9999e37 and small numbers like +-1e-37.

2.2.2 Text

For the text ’ARE YOU SURE?’, ’54th’, etc. a **string variable** must be used. (In LPE the text fragments are called strings). The string variables are preceded by the symbol $. For example:

$34, $NAME

The string variables occupy a quantity of fixed memory for each variable plus the space required to contain the text that is stored. A variable string can contain up to 512 characters.

2.2.3 Vectors

For more flexible use LOGOVIEW enables a group of variables to be referred to as if they were a single vector. This type of management is useful for managing lists of values, for example. Instead of having to refer to each variable separately, (e.g. #A, #B, #C, #D and #E) each value can be referred to as if it were part of a single vector (#A[1], #A[2], #A[3] etc.)

In this way the variables can be accessed by making use of an index that can be a constant or a variable. For example:

!23,!24 is the equivalent of !23,!23[1]
If the value 1 is assigned to the variable #IX,
we can write !23, !23[#IX]

LPE does not support two-dimensional matrices.

2.2.4 Initial values

The initial value of the numeric variables that have just been created is 0. The string variables that have just been created do not contain a character. An initial value can be assigned that is different from the default value. It is sufficient to open the configuration window and to directly insert the value next to the name of the variable to be initialized in the value column.

2.2.5 Choice of descriptive names

To facilitate the writing of LPE application packages and in order to understand them better when they are reread afterwards, the variables should be assigned names that define the values that they contain. For example, in a procedure that calculates the efficiency of fuel the variables

speed and **distance** can be used. The names of the variables can be descriptive and they must satisfy the following conditions:

- *maximum length of 32 characters*
- *they must begin with a letter but numbers or letters can follow.*
- *Capital or lower case characters can be entered. SpeEd and SPeEED are considered to be different names.*
- *Space characters, punctuation symbols and accented characters (è, ù, etc.) cannot be used.*
- *None of the command or function names can be used.*

It is also possible to refer to the variables directly by means of the order number under which they are entered on the list. This value is the offset value. The offset value is always immediately available, even if the variables do not have a name. It should nevertheless be remembered that indiscriminate use of the offsets instead of the names makes the application packages difficult to read.

Referring to variables by name rather than by offset also avoids problems when the lists of variables are edited by the ASCII editor by removing the unused variables from the list. In this case, direct reference by the offset leads to a reference error as the offset will refer to a variable that is different from the original one. On the other hand, the name will still refer to the same variable, even if the offset has been changed by the modifications made.

2.3 Preferences

WORD variables use less memory than DWORD and REAL variables
Whole variables are usually processed more quickly than those with a variable floating point.

2.4 Assignment of values to the variables

2.4.1 Assignment of values

A value can be assigned directly to the variable as follows:

```
LET #X=5
LET #Y=10
```

This event adds two numbers:

```
EVENT 5: addit:

LET #X=569
LET #Y=203
LET #Z=#X+#Y
END
```

where

addit: is the name of the procedure.

The *LET* instruction enables the result of a variable to be assigned to an expression. In the case in question the first *LET* assigns the value 569 to a *WORD X* variable. The second *LET* assigns the value 203 to the *Y* variable of the same type and finally the last *LET* assigns the sum of the two values contained in the variables *X* and *Y* to the variable *Z*.

The instruction *END* closes the event and the application package.

2.4.2 Assigning values to string variables

To assign text values to string variables, proceed as follows:

LET $X = ”Test string”

The text must be contained between inverted commas.

2.5 Assigning values to a vector

For the indexed variables and the vectors the same rules apply as for the single variables. The only difference is that they can be assigned values both by addressing the variable directly and by indexing it. For example, if we have a vector comprising ten BYTE-type variables that go from offset 23 to offset 32 inclusively, we can follow two different procedures:

LET !27 = 45

or

LET !23[5] = 45

The two methods address the same variable. It is also possible to assign the index to a variable, in which case the code would be:

LET #10 = 5
LET !23[#10] = 5

2.6 Arithmetical operations

It is possible to use:

+	add
-	subtract or make negative
/	divide
*	multiply
%	percentages

The operators have the same precedence as with calculators. For example, 3+51.3/8 is considered to be 3+(51.3/8) and not (3+51.3) /8. For further details on operators and precedence, see the Logoview NT instruction guide.

2.7 Function values

There are two types of key word - **commands** and **functions.**

A command is a direct instruction that orders the LPE to carry out a given action: *DISPLAY* and *WAIT*, for example, are commands.

A function is like a command but at the same time it returns a value that can be used later on. *ABS* is a function because it calculates the absolute value of the parameter and returns it at the output.

2.8 Expressions

A value can be assigned to a variable by means of an expression, i.e. a combination of numbers, variables and functions. For example:

LET #Z = #X + (#Y / 2)

in which *#Z* is the same as the value *#X* plus the value *#Y* divided by 2

*LET #Z = #X * #Y + 34.78*

in which #Z is the same as the value #X multiplied by #Y, plus 34.78

LET #Z = #X + COS (#Y)

assigns to *#Z* the value *#X* plus the cosine *#Y*. (*COS* is an LPE function). *COS* requires a value or a variable as a parameter and returns the result directly. The value of this type of function must be placed between round parentheses after the name of the function. The values assigned in this way to the functions are called function arguments . They will be examined in greater detail in the next chapter.

These are all operations with the variables *#X* and *#Y*. They assign the result to *#Z* without affecting the value of *#X* or *#Y*.

Three types of method are used to change variable values:

arithmetical operations, such as multiplication and addition. For example:

LET #Z = #sales + #costs

or

*LET #Z = !Z * (4 - #X)*

use of the LPE functions, for example:

LET #Z = SIN (PI / 6).

assigns to *#Z* the sine value of *PI / 6*. (SIN is an LPE function *PI* whilst is the constant π (Greek Pi or the value 3.14).

Use of key words like READ PLC or LOAD VAR that read the values from outside devices.

2.8.1 Self-referencing

In the expressions the variables can refer to one another. For example:

LET #Z = #Z + 1

increases the current value of *#Z* by one.

LET @X = @Y + @X / 4

makes *@X* the same as a quarter of its current value and adds the value *@Y*. Note that using REAL variables prevents loss of precision if there are any decimal digits in the division.

2.9 Constants

In the LPE application the numbers (and the strings between inverted commas) are called constants. Constants are easy to use. For example:

LET @X = 0.32
LET #Z = 569
LET :Y = 32768
LET $ST = ”string”
LET @X (1) = 4.87

2.10 Problems with whole numbers

When calculating an expression, LPE uses the most simple arithmetic for the numbers in question. If all the numbers are whole, the arithmetic for whole numbers is used (WORD, DWORD, TRIGGER or BYTE). If one of the numbers is not whole the arithmetic for floating point numbers is used (REAL).

This enables calculations to be made very rapidly. Care must be taken to ensure that the numbers do not exceed the range for the type of arithmetic used. For example, in:

*LET #X = 200 * 300*

both numbers 200 and 300 are whole and so the arithmetic for whole numbers is used for greater speed. The result is 60000 and cannot be assigned to the WORD variable *#X* because it exceeds the whole number range (from 32767 to -32768), so an error occurs because the excess amount is not included in the result. To avoid this problem, the value can be assigned to a DWORD variable (*:X*). If the previous example is written as:

*LET :X = 200 * 300*

LPE will assign the result to a DWORD variable and will then produce the correct result (60000).

Warning. Whole figure arithmetic uses only whole figures. For example, if #X is 4 and #Y is 7, #Y / #X gives the result 1. The decimal part of the calculation is thus lost or at least the rest of the division is lost. To avoid this problem there are REAL variables. They are somewhat slower than the others and are therefore not recommended for general use but they are required for calculating values with decimal parts.

2.11 Operation on strings

Fairly complete support is given to the strings inside Logoview. Remember that Logoview basically uses the strings to send messages or to work with databases. In all cases the language of Logoview provides powerful functions for processing strings. For example, if it is necessary to know the length of a string the function *STRLEN* can be used; if a string has to be found within another string the function *STRSTR* can be used.

2.12 Displaying variables

DISPLAY is the command that is used to obtain a display of any combination of test and variable value messages. This instruction enables strings to be composed that are then displayed at the coordinates that are given as parameters.

```
LET #X = 500
DISPLAY  0,0,0,15,”Valore di X = %d” , #X
```

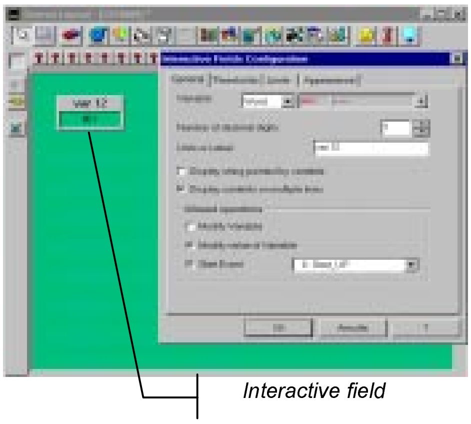
In the example the instruction prints the string at the coordinates 0-0 of the active screen. The string is made up of alphanumeric characters that are placed inside double inverted commas and the value contained in the WORD variable *#X.* The value contained in the variable replaces the code %d. This means that in that string position the entire value contained in the variable should appear. Several different % codes can be positioned within the string and a parameter positioned after the formatting string must match each code.

The *DISPLAY* makes the necessary replacements and maintains the order that goes from left to right. If the inserted codes do not match, when the string is printed out difficulties could occur. As the code %d must be used for the whole variables the code %f must be used for the REAL variables. The *DISPLAY* instruction also permits an unlimited number of types of formatting, alignments and character counts so as to provide maximum flexibility in column formation. For a complete discussion of the flags and codes, see the second part of the manual.

As the codes inside the *DISPLAY* instruction are preceded by the percentage symbol (%), the problem of printing this symbol arises. To print it, enter two symbols instead of one: *%%*

2.13 Keyboard values: the interactive fields

The interactive fields are another system that is used to display variables. These are boxes that are positioned during the configuration phase on the box in which the value is to be displayed. A variable and all the characteristics that represent the value are assigned to each interactive field. When the command is run, the screen in which the interactive field is located is displayed. LOGOVIEW displays the value of the variable assigned to it and maintains its configurations.



Unlike the *DISPLAY* instruction, which prints the interactive field only once when it is run, as its name suggests, this interactive field interacts with the assigned variable. If the value of the variable is modified these modifications are therefore displayed immediately. In addition, because of the close connection between the interactive field and the variable that is assigned to it the interactive field is the simplest and most immediate method that LOGOVIEW provides for keying in values inside the variables. Whilst the application package is being run, if the interactive field is configured for modifying the value contained inside the variable the cursor can be clicked on the same field and the new value can be keyed in. Finally, press Return (Enter) on the keyboard to confirm. The keyed-in value will replace the old value. The interactive fields is a much more sophisticated instrument than it seems. Its configurations in fact enable the user to insert a caption, modify the value contained in the variable, change the color of the display for the different values, etc. (For information on configuring interactive fields, see the application’s configuration manual).

3. Loops and skipping

The LPE application packages set out in the previous chapters consist of a series of instructions that are run one at a time from start to finish.
However, other methods may also be used for running the application.

- **Repetition of a series of instructions (loops).**
- **Running one series of instructions rather than another (IF instructions).**
- **Skipping from one line of the application to another.**
- **Running events initiated by the user.**
- **Running events triggered by status changes (triggers).**

3.1 Repetition of instructions (loops)

The commands *WHILE...END WHILE* are structures and do not run any data operations. They simply check the order in which the instructions are carried out. In fact, these two commands repeat a series of instructions until the series of instructions *WHILE...END WHILE* is followed in that order.

3.2 WHILE...END WHILE

```
EVENT 1:TEST2
LET #A=10
   WHILE #A>0
            DISPLAY  0,0,0,15,"A value=%d",#A
            LET #A=#A-1
   END_WHILE
   DISPLAY  0,0,0,15,"End of program"
   END
```

The instructions between *WHILE* and *END WHILE* are run only if the condition defined by *WHILE* is true, i.e. if this example *#A* is greater than 0.
Initially *#A=10* and therefore A=10 is displayed on the screen. *#A* is then taken to 9. *#A* is always greater than zero and then A=9 is displayed.
§>This operation is continued until A=1 is displayed. *#A* is then zeroed and End of program is then displayed.<§
Warning. If loop status is not set correctly the loop could be run an indefinite number of times (e.g. if there was no program line to decrease the variable *#A*). In addition, if the condition is not true when the loop starts for the first time the instructions between *WHILE* and *END_WHILE* will not be carried out even once.

3.3 Choosing between several different instructions

Different combinations are possible in an application (*#X* may be 1, 2, or 3...). They cause different behaviors for each different circumstance (if *#X* is 1, run one instruction, if it is 22, run another, etc).

This setting can also be obtained by means of the structure *IF...END_IF*:

```
IF condition1
   run these instructions
ELSE
   IF condition2
      run these instructions
   ELSE
      IF condition3
           run these instructions
      ELSE
           run these instructions
      END_IF
   END_IF
END_IF
```

The above lines carry out:

- the instructions after the line IF if condition1 is met

or

- the instructions after one of the ELSE lines if condition2 or condition3 is met..

or

- the instructions after the most internal ELSE line if no condition is met, i.e. neither condition1, condition2 or condition3...

After running one of the alternatives they continue with the instructions after *END_IF*.

It is not compulsory for each *IF* to have an alternative with ELSE inside it but each *IF* must finish with *END_IF*. The structure must start with *IF* and finish with *END_IF*. In each event there must be the same number of *IFs* and *END_IFs.* In addition, the instructions *ELSE* and *END_IF* can be used only after an *IF*.

Inside the statement any combination of instructions may be included, including the loops *WHILE...END WHILE*. The only limit is the maximum number of instructions that can be used to make up an event: 32767.

3.4 Nesting of loops

As we have seen in the two previous paragraphs several WHILE...END WHILE loops or decision sets can be nested together. This means that a loop can be nested inside another loop , and so on. This consideration also applies to IF...END_IF statements and to combinations of the two statements. Each nesting element is known as a 'level'. LPE has provision for up to 50 (fifty) nesting levels possible. If this limit is exceeded the system could generate an overflow error. It is nevertheless highly improbable that this limit will be reached, let alone exceeded, because highly nested loops make the code illegible. It is in fact always advisable to create events that carry out single loops that can then be called up from within a loop , as we shall see in the following paragraphs.

3.5 Using IF and WHILE

```
EVENT 1:CODE_Z
LET #A=0
   WHILE #A<10
           IF #X [ #A ] = 100
                   DISPLAY  0,0,0,15," X value [%d]=100",#A
           ELSE
                   IF #X[ #A ] > 100
```

```
                    DISPLAY  0,0,0,15, "[%d] GREATER",#A
              ELSE
                    DISPLAY  0,0,0,15, "[%d] LESS",#A
              END_IF
        END_IF
        LET #A=#A-1
  END_WHILE
  DISPLAY  0,0,0,15,"End of program"
  END
```

The application checks the values contained in a whole WORD #X vector comprising ten elements. The outside loop checks all ten vector elements, using the WORD #A variable as a vector index and checks that the loop continues until the #A variable is less than 10 (#X [0] ... #X [9] elements. Within the loop statement IF...END_IF three different comments can be printed, depending on whether the value inside the vector element is the same as, greater than or less than 100.

This example has a nesting level of three.

3.6 Arguments listed by functions

Some functions and also the commands *DISPLAY* and *WAIT* require one or more values. These are called 'arguments '. The function *STRLEN* is used to specify a string argument . The function returns the length of the string used as a argument . For example, *STRLEN* ("Long string 16") returns a whole value of 16.

3.6.1 Functions as arguments for other functions

As *STRLEN* returns a whole value it can also be used as a argument for the function *STRCMP*. The function *STRCMP* makes a comparison between two strings and returns 0 (zero) if the strings are the same. The *STRCMP* function has the following syntax:

STRCMP (strings1, strings2, string length)

where

s1 and s2 are two strings of equal length

assuming that *s1* is the variable *$STZ1* and *s2* is the variable *$NM* the function could be expressed as:

LET #RES = STRCMP ($STZ1, $NM, STRLEN ($STZ1))

In this case the result of the comparison between the two strings can be assigned without knowing the length beforehand because this is calculated when the *STRLEN* function is compared.

3.7 'True' and 'False'

A test condition used with WHILE...END WHILE and IF...END_IF can be an expression, including any valid combination of operators and functions. For example

Condition	**Meaning**
#X = 21	is value #X the same as 21? (Note - this operation **does not** assign the value 21 to #X.)
@A < > @B	is the value @A not the same as the value @B?
#X = (#Y + #Z)	is the value #X the same as the value #Y + #Z? (**does not** assign the value #Y + #Z to #X).

The expressions return a logical value that means **true** or **false. True** is represented by a value that is different from zero (to return a true LPE value, use 1). **False**, on the other hand, is represented by a zero value. Therefore, if #A is 6 and #B is 7, the expression #A > #B will return the value zero because #A **is not** greater than #B.

The following operators are used:

Operator	**Meaning**
&	binary AND
|	binary OR
^	binary XOR
~	binary NOT
=	Equality
<>	Inequality
<	Less
>	Greater
<=	Less or the same
>=	Greater or the same

3.8 Logical operators

The operators AND, OR and NOT enable the test conditions to be combined or changed. The table below sets out the effects. (c1 and c2 show the conditions.)

Example	Result	Whole number returned
c1 AND c2	True if both c1 and c2 are true	-1
	False if both c1 or c2 are false	0
c1 OR c2	True if both c1 or c2 are true	-1
	False if both c1 and c2 are false	0
NOT c1	True if c1 is false	-1
	False if c1 is true	0

With the values of whole numbers or whole long numbers AND, OR and NOT become bitwise operators - that are different from the logical operators. -. IF #A & #B, AND function like logical operators but do not produce the same result. This should be rewritten as IF (#A<>0) & (#B<>0).

3.9 How to use more than one procedure

If a single procedure has to perform a complex task, the procedure may become long and complicated. It may be easier to divide the procedure into several events, each of which can be written and modified separately.

Many LPE events are a series of procedures that are linked to one another. Each procedure carries out a particular task (for example, a calculation) and then passes the event onto other procedures so that other operations can be run.

LPE has been designed to encourage LPE application packages to be written using this method because all the procedures that make up an application can be stored in the same file and one procedure can call up and therefore run another.

3.10 Applications containing more than one event.

An event can contain an unlimited number of procedures, each one of which must finish with END or RETURN.

When an application is run it is always the first event that is run (event 0). At the end, the event stops and any other event is run only if and when it is called up.

Although any name can be used, it is normally the first procedure that is assigned a name as a start.

The procedures that operate on their own must be written and translated into separate events, otherwise they will not be able to work.

3.11 Recalling events

To run another procedure simply enter the instruction SUB followed by the name of the procedure. For example, this event contains two procedures:

```
EVENT 1:ONE
   DISPLAY 0,0,0,15,“Start”
   WAIT 40
   *call procedure two:
   SUB DUE
   DISPLAY 0,40,0,15,“End”
   WAIT 40
   RETURN

EVENT 2:DUE
   DISPLAY 0,20,0,15,“in progress...”
   WAIT 40
   RETURN
```

If this event is run, procedure ONE starts with this result: ’start’ is displayed and after the WAIT pause it calls up procedure TWO, which displays ’in progress’. After another WAIT pause procedure TWO returns to procedure ONE and ONE displays ’End’. After the last WAIT pause procedure ONE finishes.

3.12 Use of call procedures

Call procedures can be used for:

organizing LPE application packages more clearly and making them easier to modify after they have been written;

using the same procedure in different application packages, for example for carrying out applications that are common to several different operations.

To make full use of calls procedures, the values must be communicated from one procedure to another. There are two ways of doing this: total variables and parameters.

3.13 Parameters

Values can be passed on from one procedure to the other by parameters. They resemble and function in a way that is very similar to that of the functions in relation to other functions.

In the example that follows, the PRICE procedure calls the VAT procedure. After the procedure is called, it passes on the value (that INPUTDLG has assigned to the variable x0) to the parameter p0. The parameter p0 is like a new local variable inside the VAT procedure and begins with the value that is passed on to parameter p0 when VAT is called.

The VAT procedure displays the value of x plus VAT rated at 17.5%.

```
EVENT 10:PRICE
   VARIABLES NAME=X WORD=2 REAL=0

   INPUTDLG “Test”,”Enter price:”,1,:X0,:X1
   SUB IVA,:X0
   RETURN

EVENT 11:IVA
   PARAMETERS NAME=P WORD=1 REAL=0
```

```
    DISPLAY 0,0,0,15,“PRICE INCLUDING VAT =%f”,:p0*1.175
    RETURN
```

In the invoked procedure, on the line after the name of the procedure the names appear that are to be used for the parameters and the quantity of parameters. For example:

```
    PARAMETERS NAME=PAR WORD=2 REAL=0
```

All parameters have a common base name to which the parameter index must be added. For example, a WORD parameter will be indicated as:PAR1.

Array parameters are not possible.

In the call procedure, the parameter values appear in the correct order, separated by commas after the invoked procedure. For example:

```
    SUB proc, 60 , 30.
```

The values passed onto the parameters must be variable values, strings between inverted commas or constants. A call may therefore be SUB calc,$a,:x,15.8 and the parameters line in the invoked procedure may be:

```
    PARAMETERS NAME=PAR WORD=1 REAL=1 STRING=1
```

In the invoked procedure values cannot be assigned to the parameters - for example. if p0 is a parameter, an instruction like LET p0=10 cannot be used.

3.14 Multiple parameters

In the example that follows, the second procedure VAT2 has two parameters:
the value of price x0 variable is passed onto parameter p0 ;
the value of variable x1 for the VAT rate is passed onto parameter p1.
VAT2 displays the price plus VAT at the specified rate.

```
EVENT 10:PRICES2
    VARIABLES NAME=X WORD=3 REAL=0

    INPUTDLG “Test”,”Enter price:”,1,:X0,:X2
    INPUTDLG “Test”,”Enter VAT rate:”,1,:X1,:X2
    SUB IVA,:X0,:X1
    RETURN

EVENT 11:IVA2
    PARAMETERS NAME=P WORD=2 REAL=0

    DISPLAY 0,0,0,15,“PRICE INCLUSIVE OF VAT =%f”,:p0+(:p0*:p1)/100.0
    RETURN
```

3.15 Carrying over values

In the following example, the RETURN command is used to carry over the value x0 plus VAT at x1 percent, as displayed in PRICE3. This is very similar to the way in which the functions carry over a value.

Procedure IVA3 calculates but does not display the result. This means that the result can be invoked by other procedures which must make this calculation but which do not necessarily have to display it.

```
EVENT 10:PRICE3
    VARIABLES NAME=X WORD=3 REAL=1
```

```
        INPUTDLG “Test”,”Enter price:”,1,:X0,:X2
        INPUTDLG “Test”,”Enter VAT rating:”,1,:X1,:X2
        LET @X0= CALL IVA,:X0,:X1
        DISPLAY 0,0,0,15,“PRICE INCLUSIVE OF VAT     =%f”,@x0
        RETURN

    EVENT 11:IVA2
        PARAMETERS NAME=P WORD=2 REAL=0

        RETURN :p0+(:p0*:p1)/100.0
```

The RETURN command can return only one value.

To invoke a procedure that returns a value use the CALL instruction instead of the SUB instruction. In addition, the variable to which the CALL result is assigned must be compatible with the result returned by RETURN. Otherwise, an error will be carried over to runtime.

3.16 General variables

Only one value can be displayed with the RETURN command. If more than one value needs to be carried, general variables must be defined of the type described in the chapter 'Variables and Constants' above.

Unlike the local variables that are valid only in the procedure in which they are defined the general variables can be used in any procedure.

This event will function correctly:

```
    EVENT 10:UNO
        DISPLAY 0,0,0,15,”%d”,:a
        SUB DUE
        RETURN

    EVENT 11:DUE
        LET :a=2
        DISPLAY 0,0,0,15,”%d”,:a
        RETURN
```

When this procedure is run, the value 0 and then the value 2 is displayed.

The error 'non-defined variable' would be received if a local variable were used instead of a general variable to define *:a*. This is because procedure TWO would not recognize variable *:a*. In general, however, the local variables command should be used unless it is necessary to use general commands.

3.17 Temporary values

Values can be passed from one procedure to another using general variables. Any modification to the value of a variable in an invoked procedure is automatically registered in the call procedure.

For example:

```
    EVENT 10:START
        LET :varone=2.5
        LET :vartwo=2
        SUB OP
        DISPLAY 0,0,0,15,”%d,%d”,: varone, :vartwo
        RETURN

    EVENT 11:OP
        LET :varone=:varone*2
        LET :vartwo=:vartwo*4
        RETURN
```

This will display 5 8

4. Managing data file

It is possible to use LPE to create data files (database and files). Any type of information can be stored in the data files. The data can be retrieved for display, modification or for carrying out calculations.

This chapter deals with:
the creation of data files
adding or modifying records
record searches

4.1 Types of file

Logoview can manage two types of file: history files in proprietary format (.ARC) and a database in standard DBF format.

4.1.1 History files

The history files enable variables files to be created with the greatest flexibility and transparency. Storage of variables can be managed directly by ***LOGOVIEW NT*** by means of configurable automatic procedures or directly by LPE instructions. The history files allow only numeric variables to be stored: strings are not permitted. The main features of the history files are high reading/writing speed and automatic storage of date and time for each record. As the same suggests, history files can be sorted only by date/time field and do not enable filters to be set to select the contents of the displayed file.

4.1.2 Database

The databases are files in standard DBF format and are therefore compatible with other programs such as FOX pro, Access and Clipper. The databases also enable numeric variables and strings to be stored and make file management very flexible. Data can be sorted in any manner and filters can be used to select only certain records for display. Databases are not managed automatically by LOGOVIEW: the procedures written in LPE must be used.

4.2 Files, records and fields

For example, in a data file, the names and addresses of each record may have one field for the name, one for the address, one for the telephone and different fields for each line of the address.

LPE enables:

- a **history file** to be created with *CREATE_FILE*, or an existing file to be opened by *OPEN_FILE*, and a new **record** to be added by *WRITE_REC.*
- a new **database** to be created by *DB_CREATE,* or an existing database to be opened with *DB_OPEN* and a new **record** to be added by *DB_ATTACH,* new values to be assigned to the fields by *DB_PUTREC* or *DB_PUTFIELD.*

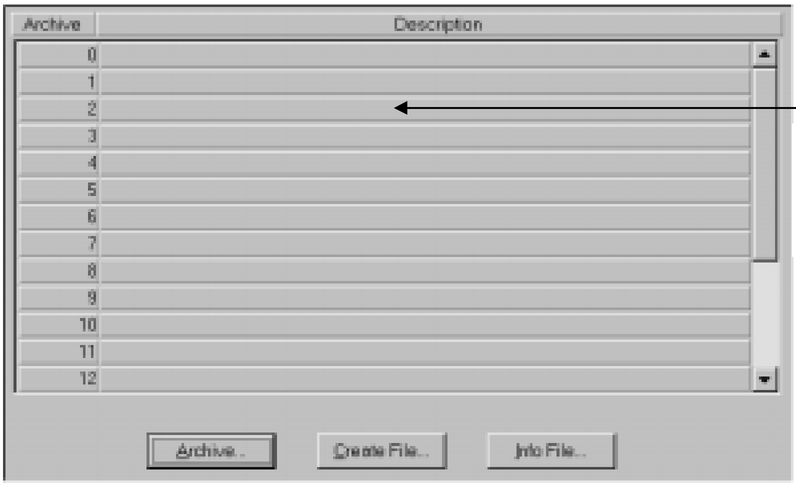
LPE also enables a file to be copied, deleted or renamed by *FILE_COPY, FILE_DEL* and *FILE_REN.*

4.3 Configuring a history file

Before being able to use a history file the record format must be configured. To define the format of a file, select ***Formats*** from the ***Display*** menu or press on the main instruments toolbar.

This action activates the window ***Formats configuration.*** If possible, configure ***History files, Database*** and ***Printout*** of ***LOGOVIEW NT***.

The ***History files*** section of this window enables 20 different types of history files to be defined, each one of which has its own format.



Archive	Description
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

List of available formats .
Double click to set up a format

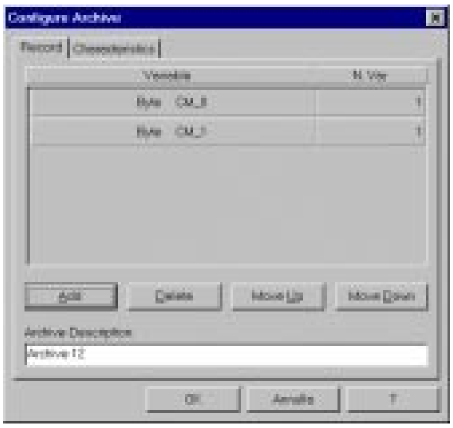
4.3.1 Configuring a record

Each history file consists of a record that is made up of blocks of variables. To configure a history file record it is necessary to specify which variables blocks must be stored.

To configure a record it is sufficient to select a file from the list and press

in the formats window.
This action displays the ***Configure Record*** window. The composition, record characteristics and updating methods of the historical record file can be configured.

The record of a file is defined by the ***Record section*** of the ***Configure file*** window. This section contains the list of the blocks of variables that will be stored.



Configure Archive

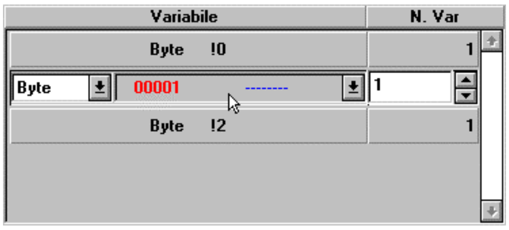
Record | Characteristics

Variabile	N. Var
Byte CM_0	1
Byte CM_1	1

Archive Description: Archive 12

To insert a variables block just press Add

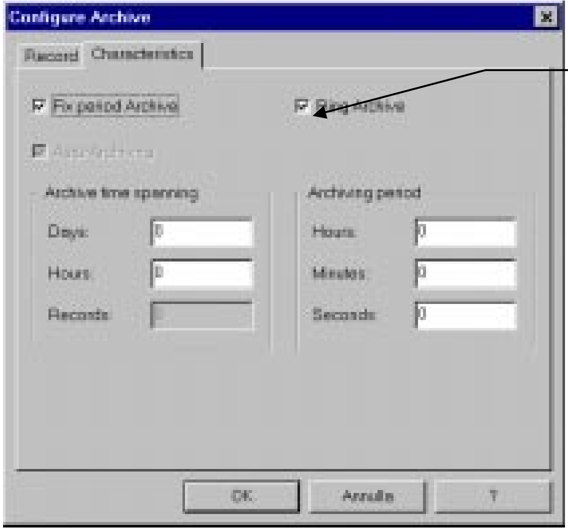
This action adds a line to the list: select it to set the required characteristics.



Variabile	N. Var
Byte !0	1
Byte 00001 --------	1
Byte !2	1

A block is a contiguous set of variables and is defined by 3 characteristics: type of variable, start offset and number of variables to achieve starting with start block. To set these three characteristics ***LOGOVIEW NT*** provides 3 combo-boxes when a line is selected. The first two are combo-boxes containing respectively the list of available types and the list of variables defined for that type.

After configuring the record the general file characteristics can be set. To carry out this operation select the section ***Characteristics*** in the ***Configure File*** window.



Configure Archive

Record | Characteristics

☑ Fix period Archive ☑ Ring Archive

☑ Auto Archive

Archive time spanning
Days: 0
Hours: 0
Records: 0

Archiving period
Hours: 0
Minutes: 0
Seconds: 0

File characteristics

These characteristics can be set in this window:

- Fixed-period file. When LOGOVIEW is activated it creates a record for each storage period independently of whether there are stored data. The records that do not contain data are marked by LOGOVIEW as empty and are not displayed. For example, if the computer is switched off for a certain time the next time that it is switched on LOGOVIEW realizes record storage and before adding data to the file will fill the file with the missing records.

This type of configuration is kept because of its compatibility with LOGOVIEW 32 but it is not advisable to use it with LOGOVIEW NT because search speed is no quicker and it takes up more space on the disk.

- Automatic storage. When this configuration is active LOGOVIEW stores new records completely automatically according to the parameters for the storage period in the window. Warning: automatic storage will not take place until the file is opened by an instruction *CREATE_FILE* or *OPEN_FILE* and will cease immediately the instruction *CLOSE_FILE* is run.
- Circular file. This characteristic enables growth of the file to be kept within a set limit. If the file has a 'fixed period' the duration can be set in terms of hours, minutes and seconds. Otherwise, the maximum number of records that that may be stored in the file must be set. When the maximum number has been reached LOGOVIEW will start to overwrite the older records.

4.4 Creating a history file

Use the command *CREATE_FILE* as follows:
CREATE_FILE name of file, file number, error var.
For example:
CREATE_FILE “history ”, 0, :err
creates a data file “history file”.
The file name is a string. It must therefore be written between inverted commas. The name string can also be assigned to a variable string (for example, *LET* $file=”history ”) and can then use the name of the variable as a argument - *CREATE_FILE* $file, 0, :err.
In the previous example no path was in the file name so the file was created in the LOGOVIEW application directory. To create a file in a specific directory the entire path must be specified:
CREATE_FILE “c:\files\history ', 0, :err
Up to 20 files can be open simultaneously. Each of them must refer to one of the file formats defined in the formats configuration window described above.

4.5 Opening a file

When using *CREATE_FILE* to create a history file the file is automatically opened but it is closed again when the application finishes or when a *CLOSE_FILE* command is given. **If an attempt is made to use *CREATE_FILE* to create a file that already exists, the old file will be deleted and the contents will be lost permanently.** To open an existing file, use the command *OPEN_FILE.*
OPEN_FILE “history ” , 0, 1, :err
Use the file name that was given to it when it was created.

Below, three events are shown for opening a file adding records and rereading them:

```
EVENT 10:openfile

*Opens file
OPEN_FILE “example”,0,0,:err
* Add 10 records
SUB write,10
•   Position the reading pointer on first record
SEEK_REC 0,:err,0,0
```

```
* Reads 10 records
SUB read,10
CLOSE_FILE 0
RETURN

EVENT 11:write
PARAMETERS NAME=PAR WORD=1 REAL=0
VARIABLES NAME=LC WORD=2 REAL=0

LET  :lc0=0
LET  :ris=0
WHILE  (:ris<>0) & (:lc0<:par0)
    * Add record to file
    WRITE_REC 0,:ris
    IF (:ris<>0)
              EXIT_WHILE
    END_IF
    LET :lc0=:lc0+1
 END_WHILE
RETURN

EVENT 12:read
PARAMETERS NAME=PAR WORD=1 REAL=0
VARIABLES NAME=LC WORD=2 REAL=0

LET  :lc0=0
LET  :ris=0
WHILE  (:ris<>0) & (:lc0<:par0)
    * Reads contents of record in variables var. LW
    * and goes on to next record
    READ_REC  0,:ris,!b100,#w100,?t100,@r100,:d100
    LET :lc0=:lc0+1
END_WHILE
RETURN
```

4.6 Notes

Opening/creating file
In this case in the *OPEN_FILE* a 0 has been inserted as a third parameter so that if the file does not exist it is automatically created.

Date and time of current record
Each time that a record is read by *READ_REC* Logoview inserts into system variables r_day, r_month, r_year, r_hour, r_minutes, r_seconds recorded when the record was stored.

Destination variables
The destination variables are passed on as a parameter to *READ_REC* and show LOGOVIEW the first variable from which the contents of the record will be copied. In the above example 10 variables are read, starting with the variable @r100.

Result of *READ_REC*
The *READ_REC* result not only shows errors but can also be used to find out when the end of the file has been reached (result =2) or to find out if the read record is valid.

Automatic advance

After the current record has been read *READ_REC* automatically positions the file on the next record in order to prepare it for a subsequent *READ_REC*.

4.7 The record number

The instruction *GET_REC* sets out the number of records contained in the file. If it is used immediately after creating a file, it will have the value 0. As records are added the count will increase.

4.8 How values are saved

Use the *WRITE_REC* command to save a new record. The parameters required for this command are the file number (the file must have been opened) and a variable for the error code. **The new values are always added to the end of the file** as the latest file record (if the file is new it will be the first record). Inside the file the values are always saved in binary form.

4.9 How to move from one file to another

When a file is opened or created the first record in the file is the current one. To read or modify another record, update the required file, i.e. select the record.
Only one record at a time is current. To change the current record, use one of the following commands:
SEEK_REC “selects' a record so that the next reading or writing operation starts with that record. For example, the instruction *SEEK_REC* 0,:err, 3,0 makes record 3 current (the first record is record 0).
The instruction *GET_REC* enables the user to find out the name of the current record.

4.10 How to find a record

FIND_REC seeks a record with a specified time and date inside the file.
For example *FIND_REC 1,#stat,24,5,1996,10,0,0*
seeks the record dated 24 May 1996 1000 hours inside the file in #*stat* and returns the result of the search. If the returned number is zero it means that the record has been found and has become the current record. If 1 is returned from #*stat* the record has not been found.

4.11 Closing a file

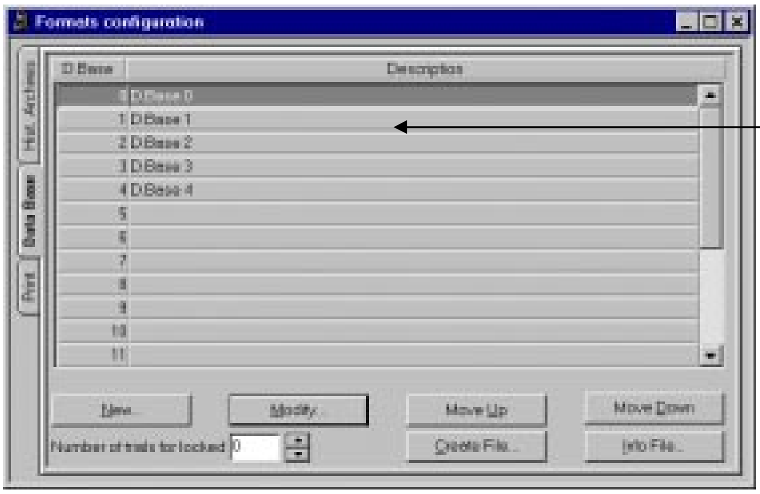
After using a data file it must be 'closed' by the command *CLOSE FILE*. The other files are closed automatically when the application finishes.

4.12 Configuring a database

The databases can also be used without configuring the record format. In this case some restrictions must be observed: databases cannot be created for which the record has not been configured and the instructions *GET_REC* and *PUT_REC.* cannot be used. It is therefore advisable to configure the record.
To define the format of a database select ***Formats*** from the ***Display*** menu or press the button.
on the main toolbar

This action opens the window ***Configure formats.*** From here ***LOGOVIEW NT's History files***, ***Database*** and ***Print*** can be configured**.**

The **Database** section of this window enables an unlimited number of Databases to be defined, each one of which has its own format.



Formats configuration

D Base	Description
0	D Base 0
1	D Base 1
2	D Base 2
3	D Base 3
4	D Base 4
5	
6	
7	
8	
9	
10	
11	

Number of trials for locked: 0

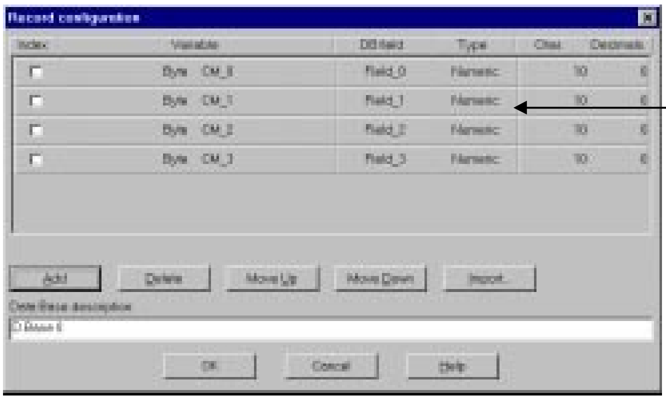
Formats available
Double click to configure a record

4.12.1 Configuring a record

Each database consists of records that are made up of fields. To configure a database record specify the variables that are to be stored and in which files they must be stored.

To add a record, just press the button New... in the format window.

This action displays the window ***Configure Records*** in which the record can be configured.



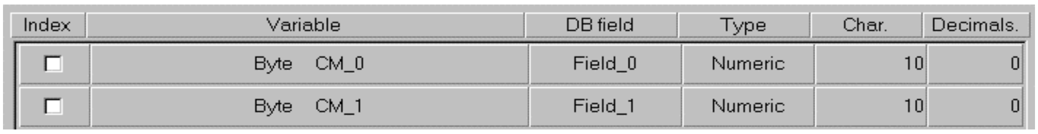
Record configuration

Index	Variable	DB field	Type	Char.	Decimals.
□	Byte CM_0	Field_0	Numeric	10	0
□	Byte CM_1	Field_1	Numeric	10	0
□	Byte CM_2	Field_2	Numeric	10	0
□	Byte CM_3	Field_3	Numeric	10	0

Data Base description: D Base 0

List of fields and variable of the record.

To insert a variables block, press Add
This action adds a line to the list: select it to set the required characteristics.



Index	Variable	DB field	Type	Char.	Decimals.
□	Byte CM_0	Field_0	Numeric	10	0
□	Byte CM_1	Field_1	Numeric	10	0

For each field, specify the following parameters: index for that field (if required); name of support variable, name of field, type of field, maximum number of characters that can be stored in the field, the number of digits to the right of the decimal point (in the case of a numeric field). It must be remembered that for some types of operation (e.g. searches) the field must be indexed. Each record can contain up to 255 fields, of which 47 are indexed.

4.13 Creating a database

Use the command *DB_CREATE* as follows:
DB CREATE file name, format number, handle var., error var.
For example:
DB CREATE “clients”, 0, :hnd, :err
creates a file called “clients.dbf”. This contains the actual data and also creates the “clients.cdx” file that contains the indexes of “clienti.dbf”.
The file name is a string. It must therefore be placed between inverted commas. The name string can also be assigned to a string variable (for example *LET* $file=”clients”) and the name of the variable can then be used as a argument - *DB CREATE* $file, 0, :hnd, :err. The instruction *DB CREATE* has a final optional parameter. If it is current and the same as '1' the file can be shared with other programs that may be LOGOVIEW programs or programs of other parties that are able to read DBF files. For example, this instruction creates a shared database: *DB CREATE* “clients”, 0, :hnd, :err, 1
In the above example no path was specified in the file name so the file was created in the LOGOVIEW application directory. To create a file in a specific directory, specify the entire path:
DB CREATE “c:\files\clients”, 0, :hnd, :err

4.14 Handles

Up to 128 databases can be open at any one time. Each time that a new LOGOVIEW database is created or opened LOGOVIEW assigns an identifier to the database in question. This identifier is called a **handle** and must be set as a parameter for all the instructions that operate on databases in order to establish the database on which the operations should be carried out.
When a file is closed, its handle is freed, so that it can be used by another file.

4.15 Opening a file

When a database is created by *DB_CREATE* it is opened automatically but it is closed again when the application comes to an end or when a *DB_CLOSE* is carried out. **If the user tries to use *DB_CREATE* to create an already existing file the old file will be deleted and the contents will be permanently lost.** To open an existing file, use the command *DB_OPEN.*
DB_OPEN “clients”, 0, :hnd, :err
Use the same file name as when it was created.

Below, three events can be seen for opening a database, adding records and rereading them from the end of the database.

```
EVENT 10:openfile

* Opens the DB
DB_OPEN “example”,0,:hnd,:err
*Add 10 records to file
SUB write 10
* Place reading/writing pointer on last DB record
DB_BOTTOM :hnd,:err
* Reads 10 records
SUB read,10
* Close DB
DB_CLOSE :hnd
```

```
RETURN

EVENT 11:write
PARAMETERS NAME=PAR WORD=1 REAL=0
VARIABLES NAME=LC WORD=2 REAL=0

LET  :lc0=0
LET  :ris=0
WHILE  (:ris<>0) & (:lc0<:par0)
    * Adds empty record
    DB_ATTACH :hnd,:ris
    IF (:ris<>0)
            EXIT_WHILE
    END_IF
    *initializes empty record with contents of LW variables
    DB_PUTREC  :hnd,:ris
    LET :lc0=:lc0+1
 END_WHILE
RETURN

EVENT 12:read
PARAMETERS NAME=PAR WORD=1 REAL=0
VARIABLES NAME=LC WORD=2 REAL=0

LET  :lc0=0
LET  :ris=0
WHILE  (:ris<>0) & (:lc0<:par0)
    * Reads contents of message current in LW var.
    DB_GETREC  :hnd,:ris
    *Goes to previous record
    DB_SKIP :hnd, -1,:ris
    LET :lc0=:lc0+1
END_WHILE
RETURN
```

4.16 Notes

Sharing file

An optional parameter has been omitted from *DB_OPEN*. In this case the file is opened in the exclusive mode. This means that if the file is already being used by another program it cannot be opened. Similarly, once the file has been opened it cannot be used by other programs until it is closed by LOGOVIEW. In order to share a database with other programs a parameter must be added to *DB_OPEN:*

```
DB_OPEN “example”,0,:hnd,:err,1
```

Once a file has been opened it can also be used by other programs provided that they are able to share it.

The same consideration also applies to the instruction *DB_CREATE*.

Destination/source variables

For the sake of simplicity, the *DB_PUTREC* and *DB_GETREC* instructions have been included. They write and read the variables that have been configured in the DB configuration window. If variables need to be read or written that are different from those configured the instructions *DB_PUTFIELD* and *DB_GETFIELD* must be used to read and write each single DB field.

Automatic feed

Unlike the history files, the databases have no automatic feed so for all shifts special functions must be used such as *DB_SKIP*.

Flushing

The current record is stored in the memory by LOGOVIEW and all modifications to the record are stored in a temporary buffer until one of the following events occurs:

- a *DB_FLUSH*
- an instruction is run that modifies a current record (for example, a *DB_SKIP*)
- the database is shut down

In one of these cases the modifications are written onto the file and are therefore also visible to other programs that are using the same database. In addition, the instruction *DB_FLUSH* forces the operating system to write on the disk and updates the database header so that the no database will be damaged if the computer shuts down unexpectedly. The *DB_FLUSH* is the equivalent of closing and reopening the database.

Warning: if the computer is shut down after records are added with *DB_ATTACH* but no *DB_FLUSH* there is a strong possibility that the database is damaged. The only way to recover the data is to protect the database with one of the utilities that are commercially available. If the damage is limited to the index file an attempt can be made to recover the data by deleting the file and running LOGOVIEW again. When LOGOVIEW does not find the file it will try to recreate it.

4.17 Deleting records

Records can be deleted from the database using the instruction *DB_DELETE*. This instruction removes the current record but in fact the record is not physically eliminated from the file. It is only marked as being deleted. The instruction *DB_PACK* physically eliminates all records marked as deleted. However, this instruction should be used only when really necessary because the physical elimination of deleted records is rather a slow procedure that involves copying the whole file. It is better to open the database exclusively before running *DB_PACK*.

4.18 The record number

The instruction *DB_GETSTAT* shows the number of records contained in the file. If it is used immediately after creating a file it will have the value 0. As records are added, the count will increase. Warning: the number returned by *DB_GETSTAT* is the total number of records in the file, regardless of whether there are any deleted records or whether the current tag is running file filtering. There is no way of knowing how many undeleted records are current in the file or how many records have been selected by a given tag. The only way of finding out about these two data items is to scan the entire file and to count the number of records current. For example, the following code counts how many records have been selected by the ’TEST’ tag.

```
* gen. variable:hnd contains the handle of the DB
* gen. variable:count contains the record number
LET :count=0
DB_TAGSELECT :hnd,”TEST”,:ris
DB_TOP :hnd,:ris
WHILE  (:ris<>0)
   DB_SKIP :hnd, 1,:ris
   LET :count=:count+1
END_WHILE
```

4.19 How values are saved

Use the command *DB_PUTREC* to save a new record. This command uses the database handle as parameters (the database must already have been opened) and a variable for the error code. **The new values always overwrite those in the current record.** In order to add new data to the bottom of the database the instruction *DB_ATTACH* must first be used to add an empty record. Then run *DB_PUTREC* without modifying the current record. Unlike the files inside the databases they are always stored in text format. The numeric fields are converted into a text and are then written onto the file. This means that the numeric fields can be read both on numeric variables and on string variables.

4.20 Sorting the database

The aim of a database is to organize the data in such a way that they can be easily retrieved. We have already seen that the data are divided into fields and records. We will now see how the records can be sorted inside the database. The simplest way of sorting data inside the database is to physically move the data inside the file. Unfortunately, the simplicity of this system means that a large quantity of data is moved from one part of the file to another. In addition, it allows for only one sorting method.

4.20.1 Indexes

A preferable system is to leave the records in place, as they were inserted in the database and to use an additional file known as *INDEX* to sort the file. When an index file is created the file is in fact sorted. Index files can be managed with great efficiency. An unlimited number of indexes can also be permanently available. LOGOVIEW NT uses .CDX index files, which are compatible with FoxPro 2.0.

4.20.2 Tag

Each index file may also comprise up to 47 different sorting methods. Each method is identified by a label known as a *tag.* All the open index files are automatically updated by ***LOGOVIEW NT*** each time that the database is modified.

OBJECTS.DBF (DATA File)

Rec.	***Item***	***ID***	***Price***
1	*Screw*	*27*	*440*
2	*Bolt*	*64*	*200*
3	*Nail*	*33*	*100*
4	*Bracket*	*99*	*2000*
5	*Wrench*	*12*	*5000*
6	*Hammer*	*54*	*3000*

OBJECTS.CDX (Index file)

ITEM_TAG	***ID_TAG***
Bolt	*12*
Wrench	*27*
Nail	*33*
Hammer	*54*
Bracket	*64*
Screw	*99*

↑ *Tag* ↑ *Tag*

PRICE.CDX (Index file)

PRICE_TAG	
100	← *Key*
200	← *Key*
440	← *Key*
2000	← *Key*
3000	← *Key*
5000	← *Key*

↑ *Tag*

In the above example we have the file OBJECTS.DBF. This contains the effective data. There are also two index files, OBJECTS.CDX and PRICE.CDX. The first contains two *tags* that enable the data file to be viewed sorted according to OBJECT or ID. The second contains just one *tag* and enables the data file to be viewed sorted by PRICE. Each element of the index file is known as a ***KEY.***

4.20.3 Index expressions

The method of sorting an index file is defined by a database expression that is used to generate the index keys. An *index expression* must be evaluated as a CHARACTER type or as a NUMERIC type. The most commonly used expression consists only of the field name.

Other commonly used names generate tags that are based on two or more fields. For example: “OGG_TAG+ID_TAG”

Apart from chaining in database expressions many other functions can be used. For example, the key can always be evaluated with upper case characters: "UPPER(OBJECT)". Similarly, other functions can be used to convert the numeric fields into character fields or vice versa. If required, tag sorting order can be inverted: the keys appear in reverse order compared with the normal sorting order.

4.20.4 Filter expressions

Whilst the index expression determines the order in which the records must be stored, the filter expression determines which records should be viewed and which should be excluded by the tag. It is therefore obvious that the filter expression must give a result of the type TRUE/FALSE. The expression is evaluated for each record and if it gives the result TRUE the record is included in the tag. On the other hand, if the result is FALSE, the record is excluded. One example of a filter record is the following: “Price<1000 .AND. .NOT. DELETED()”. This expression selects all the records that have never been deleted and in which the field *Price* contains a value below 1000. For further details on the expressions, see the relevant chapter in the Guide to the use of LOGOVIEW NT.

4.20.5 Suggestions

By default, when LOGOVIEW opens a database the CDX file has the same name. It is nevertheless possible to open other CDX files from the event, depending on individual requirements. It should be remembered that whenever a LOGOVIEW database is modified it must also update all the tags relating to that database so the greater the number of tags the longer LOGOVIEW will take for modify the database. If large-scale database modifications are to take place, it is advisable to close all the index files, make the modifications and then recreate the indexes. It must also be stressed that the closed index files are not updated so if the database is modified the index files will no longer reflect the contents of the database so that if the file is then used this could lead to unforeseeable results.

4.20.6 Instructions for tags and indexes

To open other index files in addition to the default file use the instruction *DB_OPENIDX*. To create a new file, use the instruction *DB_CREATEIDX*. For example, to create a new tag the following line can be used:

```
DB_CREATEIDX
“new”,:hnd_db,:hnd_idx,:er,”RECDEL”,”FIELD1+FIELD2”,”DELETED()”
```

This instruction creates a new index file called “new.cdx”. Inside the tag “RECDEL” is created. This is sorted by chaining the two fields FIELD1, FIELD2. The instruction selects only deleted records. Note that the instruction requests the handle of the db to which the index should be assigned and returns a handle. The index files can be closed manually by the instruction *DB_CLOSEIDX* or automatically when the assigned db is closed. In both cases the handle returned by *DB_OPENIDX*, *DB_CREATEIDX* will no longer be valid. To select one of the tags inside one of the open index files use the instruction *DB_TAGSELECT*. On the other hand, the instruction *DB_TAGSELECTED* returns the name of the current tag.

4.21 How to move from one record to the next

When a file is opened or created, the first record in the file is the current one. To read, modify or delete another record, make it the current record. In other words, place the pointer on the record.

Only one record at a time can be current. To change the current record, use one of the commands illustrated below.

DB_TOP: place the pointer on the current record on the first record

DB_BOTTOM: place the pointer on the current record on the last record

DB_SKIP: enables the pointer to be moved to the current record in relation to the present position. For example, it is impossible to move the pointer forward by 10 records or back by 1 record.

DB_GO: moves the current record onto any database record by specifying its order number, i.e. its absolute position inside the file regardless of the type of sorting selected.

Note that instructions *DB_TOP*, *DB_BOTTOM*, *DB_SKIP* use the current tag whilst *DB_GO* completely ignores the selected tag and enables the current record to be positioned on any record, including those that have not been selected by the tag.

The instruction *DB_GETSTAT* enables a series of information to be obtained such as, for example, the number of the present record, the number of records in the database, the number of fields and the status of the End Of File (EOF) and Beginning Of File (BOF) flags.

4.21.1 EOF and BOF

Logoview has two flags for each database. They are called End Of File (EOF) and Beginning Of File (BOF). They are used to show whether the current record is valid. If EOF has been set it means that an attempt has been made to go beyond the last record (e.g. with a *DB_SKIP*) and the current record is not valid. Similarly, BOF is set when an attempt is made to go beyond the first record with a negative *DB_SKIP*. In this case again, the current record is not valid.

4.22 How to find a record

DB_SEEK looks for a record inside the database with a key that has a set value. DB_SEEK works exclusively on the current tag and generally enables a key to be found if only the key is specified in the index expression that corresponds to the contents of that field. On the other hand, if several records are specified, the key will also be a composite key and a composite key will also have to be specified in DB_SEEK. Suppose that in the above example there was a database with two fields, ’OBJECT’ and ’PRICE’, both being text fields that are 10 characters in length. In the default index file we have two tags with the same name as the fields that have just one field as an index expression. If I need to seek a record on the basis of the contents of the ’OBJECT’ field I must first select the tag and then carry out the search.

```
DB_TAGSELECT :hnd, ”ITEM”,:err
DB_SEEK :hnd,”Screw”,:err
```

If DB_SEEK has found the record in :err it returns 0. Otherwise, it returns an error code. If the tag contains more several examples of the 'Screw' key, the others can be sought by using the instruction:

```
DB_SEEK :hnd,”Screw”,:err,1
```

If we now create another index file with the index expression 'OBJECT+PRICE' we will have a key comprising two fields and we will therefore also have to specify a composite key for the searches.

```
DB_SEEK :hnd,”Screw     440”,:err
```

As can be seen, spaces have been left after “Screw”. This is because the key is always made up of the total number of characters in the field (10 in our case) and as “Screw” contains fewer words than ten words spaces must be added until the field dimension is reached. The previous search seeks a key whose "OBJECT" field is the same as “Screw” with a “PRICE” field that is the same as “440”. If we wish to look for any item with a price of 440 we just need to specify a string in which there is the number 440 preceded by 10 spaces.

```
DB_SEEK :hnd,”          440”,:err
```

4.23 Locking

When a database is used by several programs it is fundamentally important to prevent the file from being updated simultaneously by several programs. To prevent this situation, LOGOVIEW automatically locks the record that is to be modified. It also locks index files that require updating. After writing onto the disk the files are automatically freed. In addition to automatic lock management manual locking is also possible using the instructions *DB_LOCK* and *DB_UNLOCK*. *DB_LOCK* denies access to a single record, the entire data file or to the data file and all the index files. On the other hand, *DB_UNLOCK* eliminates all the locks on the database. It is of fundamental importance for every *DB_LOCK* to be followed by the corresponding *DB_UNLOCK*. The database will remain locked and no other program will be able to access it.

When LOGOVIEW requests that a database be locked but does not manage to obtain the lock, keep on trying until it does lock. This behavior can be modified by setting a value other than zero in the field 'No of DB lock attempts' in the window '**Configure formats**', “**DataBase**” file. If a number other than zero is set, LOGOVIEW makes a limited number of attempts and then returns an error to indicate that it was unable to complete the operation.

4.24 Closing a file

After using a data file it must be ’closed’ by using the command *DB_CLOSE*. The databases are closed automatically when the application is terminated.

5. Text files

In addition to the history files the Logoview NT databases enables simple ASCII files to be read and written, i.e. text files organized into separate lines by CR/LF.
To open or create a text file the instruction *FOPEN* must be used. This instruction returns a handle that can then be used with the other instructions for the text files. In the following example a file is opened and the contents are read and copied onto another file:

```
FOPEN “sample.txt”,0,:hin
FOPEN “copy.txt”,1,:hout
WHILE (1)
   FREADLN :hin,$temp,:err
   IF (:err<=0)
•  leaves loop if there is an error reading or if 0 bytes are read.
   EXIT_WHILE
   END_IF
   FWRITE :hout,$temp,:err
END_WHILE
FCLOSE :hin
FCLOSE :hout
```

The FOPEN instruction enables both an existing file to be opened and a new one to be created. The required behavior can be obtained on the basis of the value of the second parameter. The simplest way of reading a text file is to read one line at a time with the instruction *FREADLN* .This instruction reads an entire line until the 'new line' character (NEWLINE= ASCII code 10) and places the line in a string variable. Remember that the last character in the string will be the NEWLINE character. Instead of the instruction *FREAD* a set number of characters can be read regardless of the line structure of the text.
The only writing function is *FWRITE*. This enables a fixed number of characters to be written. To start a new line, write the character NEWLINE into the text manually. As this character cannot be keyed in, alternative methods must be used. With LOGOVIEW two methods can be used:
First method: enter NEWLINE character using the instruction *STRPRINT*

```
STRPRINT $temp,”New line%c”,NEWLINE
FWRITE :hnd,$temp,:err
```

Second method: convert the sequence \n into NEWLINE characters with the instruction *CONVERT_ESC*

```
LET $temp=”New line\n”
CONVERT_ESC $temp,0
FWRITE :hnd,$temp,:err
```

The text files do not allow the reading/writing pointer to be moved at will. It can only be placed at the beginning or end of the file. The instruction *FSEEK* enables both the reading/writing pointer to be moved and file status to be obtained if error or End Of File are current.
After use the text files must be closed by the instruction *FCLOSE.*

6. Operations on files

In addition to functions for reading and writing text files, files and databases LOGOVIEW also comprises general file management functions. First of all there is the instruction FILEDLG. This enables a standard window to be opened for choosing the name of a file. For example, FILEDLG enables the user to choose which database to open.

```
FILEDLG “Open DataBase”,”File DBF|*.dbf||”,”dbf”,””,3,$name,:res
```

This function opens the window, waits for the user to select and then returns the selected file to $name.
Other functions include:

FILE COPY: enables a file to be copied
FILE REN: enables the name of a file to be changed
FILE DEL: enables a file to be eliminated.

Another function connected to the files is EXEC. This instruction enables an external program to such as File Manager or Notepad to be run:

```
EXEC “notepad prova.txt”,:err
```

This instruction starts Notepad editing on the prova.txt. file. The instruction EXEC accepts an additional parameter that enables the status of the window of the run program to be changed. For example, a program can be run that has been maximized or reduced to an icon.
The EXEC instruction can also be used to run a batch file:

```
EXEC “cmd /c mybatch.bat”,:err
```

or to open the control panel:

```
EXEC  "control timedate.cpl",:err
```

In this case the icon opens to modify dates and time.

7. *Error management*

7.1 Syntax errors

Syntax errors are errors that are reported whilst a procedure is being translated. (Other errors may occur whilst a program is being run). The LPE translator will go to the line where the first syntax error was detected.

Each programming language has its own method for using the commands and functions, especially the method in which a program is written. A series of errors that can easily be made using LPE are set out below.

Errors	Notes
Parentheses not symmetrical	Round closing parenthesis missing “)”
Syntax error	This error is reported when LOGOVIEW cannot accurately identify the error and means that the correct syntax was not used to write the parameters.
Undefined variable	The variable indicated in the message has not been found among the general or the local variables.
Asymmetrical inverted commas	A “ character is missing at the end of a string.
Variable requested	A numeric expression has been requested in a parameter that requires only one variable.
Square closing parenthesis missing	In one of the closing parentheses a square closing parenthesis “]” is missing
Function different from requested type.	A function has been used that returns a REAL value inside an expression that requires INT values such as the index expression inside square parentheses.
Parentheses required after the function	When a function requires parameters round parentheses must be used after the function name: e.g. *ABS(1.1)* and not *ABS 1.1*
REAL constant too large or too small	A constant has been used that is outside the range of the values permitted for REAL variables.
STRINGA required	A numeric expression has been used in a parameter in which a string was required.
$$ not supported by Logoview NT	Operator $$ is not supported by Logoview NT
Numeric variable required	Numeric variable or string has been used in a parameter in which a single numeric variable is required.
STRING variable required	Numeric expression or string constant has been used instead of STRING variable.
Numeric expression required	String has been used in a parameter in which a numeric expression is required.
String expression required	A numeric expression has been used in a parameter in which a string is required.
Parameters missing from function	Fewer parameters than those required have been specified in a function.
Memory insufficient for compilation	Memory insufficient for normal

	LOGOVIEW operations.
Assignment to a parameter is not permitted.	It is not permitted to assign a new value to one of the parameters of an event.
Incorrect separator character	An unknown character has been found at the end of a parameter. Use only commas to separate parameters.
Insufficient number of parameters	Fewer parameters have been entered than those required by the instruction.
Assignment symbol '=' missing	Assignment symbol missing from LET instruction..
Non-defined label	Reference has been made to a non-defined event label.
Unknown identifier	An identifier has been used whose variable name and function are not known.

7.2 Errors in running procedures

LPE can display an error message and stop a program being run in 'error conditions'. This may occur because:

- there is an error or bug in the program that was not recognized during the translation phase, e.g. a calculation involving division by zero.
- The structure of the program is incorrect. For example, the RETURN to an event is missing or an END_IF or an END_WHILE have not been positioned correctly.
- A problem has occurred that prevents a command or a function from operating. For example, an OPEN FILE command may fail because the file does not exist.

In all these cases a window will open that displays an error message and offers the possibility of continuing or interrupting runtime. In some cases it will not be possible to continue runtime and LOGOVIEW will have to terminate it. The error windows can also be disabled by altering runtime configuration ('Runtime configuration' from the 'Options' menu). All the error windows can be disabled or only those containing the instructions on the history files or the databases. In all cases the errors that occur during runtime are displayed in the window with the errors list.

8. Events activation

After the application starts event 0 is run automatically. Apart from calls to a subroutine with the instruction SUB and CALL all the other events run only in certain circumstances:

- time assigned to the event has elapsed
- the timer assigned to the event has changed status
- reading of data block by PLC
- change in alarm status
- button activated by user
- editing of interactive field
- editing of table cell
- selection of menu item
- modification of animation variable

All the circumstances listed above cause a new task to be activated that starts to run the event concurrently with any other events in progress. In this chapter 'running an event' refers not only to the event but also to those invoked by the SUB and CALL instructions. For example, if a timer activates the event:

```
EVENT 10
        …
        SUB TEST1
        SUB TEST2
        …
        RETURN
```

this event is considered 'to be running' until the instruction RETURN is reached and is therefore considered to be still running during TEST1 and TEST2 procedures.

8.1 Multitasking

Logoview has a multitasking event running system. This means that several events can be run at the same time. Obviously, the instructions are not run simultaneously. Instead, LOGOVIEW rapidly alternates between running the instructions for different flows so as to give the impression of simultaneously running instructions. Each running flow is called a **task.** Logoview has a range of tasks:

- task 1: starts to run event 0 when the application starts
- tasks 2–8: are for direct management by the developer.
- tasks 9-35: are for running events activated asynchronously. Each time that an event occurs that requires an Logoview event to be activated Logoview seeks a free task between 9 and 26 and activates the event. If there is no free task the event that is to be activated is queued and as soon as a task is free the event is run.

Note that there is a clear distinction between an event and a task. The term 'event' identifies a procedure whilst an event indicates a flow that is activated by an event. It is not unusual for an event to be run by several flows simultaneously.

8.1.1 Activation

As already mentioned, at the start of the application Logoview creates task 1. This starts to run event 0. Other tasks can be created by controlling the application with the TASK instruction.

There is then a series of circumstances (listed at the start of the chapter) in Logoview that lead to the events being activated that have been configured in the application. Each time that one of these circumstances occurs Logoview seeks a task to run the configured event.

Some event activation mechanisms stop several tasks being activated simultaneously on the same event. On the other hand, in other cases this must occur. A typical case in which Logoview prevents several tasks from being activated is that of the timers. In the case of a timer it is essential that the operation occurs once only when the set time elapses. However, on occasions running the event can take longer than the time set by the timer. In these circumstances Logoview has to again activate the event before the current event has terminated. This could cause different problems. For example, the unit running the event might crash. If the event takes longer than the set time this event will be reactivated as soon as it has terminated.

In all other cases it is perfectly legitimate and is in fact necessary that one event runs several tasks simultaneously. This may happen, for example, when an event is assigned to an alarm class. In this case the event will be activated whenever alarm status for that class is modified. Given the nature of the alarms it is not uncommon for several alarms to be set off almost at the same time so it is vital that the same event can be run by several tasks at the same time.

8.1.2 Termination

A task finishes when the event on which it has been activated reaches the instruction RETURN or END so that no other instructions need to be run.

An event can use the instruction KILL to terminate tasks created with the instruction TASK. However, this method is not recommended because the exact moment in which the instruction is run cannot be predicted so the sudden termination of a task could leave the task in an unstable condition. Let us suppose that the task in question has locked a database: if it is terminated before the database is unlocked access to the database will be blocked for all the other programs until another UNLOCK happens to occur.

8.1.3 Task status

A task can assume different statuses. These statuses are normally used internally by Logoview but it may be useful to become familiar with them in order to have a clearer idea of how they function.

Running: this status identifies the only task that is running at a given moment. Only one task at a time is in running status.

Ready to run: This status identifies tasks that are ready to run and which are waiting to be run by the Logoview scheduler. During normal Logoview operations each active task moves from "Ready to Run" status to "Running" status. It is run for a certain amount of time and then returns to 'Ready to Run' and leaves the CPU free for other tasks. This cycle is repeated for the whole life of the task.

Suspended: This status identifies suspended tasks. These tasks are waiting for an external event, which might be a semaphore, a timer or a long operation that is in progress. For example, the instruction WAIT that is used to cause delays in running events merely activates an internal timer and then **suspends** the current task. When the time set on the timer elapses the task is reactivated by turning 'suspended' status into 'ready to run' status.

8.1.4 Synchronization

In all multitask development environments one of the fundamental aspects is synchronizing operations between different parts of the application. Synchronization requirements are basically due to the fact that access to certain resources must be restricted to one task at a time. For example, let us suppose that two tasks must add records to the same database: if there are no synchronization mechanisms, records might be unexpectedly overwritten (the parameters have been omitted for the sake of simplicity):

```
EVENT 10
        DB_ATTACH
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        RETURN

EVENT  11
        DB_ATTACH
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        RETURN
```

If events 10 and 11 are run simultaneously, a sequence of this type might occur:

```
EV 10: DB_ATTACH       < Event 10 adds a record
EV 10: DB_PUTFIELD     < starts to fill the fields
EV 10: DB_PUTFIELD
EV 11: DB_ATTACH       < event 11 adds another record
EV 11:DB_PUTFIELD      < starts to fill the fields
EV 11:DB_PUTFIELD
EV 10:DB_PUTFIELD      < returns to run event 10 but now fills
EV 10:DB_PUTFIELD      < record added by event 11 !!
```

This type of sequence leads to a partially initialized record whilst another record will be initialized with incorrect data. To prevent situations of this type Logoview provides two synchronization mechanisms: LOCK/UNLOCK and SET SEM/RESET SEM.

The instruction LOCK stops other tasks being run until UNLOCK is run. This guarantees that the sequence of instructions between LOCK and UNLOCK cannot be interrupted by other tasks.

The previous example could be rewritten thus:

```
EVENT  10
        LOCK
        DB_ATTACH
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        UNLOCK
        RETURN
```

```
EVENT  11
        LOCK
        DB_ATTACH
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        UNLOCK
        RETURN
```

By re-writing the events in this way the user can be sure that the operations on the database will always be carried out in the correct order. The instruction LOCK does have one disadvantage: it also stops tasks from being run that do not influence the sections that the user wishes to protect. To overcome this advantage Logoview has been provided with semaphores.

8.1.5 Semaphores

A semaphore is an object that can be used to protect sections of code from simultaneous running of several different tasks. A semaphore can be in two statuses: ’signaled’ and ’non signaled'. When a semaphore is in “non signaled' status it means that nobody has taken possession so that the section of code that it protects can be run. When a semaphore is in 'signaled' status it mans that a task has taken possession of it so that the section of protected code cannot be run.

Logoview has 32 semaphores, which can be freely used inside its applications. There are basically two instructions for using the semaphores: SET SEM and RESET SEM.

SET SEM first checks semaphore status: if it is 'non signaled' it changes it to 'signaled' status and continues running the instruction. On the other hand, if the semaphore is 'signaled', SET SEM suspends the current task.

RESET SEM must be run by the same task that has run SET SEM and sets the semaphore. If there is no other suspended task on the semaphore, RESET SEM simply changes semaphore status into 'not signaled'. On the other hand, if there are suspended tasks, RESET SEM reactivates the suspended tasks. In this way, each of the tasks that has been suspended to wait for the semaphore is able to enter the protected area. Obviously, there may be several areas protected by the same semaphore. In this case, only one task at a time will be allowed into any of the areas. The previous example can be rewritten using a semaphore:

```
EVENT  10
        SET SEM 1
        DB_ATTACH
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        RESET SEM 1
        RETURN

EVENT  11
        SET SEM 1
        DB_ATTACH
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        DB_PUTFIELD
        RESET SEM1
        RETURN
```

Unlike the version that used LOCK/UNLOCK this version does not prevent other tasks from running and therefore improves the general performance of the application.

8.2 Screens

Amongst the properties of each task there is the screen on which it has been activated, if it has been activated on a screen. Not all the circumstances that activate an event (and therefore a task) assign a screen to the task.

Logoview does not assign a screen to the task when it activates an event in the following circumstances:

- The set time assigned to the event has elapsed
- The trigger assigned to the event has changed its status
- PLC data block reading
- Alarm status change
- Modification of animation variable

For none of the events activated by these causes will it be possible to use the instructions that require a screen on which to work such as MM_ … instructions, CH_… instructions, etc. The instructions guide specifies whether an instruction can be used in events to which no screen has been assigned.

In the events that have been started up in the following circumstances Logoview **always** assigns a screen to the task:

- Activation of a button by the user
- Editing an interactive field
- Editing a table cell
- Selecting a menu item

Having an assigned screen enables certain types of instructions to be used and also enables graphic instructions to be used without having to specify the screen on which one intends to operate. By default, all the graphic instructions draw only on the screen assigned to the task. However, this behavior can be modified and other screens can be drawn on if the instruction SET_SCREEN is used.

The instruction SET_SCREEN enables drawing even on screens that are different from the assigned one. The SET_SCREEN instructions can be bunched in order to enable several different screens to be drawn on. Obviously, drawing will take place only if the screens have been opened. RESET_SCREEN eliminates the effect of the last SET_SCREEN.

```
EVENT 10
        SET_SCREEN "one"
        LINE 0,0,100,100,4
        SUB 20
        LINE 100,100,100,200,4
        RESET_SCREEN
        RETURN

EVENT 20
        SET_SCREEN "two"
```

```
BOX 10,20,100,200,6
RESET_SCREEN
RETURN
```

In this example event 10 enables drawing ’one’ to be drawn on, draws a line and calls event 20. Event 20 enables the drawing on screen ’two’, which is added to the one on screen ’one’. The final RESET_SCREEN restores the previous situation so that upon returning to event 10 only screen ’one’ can be drawn on.

SET_SCREEN has two special parameters: “$ALL” instructs Logoview to draw on all the open screens and “$NONE” tells Logoview not to draw.

9. Instructions linked to the objects

Some Logoview instructions intervene specifically on special types of objects positioned on the screens. This category of instructions requests the name of the object on which it should operate as a first parameter. In addition, the event that runs the instructions must be activated by a screen, for example by a button. This means that the events activated by the timer or by the trigger cannot use this category of functions because they do not have an assigned screen. The instructions intervene on specific objects and can be grouped together according to the type of object on which they intervene.

MM_…: The instructions that begin with MM_ intervene on multimedia objects and enable event operations to be checked.
CH_…: The instructions that begin with CH_ intervene on graphics, enable graphics to be updated and modify some of the characteristics.
TAB_…: The instructions that begin with TAB_ intervene on tables.

10. Multimedia

Logoview NT provides some Windows multimedia functions. The simplest and most immediate way of using these functions is to place multimedia objects in the screens .

10.1 Multimedia objects

Using multimedia objects makes it very easy to reproduce films and sound and also provides the user with a series of checks of reproduction. The multimedia objects are based on the Windows standard known as **MCI** (Media Control Interface). They can easily be checked by events by means of **MM_** instructions. The most common MCI commands, such as PLAY, OPEN and CLOSE have been implemented directly as an instruction whilst all the other MCI commands can be sent to the objects as an MM_COMMAND instruction.

10.2 MCI_COMMAND instruction

Another method of using Windows multimedia functions is to directly access the MCI interface by means of the instruction MCI_COMMAND. This enables devices to be controlled that do not need to display an object in a window such as the reproduction of music and sounds. The MM_COMMAND and MCI_COMMAND instructions have the same function. The only difference is that the MM_COMMAND intervenes on a specific object in a screen so that a work area is already provided for all the devices that require it. The MCI_COMMAND, on the other hand, cannot provide a window for the multimedia devices. If necessary they open an autonomous window. The difference can easily be seen in the case of the reproduction of a video. Multimedia objects are reproduced inside the assigned screen. On the other hand, if the instruction MCI_COMMAND “PLAY myvideo.avi” is used reproduction takes place in an autonomous window that is imposed on the Logoview window. Apart from this difference the two instructions are identical, require the same parameters and return the same results.
In particular, the MCI commands consist of a sequence of strings and return a result in the form of a string result and a numeric error code. If the error code is zero this means that the instruction has been successful. On the other hand, a string can be obtained that describes the error by means of the instruction MCI_GETERROR.

10.3 MCI

The MCI interface provides the Windows applications with the capacity to check audio and video devices independently of the device. The applications can therefore control any multimedia device that is supported by Windows including waveform-audio devices, MIDI sequencers, CD-Audio devices and digital-video devices. For a detailed description of MCI consult the Windows SDK manuals that are published by MS Press or enclosed with the Microsoft development tools. Below, we have set out a general explanation of MCI and its use.

10.3.1 Syntax of the command strings

The command strings MCI use a simple *verb-objects-modifier.* Each command string includes a command, a device identifier and the command argument . The arguments are optional for some commands and compulsory for others.
A string command has the following form:

command device_id arguments

These components contain the following information:

- *command* specifies an MCI command such as **open**, **close** or **play**
- *device_id* identifies an instance of a driver MCI. The *device_id* is created when the device is opened.
- *arguments* specifies the flags and the variables used by the command. The flags are key words that are recognized by that command. The variables are numbers or strings that address the command or the flag.

  For example, the command **PLAY** uses the arguments "from *position*" and "to *position*" to indicate the positions in which to start and stop **PLAY**. The flags can be listed in any order. When a flag is used that requires a variable the value of the variable must be provided. Non-specified (and optional) arguments take on default values.

The following value sends a "from" and "to" **PLAY** command with flags.

```
STR_PRINT $L0,"play %s from %u to %u", $P0,:P0,:P1
MCI_COMMAND $L0,$L1,:L0
```

10.3.2 Data types with command variables

The following types of data can be used for the variables in an MCI command:

Data Type	Description
Strings	Strings are marked off by an initial space, a final space and double inverted commas. The MCI removes the single inverted commas from the string. To insert a quote into a string insert a pair of inverted commas. To use an empty string insert two double inverted commas marked off by two spaces.
Long whole numbers with sign	Whole numbers with a sign are marked off by spaces. If no explicit indications are given, whole numbers can be positive or negative. If negative numbers are used the minus sign must not be separated from the first digit of the number.
Rectangles	The rectangular type is a list sorted into four short whole numbers with sign. The spaces mark off this type and separate each whole number in the list.

10.3.3 Classification of MCI commands

MCI defines four types of commands: *system*, *required*, *basic*, *extended*. The classification of commands is described below:

- The **system** commands are managed directly by the MCI rather than the driver.
- The **required** commands are managed by the driver. All MCI drivers must support **required** flags and commands.
- The **basic** commands (or optional commands) are used by devices. If a device supports a basic command, it must support a set of flags that has been designed for that command.
- The **extended** commands are specific to a type of device or driver. Extended commands include commands such as **put** and **where** for **digitalvideo** and **overlay**

devices or existing command extensions (such as the “stretch” flag for the **status** command on the **overlay** devices).

Whilst *system* and *required* commands are the minimum set of commands for an MCI driver the *basic* and *extended* commands are not supported by all drivers. A Windows application can always use the commands *system* and *required* but if it needs to use the commands *basic* or *extended* it must first ask the driver if the command is supported by means of the **capability** command.

System Commands

MCI processes the following system commands directly.

Command	Description
break	Defines the break key that interrupts MCI commands
sysinfo	Returns information on MCI devices

Required Commands

All MCI devices support the required commands. In addition, they must support a standard set of flags for each command.

Command	Description
capability	Returns device capabilities, i.e. the supported commands.
close	Closes the device
open	Initializes the device
status	Returns information on device status. Some of the flags of this command are not required so this is also part of the basic command.

Basic Commands

Support of these commands by MCI drivers is optional.

Command	Description
load	Loads data from a file
pause	Suspends reproduction. Reproduction or recording can be resumed from the current position.
play	Starts to transmit output data
record	Starts to record input data
save	Saves data onto disk
seek	Moves current position backwards and forwards
set	Sets device's operating status
status	Returns information on device status. This is also a compulsory command. As some flags are optional it is also listed here.
stop	Stops reproduction

Extended commands

Some MCI devices have additional commands or add flags to existing commands. Whilst some *extended* commands are applied only to a specific device, most are applied to all the drivers of a particular type of device. For example, the set of commands for the **sequencer** device extends the **set** command to add the time formats required by the MIDI sequencers. It should not be automatically assumed that a device supports *extended* commands. The **capability** command must always be used to establish whether a given characteristic is supported. The following extended commands are available for the devices listed alongside.

Command	Device	Description
configure	digitalvideo	Displays a configuration window
cue	digitalvideo, waveaudio	Prepares for reproduction or recording
delete	waveaudio	Deletes a segment of data from the file media
escape	videodisc	Sends custom information to the device.
freeze	overlay	Disables video acquisition in the screen buffer
put	digitalvideo, overlay	Defines source and destination windows and screens.
realize	digitalvideo	Tells the device to make up its own palette in the context device of the displayed window
setaudio	digitalvideo	Sets audio parameters for the video
setvideo	digitalvideo	Sets video parameters
signal	digitalvideo	Identifies specific position with a signal.
spin	videodisc	Starts or stops disk rotation.
step	digitalvideo, videodisc	Starts step-by-step reproduction backwards or forwards.
unfreeze	overlay	Enables screen buffer for acquisition of video data
update	digitalvideo	Updates the screen current in the device context
where	digitalvideo, overlay	Returns the rectangle of the screen, destination or source windows
window	digitalvideo, overlay	Checks display window

10.3.4 Wait and Test flags

Most MCI commands include flags that modify command behavior. “wait” is common to all devices, 'test' is available only with the devices **digitavideo** and **vcr**.

wait

MCI commands normally return immediately to the caller, although it takes several minutes for the action started by the command to be completed. The flag 'wait' can be used to ensure that the device waits for the command to be completed before returning the control to the caller. For example, the following **play** command will not return control to the caller until reproduction has been completed. **Warning: using the “wait” flag will suspend the entire LPE interpreter and will therefore also suspend other active tasks.** The other threads current in Logoview will not be stopped, e.g. alarms or communications management.

```
MCI_COMMAND "play mydevice from 0 to 100 wait",$s0,:err
```

Note: the user can interrupt a suspended operation by pressing 'break'. The default keys are CTRL+BREAK. The applications can redefine this key by using the **break** command. When a 'wait' operation is cancelled, the MCI will attempt to return control to the caller without interrupting the command assigned to the “wait” flag.

test

This flag interrogates the device by asking if it can run the command. The device returns an error if it cannot run the command. When this flag is specified the MCI returns to the caller without running the command. This flag is supported by the devices **digitalvideo** and **vcr** for all the commands except for **open** and **close**.

10.3.5 MCI device controls

To check an MCI device it must be opened, the control commands must be given and then closed. The commands may be very similar even for completely different devices. For example, the following sequence of commands reproduces the sixth track of an audio CD:

```
MCI_COMMAND "open cdaudio",$l0,:l0
MCI_COMMAND "set cdaudio time format tmsf",$l0,:l0
MCI_COMMAND "play cdaudio from 6 to 7",$l0,:l0
MCI_COMMAND "close cdaudio",$l0,:l0
```

The following example shows how to reproduce the first 10000 samples of a wav file:

```
MCI_COMMAND "open c:\mmdata\purplefi.wav type waveaudio alias finch",$l0,:l0
MCI_COMMAND "set finch time format samples",$l0,:l0
MCI_COMMAND "play finch from 1 to 10000",$l0,:l0
MCI_COMMAND "close finch",$l0,:l0
```

These examples illustrate some interesting aspects of MCI commands:

- the same base commands (open, set, play, close) are used for CD audio and for wav files. The same commands are used by all MCI devices.
- The open command for waveform-audio (file wav) includes the name of a file. The waveform-audio is a *composite* device (i.e. assigned with a data file) whilst the CD-audio is a *simple* device, i.e. without an assigned data file.
- The variables used with the flags “from” and “to” are appropriate for the respective time formats. For example, CD-audio variables specify a range of tracks whilst waveform-audio variables specify a range of samples.

10.3.6 MCI devices

MCI recognizes a basic set of two types of device. One type of devices is a set of drivers that share a set of common commands and which are used to check similar devices. Many MCI commands, such as **open**, ask for a type of device to be specified.
The following table lists the most common types of device:

Device Type	Description
cdaudio	CD audio player
dat	Digital-audio tape player
digitalvideo	Digital video in a window (not GDI-based)
overlay	Overlay device (analog video in a window)
scanner	Image scanner
sequencer	MIDI sequencer
vcr	Video-cassette recorder or player
videodisc	Videodisc player
waveaudio	Audio device that plays digitized waveform files

The device **cdaudio** supports the following commands:

break	capability	close
info	open	pause
play	resume	seek
set	status	stop
sysinfo		

The device **digitalvideo** supports the following commands:

break	capability	capture
close	configure	copy
cue	cut	delete
freeze	info	list
load	monitor	open
paste	pause	play
put	quality	realize
record	reserve	restore
resume	save	seek
set	setaudio	setvideo
signal	status	step
stop	sysinfo	undo
unfreeze	update	where
window		

The device **overlay** supports the following commands:

break	capability	close
freeze	info	load
open	put	save
set	status	sysinfo
unfreeze	where	window

The device **sequencer** supports the following commands:

break	capability	close
info	open	pause
play	record	resume
save	seek	set
status	stop	sysinfo

The device **vcr** supports the following commands:

break	capability	cue
freeze	index	info
list	mark	pause
play	record	resume
seek	set	setaudio
weekcode	settuner	setvideo
status	step	stop
sysinfo	unfreeze	

The device **videodisc** supports the following commands:

break	capability	close
escape	info	open
pause	play	resume
seek	set	spin
status	step	stop
sysinfo		

The device **waveaudio** supports the following commands:

break	capability	close
cue	delete	info

open pause play
record resume save
seek set status
stop sysinfo

10.3.7 Using MCI commands

In this section we shall take a brief look at how to open and use MCL devices.

Open

Before using a device use the open command to initialize it. This command stores the drivers in the memory and enables an identifier to be obtained that is used in subsequent MCI commands. The number of devices that can be opened depends only on the amount of memory available.

Using an alias

When a device is opened, the ’alias’ flag can be used to specify an identifier for the specific device. This flag enables a brief identifier to be assigned for the devices with long file names. It also enables several instances of the same file or device to be opened.
For example, the following command assigns the identifier “birdcall” to the name “C:\NABIRDS\SOUNDS\MOCKMTNG.WAV”:

```
MCI_COMMAND "open c:\nabirds\sounds\mockmtng.wav type waveaudio alias
birdcall",$l0,:l0
```

Specifying a device type

When a device is opened, the “type” flag can be used to refer to a type of device rather than to a specific device driver. The following example opens the file waveform-audio C:\WINDOWS\CHIMES.WAV (using the flag “type” to specify **waveaudio** device) and assigns it the alias “chimes”:

```
MCI_COMMAND "open c:\windows\chimes.wav type waveaudio alias chimes",$l0,:l0
```

Simple and composite devices

MCI classifies devices as *simple* or *composite*. The drivers for the composite devices require the name of a file.

Simple devices include **cdaudio** and **videdisc**. Simple devices can be opened in two ways. To specify a string with the name of the device, in order to open the device **videodisc,** for example, the following command can be used:

```
MCI_COMMAND “open videodisc”,$l0,:l0
```

- By specifying the actual driver name. In this case, the application will be closely linked to the driver and any variation in the system will stop the program from working.

Composite devices include **waveaudio** and **sequencer**. A composite device can be opened in three different ways:

- by specifying only the name of the device. In this way a device can be opened without specifying a file name. Most of the composite devices process only the **capability** and **close** command when they are opened in this way.
- By specifying only the name of the file. The MCI will assign to the device by examining the registry.
- By specifying the name of the file and the device.

For example, the following command opens the device **waveaudio** with the file myvoice.snd:

```
MCI_COMMAND "open myvoice.snd type waveaudio",$s0,:l0
```

Creating new files

To create a new data file it is sufficient to leave an empty space for the name of the file. MCI does not save the file until the command **save** is given. To create a file, use an alias in the **open** command. The following command opens a new **waveaudio** file, starts and stops recording and finally saves and closes the file.

```
MCI_COMMAND "open new type waveaudio alias capture",$l0,:l0
MCI_COMMAND "record capture",$l0,:l0
MCI_COMMAND "stop capture",$l0,:l0
MCI_COMMAND "save capture orca.wav",$l0,:l0
MCI_COMMAND "close capture",$l0,:l0
```

Play

After opening the MCI device the **play** command can be used to start reproduction. Without flags this command starts reproduction from the start to the end of the media.

Stop

The **stop** command enables reproduction to be momentarily stopped. To resume, use the command **play**.

Shutting down device

The **close** command releases resources allocated by the **open** command and informs MCI that the application has finished using the device. Always use the command **close** after using an MCI device.

This example shows how to open, reproduce and close a waveaudio file:

```
MCI_COMMAND “open c:\winnt\media\flute.wav alias msc”
MCI_COMMAND “play msc”
MCI_COMMAND “ close msc ”
```

A video can be reproduced in the same way:

```
MCI_COMMAND “open c:\winnt\media\test.avi alias msc”
MCI_COMMAND “play msc”
MCI_COMMAND “ close msc ”
```

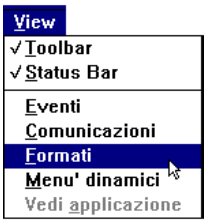
A shorter form can also be used as an alternative:

```
MCI_COMMAND “play c:\winnt\media\flute.wav”
MCI_COMMAND “play c:\winnt\media\test.avi”
```

11. Output on printer

The Windows printout is organized into pages. Windows requires information to be first organized into a page and then be printed. This is why Logoview enables print formats to be organized into pages.
The ***LOGOVIEW NT*** section that is dedicated to the print formats enables the different page structures to be created that are required for the application. These structures are then filled by the event. ***LOGOVIEW NT*** sets no limits to the number of formats that can be configured. It is therefore possible to have a different format for each size of group for which a printout is required. As we have said, printing occurs in two phases: in the first phase the skeleton of the page to be printed is set and in the second phase the actual print command is given. The second phase is carried out by an event by means of the available print instructions.

11.1 Creating formats



To define a print format select the item ***Formats*** from the ***Display*** menu or press on the main instruments toolbar. For a detailed explanation of how to create print formats see the chapter 'Print Formats' in the manual 'Guide to using Logoview NT'.
The print formats are organized into modules. In other words, the page can be divided into sections of equal size. This enables, for example, reports to be generated that are arranged by line, column or a combination of the two. Whilst defining the report you need prepare only one of the forms and Logoview will repeat it on the page during printing.
During the definition phase the type of format output must also be set. In other words, do we require an automatic printout every x pages (where x is a number greater than zero) or a printout controlled by the instruction FLUSH_FORM? If an automatic printout is required, the default printer to be used for the printout must also be set.

11.2 Generating printouts

Printouts must be generated during runtime by instructions inserted into the events. Each time that a form in one of the defined print formats is to be printed, recall the instruction PRINT_FORM. Logoview stores the different forms that are to be printed in temporary files. If a form takes up an entire page each time that the instruction PRINT_FORM is activated it generates a new print page. Otherwise, Logoview generates the different forms one at a time until the entire page is filled. The modules on the page are generated starting in the top left hand corner. A line is then completed moving from left to right and then the next line is generated and so on until the page is finished.
The actual printout can be carried out in one of the following ways:

- With the instruction FLUSH_FORM. Use the instruction FLUSH_FORM to force the event printout. This enables event printouts to be generated completely automatically without any intervention by the user.
- Automatically when the maximum number of pages in the buffer has been reached. When the set number of pages in the buffer has been reached Logoview automatically starts a default printout. This system is useful for long printouts for which it is inconvenient to keep track of the pages by means of the event.

- Manually from the menu. During runtime the user can access the list of print formats of the pages waiting to be printed and can print them.

Whatever method is used, only complete pages can be printed. The only exception to the rule is the printout of the alarms log. However, this is not a normal printout because it is made up of a simple flow of ASCII characters that are sent directly to the parallel or serial port; because of the fullness of print reports this is not possible so complete pages have to be printed.

If, for example, we have a print format that consists of 10 modules the following event will generate a page and print it on the default printer.

```
EVENT 10:PRINT
VARIABLES NAME=LOC WORD=2 REAL=0

LET :0=0
WHILE (:0<10)
    PRINT_FORM 0
    LET :0=:0+1
END_WHILE
FLUSH_FORM 0,0,:0
RETURN
```

12. Graphic instructions

The LPE language comprises many different graphic instructions that enable the most common graphic elements such as lines, circles and rectangles to be drawn. The graphic instructions work only on the current screen so the event that runs the instructions must have been activated by a screen, for example, by a button. This means that events that are activated by a timer or a trigger cannot normally use this type of function because they do not have any screen assigned to them. This is only partially true because the instruction SET_SCREEN enables the user to draw on any screen, providing it is open. Chapter 8.2 showed how Logoview assigns the screens to the tasks. We will now see how the graphic instructions affect screens . For a detailed explanation of screens and colors see the chapter 'Screens' in the 'Guide to the use of Logoview NT'.

12.1 Coordinates

During the development phase reference boxes can be placed on the screens. These boxes are used both for inserting the graphic items directly during the development phase and for facilitating the drawing of graphic features during runtime.
The reference coordinates used by the graphic instructions can be absolute, i.e. refer to the entire screen or else they can refer to the current box.
In particular, when an event is activated in the following occasions the box that has activated the event is set as a current box.

- Activation of a button by the user.
- Editing of an interactive field
- Editing of the cell of a table
- Modification of the animation variable

For this reason, any drawing instructions will be run by using coordinates referring to that box.
The current box can be modified by the instruction SET_BOX to enable drawing with coordinates referring to boxes that are different from the default boxes and in order to be able to draw from events without an assigned screen.
To deactivate the current box use the instruction RESET_BOX; all the instructions after RESET_BOX will have absolute coordinates.
To summarize: if there is a current box the graphic instructions work with coordinates referring to that box; if there is no current box the coordinates will be absolute.
In all cases the origin of the axes coincides with the vertex in the top left-hand side of the screen .

In the following example the rectangle is drawn at a distance of 10 pixels from the vertex in the top left-hand corner of box 2 whilst the line is diagonal and goes right across the screen.

```
SET_BOX 2
BOX 10,10,20,20,0
RESET_BOX
LINE 0,0,639,479,0
```

12.2 Colors

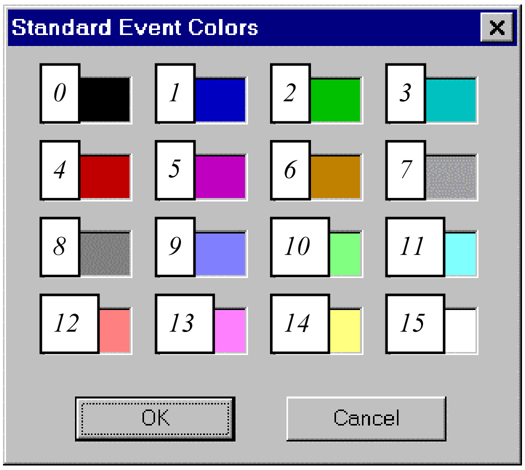
Many parameters also accept a color as a parameter. This will be the color with which the graphic instruction is carried out on the screen. For example, the instruction BOX draws a rectangle of a certain color on the screen that is currently loaded. This instruction accepts five parameters. The first four represent the coordinates of the rectangle and the fifth represents the color of the rectangle.

BOX x1,y1,x2,y2,**color**

The **color** parameter can be expressed in three different methods. The color can therefore be expressed in three different ways in Logoview. These methods are explained in the following paragraphs.

12.2.1 Color number

The first method for specifying a color in a graphic instruction is to indicate the number inside the table shown in the window “Standard event colors”.



The corresponding numbers are shown in the figure. Color numbers from 0 to 15 are available and they correspond to the colors shown in this window. These colors meet the condition of best match. In other words, the color is chosen that is closed to those in the palette of 256 colors in the screen. The colors may therefore vary from one screen to another, depending on the palette used by the screen. It is also possible to use 16 additional colors (numbers 16 to 31) that correspond to the standard Windows colors.

In the color parameter of a graphic instruction the first method of indicating a color is therefore to specify a number from 0 to 31.

12.2.2 Macro RGB

It is also possible to specify the color by means of the macro **RGB.** This macro must be positioned in the color parameter of a graphic instruction.

For example, the instruction BOX is written thus:

BOX x1,y1,x2,y2,**RGB(g_Red, g_Green, g_Blue)**

The macro accepts three parameters that are red, green and blue shades of the color that is to be used. The macro may not supply the correct color as identified by the three parameters: it always provides the best match. In other words, the color that is most similar to those in the palette of 256 colors of the screen is chosen.

12.2.3 RGBA Macro

The **RGBA** macro can also be used. The format and the use of the format are identical to those specified for the RGB macro. The only difference is that the color is simulated by the technique of dithering. The color may be exactly the color indicated by the parameters of the macro but it will not be a pure color. The colors obtained by dithering are composed of pixel screens of different colors. Unfortunately, this affects some graphic operations and adversely affects results. Great caution must therefore be exercised when using colors obtained by this macro.

12.3 Logic operations

In Logoview there are two ways of writing colors into the bitmaps. The first system, the one that is usually used, consists of replacing the color in the foreground with the drawing color. The second method is to combine the design color with the color already current in the screen by means of an XOR operation. The GR.SET instruction activates the first of the two methods. After this instruction has been run all the graphic operations draw their own graphic elements with the color given as a parameter.

On the other hand, the instruction GR.XOR activates XOR operations in such a way that each graphic operation combines the drawing color with the color already current in the background. Note that if the same graphic instruction is carried out with XOR the original color is restored. For example, if color 7 is on the screen and a rectangle is drawn in XOR with color 31 the new color will be: 7 XOR 31= 24. If the same rectangle is drawn again color 31 will change with the new background color, which is 24: 24 XOR 31= 7

12.4 Instructions

The graphic instructions include drawing instructions and other instructions that are not strictly speaking graphic instructions but which are connected with the graphics or with the screens.

The following instructions are available for drawing graphic items.

Instruction	Description
BOX	Draws a rectangle
CHANGE COL	Changes a color in a rectangular area
DISPLAY	Writes a text
DISPLAYBOX	Writes a text in a rectangle
ELLIPSE	Plots an ellipse
LINE	Draws a line
PAINT	Fills a closed area
SOLID BOX	Draws a full rectangle
SOLID ELLIPSE	Draws a full ellipse

The following signals can be used to manage box and screen.

Instructions	Description
CLOSE SCREEN	Closes the screen that is assigned to the task that calls the instruction.
RESET BOX	Deactivates the current box

RESET SCREEN	Cancels the effect of the last SET SCREEN
SCREEN	Opens a new screen
SET BOX	Changes the current box
SET SCREEN	Enables writing on screens other than the default screen.
LOAD_AREA	Loads a bitmap type image inside a triangle inside the current screen
OPENURL	Opens an HTML file in a Web server or local disk.

The graphic instructions include a pair of special instructions: the SCREEN enables the current screen to be opened and OPENURL enables an html file to be displayed.

SCREEN

This instruction enables a new frame to be opened. The new screen can substitute the current screen or generate a new window that is superimposed on the current one. The SCREEN instruction can also be used to open the line history or instant trend lines. In this case it is sufficient to specify the trend title instead of the frame name.

OPENURL

This instruction enables pages in the standard Web html format to be displayed. The pages can either be on a Web server in the local net or on the Internet. Obviously, only one page can be opened at a time but once a page is opened the user will also be able to follow other links and to therefore also navigate on the Web.

13. Database masks

In order to simplify displays of the data contained in the database Logoview enables masks to be used that have been created by the Titano program. These masks are windows that are identical to those of the frames, inside which there are fields that display the contents of the database fields.
The database masks must be created and modified by Titano. Once they are inserted inside Logoview they can be opened like normal screens or by means of the instruction *DB_OPENVIEW*. This enables greater control of window opening.

CONTENTS

Introduction

LOGOVIEW NT is a software package providing an integrated environment to allow the development of plant supervision applications while taking full advantage of the Win32 Operating Systems interface capabilities.

This Manual

This manual contains introductory information intended to make the Developer familiar with the ***LOGOVIEW NT*** interface and to install it. In addition it provides the reader with just a broad description of all available functions, thus constituting a complementary part of the complete documentation set.

This introductory volume is made up of three parts:

- ***PART I*** *Introduction*

  Describes the basic notions required to begin using LOGOVIEW NT and provides a complete guide to the installation and to the User interface

- ***PART II*** *The development environment*

  Describes the main functions of the Logoview development environment, allows to easily create an application by taking advantage of the Wizard and provides some useful notions for the application development.

- ***PART III*** *The runtime environment*

  Describes the main functions of the Logoview runtime environment and provides a general view of how to interact with a running application.

Conventions

Throughout this book the following symbols are used to highlight in an immediate mode some ***LOGOVIEW NT*** aspects.

❓ ***This icon anticipates some answers to questions which the LOGOVIEW NT user might possibly raise on the subject under description.***

❗ ***This icon highlights possible errors, along with their solutions, which might be experienced while using LOGOVIEW NT.***

💡 ***This icon highlights suggestions and examples on the proper application of given LOGOVIEW NT functions.***

👁 ***This icon highlights the Notes***.

Technical Support

In order to ensure that our Customers will make the best possible use of the purchased products, *Logosystem S.p.A.* provides a *Customers Technical Support Service.*

Should problems arise while using a *Logosystem* product, it is suggested that you first check with the printed documentation, the one on magnetic support and the online Help. This should suffice in most cases. The *Customers Technical Support Service* should be considered and contacted in those cases when no solution could be found otherwise.

The *Customers Technical Support Service* is available through any authorized distributor. The distributor addresses are listed in the card included with the ***LOGOVIEW NT*** kit. The ***LOGOVIEW NT*** version number and your hardware configuration could be required when contacting Service.

PART I

Introduction

1. The Supervision : Introduction

Since the first systems based on electro-mechanical technology till to the present day ones, based on Personal Computers and Workstations with advanced software, the Process Supervision Systems show ever growing capabilities and performance. But how to evaluate their real effectiveness?

The Supervision System consists into the man/machine interface. By means of it the Operator can keep under his check the Plant from a central work station, operate the various controls and evaluate the obtained results.

Any Supervision System usually includes:

- central unit for processing and distributing the plant data (minicomputer, microcomputer or PC).
- mass storage for data archiving.
- graphic screen terminals for displaying the process variables.
- keyboard.
- operator benches and panels.
- plant synoptic (optional).
- connection to some input/output interface (direct or through LAN).

We took this definition from the Analytic Guide to the Supervision Systems of Factory Automation - THE AUTOMATED FACTORY SHOW.

1.1 Everything manual once

Not many years ago, “supervising” a plant carried a quite definite meaning. The Production Manager would leave his own office and personally check that everything proceeded to the best. Something more advanced, but not much, happened in the Electrical Power Plants, where large panels were depicting the whole system, including measuring instruments and lamp signals on top of lines and graphic symbols of various colors.

Only at the beginning of the eighties the first text displays appeared on board of machine tools together with the first software applications, developed on a case by case basis and only for the large production plants.

The message displays, in particular, have provided the first step toward a practical and effective interface between the floor and the user.

The operator interaction was and is nowadays made easier by the display of alarms and events and by the built in integration of online instructions, manuals and repair action tips (taking advantage of the memory capabilities of modern devices). The natural evolution from the simple text display is the operator panel. This is a graphic interface including function keys and alphanumeric keyboards, allowing bi-directional communication between user and machine.

During the nineties, you could witness the spreading of affordable supervision systems based on computers and software packages equally effective on large complex plants and small production units, running indifferently on large industrial and small consumer computer hardware. Hence, ten years after being introduced, the first text displays, although constituting the de facto standard solution for most machine tools, begin to appear obsolete in some particular applications, namely those requiring a more and more advanced integration between PC and PLC and effective supervision even in cases of medium scale complexity.

The evolution of this trend is tightly linked to the Programmable Controller‘s success as a control device in the automatic factory. The capabilities offered by the PLC have been steadily growing and the supervision applications have constantly paralleled this trend, by offering solutions more and more effective and complex, at the same time maintaining reasonable costs when compared to performance.

Likewise, the Operator Supervision systems have been keeping this pace and, consequently, they are today applied in almost all cases. It is essential for their progressive expansion, both hardware and software, the maintaining of an adequate price/performance ratio and a high degree of overall reliability.

1.2 System composition

An average supervision system can be summarily described as being composed of a computer equipped with serial, parallel and video ports connecting, as appropriate, one or more PLCs, a printer, a mouse or a joystick and a video monitor. On top of that a software package will provide the system functionality and the programming tools.

The application programming, fully guided, includes two phases. The first one ensuing the graphic pages implementation, the second one allowing the communication parameters definition.

The graphic implementation is greatly simplified by pre-compiling a library of symbols and colors which can be in turn easily updated even at pixel and color palette level. The mouse and the functional keys allow easy management of the screen layout, so that fairly complex pages can be completed in quite reasonable time.

Once the graphic part has been fully defined, the communication blocks can be established, assigning individual computer bits to map the various PLC inputs and outputs, and computer variables to store data registers, this one constituting a process we call system configuration.

1.3 Effectiveness and Performance

The effectiveness of a system is always related to its at hand performance both hardware and software. The market offers a wide selection of systems and almost all Providers are in the position to present a number of options. The reason is that the supervision systems providers often are also manufacturers of other automation products (such as PLCs) and therefore they know very well the problems they are bound to meet.

In general, the main available functions include:

- the graphic reproduction of the whole controlled plant with real time display of animated synoptic screen layouts.
- the display of voltages, temperatures, pressures and other variables in various forms including numeric tables, bar graphs, line trends, both in dedicated pages and as part of screen layouts.
- the management of recipes, that is sets of process parameters which, once loaded into the PLC, establish its proper controlling cycle.
- the recording of events, alarms and parameters set by the Operator on magnetic and/or paper support for long term archiving and further processing.

- the fast interaction between the Operator and the process. He or she should not be concerned with how the control program has been implemented, but rather simply answering to the various options set forth by the appearing screen menus, pressing functional keys, entering values and providing consent to operations such as starting, stopping and resetting devices.
- the off-line process simulation, with consequent confirmation of the intended process and control changes before they are actually implemented and evaluation of the possibly undesired consequences of faults and malfunctions.
- the drawing of statistical charts on the production progress.

1.4 Evaluation Criteria

Which are the criteria allowing to effectively evaluate the price/performance ratio of a supervision system?

Besides the traditional purchasing criteria we all usually apply when approaching a new product (the Supplier’s name, the after selling service, the included manuals...) others can be ascertained more precisely identifying the system quality.

These are, briefly, the most important ones:

- the number of digital and analog points the system can handle and, more generally, the amount of variables for holding the received floor data.
- the historical trends which can be displayed and stored.
- the updating speed of data on screen.
- the graphic symbol sets and the available colors.
- the accessibility of the Operator’s controls.
- the available communication protocols.
- the data acquisition capability.

More generally, a good supervision system must grant a simple Operator’s interface, allowing a comfortable monitoring of the plant production trend.

The Operator himself must be put in the condition to carry on real time interventions thus correcting process deviations from the norm and entering new parameters any time it is convenient to do so. The saving in terms of space and cost with respect to the traditional synoptic panels is easily appreciable. The picture of a valve, for example, is not better defined or any way different from the one on the panel, but now it is an easily repeatable symbol which is no longer part of a non reusable panel. Needles to mention the reduction of wiring, the quick updating of the layout by simple software changes and the effectiveness provided by the graphic animation.

1.5 The Supervision in the Plant

In the present day a new trend change can be witnessed. From the central control philosophy to the distributed control one, where the supervising subsystems become the key instruments for the production optimization. The capability of generating periodic reports is being added to the classical and effective visual supervision functions, leaning towards a total command of the plant. The supervision systems have decisively established themselves in the last years in all those automation implementations requiring to maintain the operational parameters within ever precise limits and ever more complex interactions among the intelligent devices. Large and small computers have widened their presence in the plants with immediate benefits on the production procedures and parallel achievement of functionality and profit goals. The various companies which applied them are this way trying to increase the productivity of their own industrial processes, not forgetting safety, consistency and quality of the product and plant availability. The attempt is to reduce the energy consumption, the waste of material, the plant down time and the amount of off quality product.

1.6 Distribution and Control

A few of these objectives are inherently reached by the basic automation alone, by the spread of the digital control. This way the requirement can be reached of keeping the plant running as steadily as possible with the purpose of amortizing the investment costs in the minimum time, at the same time heading towards the equally important issues of safety both for human operators and mechanical devices. Other objectives are reached instead by means of advanced control systems allowing the dynamic stability of the process. It is self evident that the more stable is the running of a continuous plant, the more consistent will be the product quality. The automatic plants, made dynamically stable, can be subsequently more easily amortized by the use of supervising systems which, in fact, are dedicated to the maximization of production and minimization of costs. Such systems are made of a color video screen connected to a computer allowing the plant graphic representation, display of the greatest number of alarms, monitoring of calculated variables such as mass and energy balance, operational units efficiency, printing of tables and so on. The availability of more and more advanced hardware and software allows us to witness in these last years a significant evolution in the supervision concept. In the industrial frame, we define as supervisors the computers which are able to provide:

- fully comprehensive and constantly updated data bases of the process under control.
- man/machine interface making use of video graphic technology which includes extensive interactivity.
- modeling and calculation algorithms, including evaluation and predicting functions, to be configured on a case by case basis.
- in addition, very recently it is becoming more and more significant the video technology applied to remote surveillance making use of multimedia software.

1.7 Monitoring Structure

A supervision station should allow the operator to reach in due time the whole information pieces required to monitor the plant running and to perform interventions, when applicable, consisting into parameter changes, preventative and repair maintenance operations and so on. The application of a color graphic supervision system allows an immediate representation, extremely useful for the operator, who should check the plant functioning on the screen in the most effortless way. The representation can be seen as made of fixed parts (background pictures, positions, comments and so on) and of variable parts (alphanumeric data, bar graphs, line trends both short and long term).

The first ones are displayed at each operator's request (operator loading). The latter ones are instead brought on screen and constantly updated on the basis of real time data acquired from the floor. In particular, the alphanumeric data and the bar graphs are updated at any change of the corresponding variable, while the line trends at the fixed programmed rate. A symbol can be associated to any variable part, which makes it possible for the operator to force the corresponding variable value, that is to arbitrarily change it, during the plant running. The forcing procedure is usually quite simple. You do so by positioning the cursor over the associated symbol, using a mouse or a joy stick, and then entering the new value. In the same way you can call other screen layouts from within any one of them. Just position the cursor over the *new page selection* symbol and then press the confirmation key. Often forcing variables turns out to be extremely useful during the plant tuning phase. You could, for instance, initialize a loop controlling variable, watch its effect on the outputs and possibly repeat the procedure until the outcome is satisfactory.

1.8 The Screen Layouts Tree

The capability of forcing screen layouts calls allows the Designer to define a Screen Layouts Tree and to jump inside it according to pre-assigned choices, starting from one of them having the role of an index. Starting from a very first page depicting the general plant synoptic, the Operator can request screen changes reaching other screens depicting other parts of the plant in more detail, then repeat the procedure to obtain even more details or move back to more general screen layouts. To highlight various disparate states of objects the Supervision systems allow object symbols to appear in various colors (for instance *valve open* in green and *valve closed* in red), or complete sentences, or different shapes to become visible on screen according to the actual plant state as detected. For instance, should the level of a tank reach an undesired level, the tank picture might show it as full of liquid or empty, as applicable, and the actual tank level could appear aside in numeric form, maybe blinking should this be appropriate, together with an alarm message.

Information of this kind is certainly much more effective then the traditional numeric indication, allowing the Operator to immediately become aware of any peculiar plant state and quickly prompt the appropriate action.

1.9 Periodic Reports

Modern industrial management can not leave aside the capability of generating periodic reports on the plant status, amount of production, maintenance requirements and so on. The answer to these requirements is provided by the Supervision Systems by means of the archiving of historical graphs trends both of the long and short time type. These can be easily printed along with other meaningful information such as the records of events and alarms (which can be archived on magnetic support and called back on request). The plant emergency situations, or even only the plant status changes, besides being graphically highlighted, stored on disk and printed on paper, can be further emphasized by means of optical and acoustical warnings, no matter which screen layout is being displayed. The alarm, or event message ought to be acknowledged and consequently removed from the screen, although not from memory. Hence it will reappear periodically (or at some pre-programmed times) until removal of the generating cause. Needless to say, this programming is fully guided and does not require any particular notion on information theory. The practical use of these systems can normally be taught in a very short time, such as a few days. Once teaching is done even the most inexperienced programmer will be at ease dealing with symbols, color palettes and menus of various kind. A substantial contribution to the simplification of these matters has been granted by the raise of user friendly operation environments, the most known of them being WINDOWS.

1.10 What is Logoview

The package ***LOGOVIEW NT*** has been developed just with the purpose of providing an answer to the real needs highlighted above. This consists into an integrated environment which, thanks to its open architecture and to the applied advanced technology allows to meet the requirements of any control and monitoring application.

Besides the traditional fields of machine tools and robots, the control systems implemented under the ***LOGOVIEW NT*** environment can be applied to particular cases possibly requiring flexible controls being able to satisfy to the peculiar requirements of various working processes: shaving, wood working, metal working and so on.

Using ***LOGOVIEW NT*** one is finally given the option of assemble programs, possibly quite complex, without needing specific knowledge on basic programming.

In addition, the graphic integrated environment, applied by ***LOGOVIEW*** since its appearance in the very first version, allowed the introduction of an innovative way of conceiving the programming activity. With ***LOGOVIEW NT*** we really have at hand an object oriented programming system, where the objects are no longer consisting of abstract concepts such as functions, but rather made of ANIMATIONS, INTERACTIVE FIELDS, BUTTONS and so on.

1.11 What does Logoview offer to the Designer?

The hardware cost reduction caused a shift of the general attention towards the software production issues which now appear more critical in the overall cost control of the development and maintenance of the final product.

LOGOVIEW NT provides software tools capable of assisting the designer along the whole life cycle of the application, from the conceptual sketch to routine maintenance.

Additional functions will doubtless be added to the ones already offered by the present system, summing up your own specific know-how to the technology results already made available by ***LOGOVIEW NT***, in order to obtain a product matching personal needs and meeting the specific requirements of Users that will apply the final product.

2. Logoview : An industry product dynasty

In view of the discussed trend aiming to investment protection, ***LOGOVIEW NT*** provides another step towards the integration of the plant information tools.

In spite of the clear differences shown by hardware platforms and, above all by the various operating systems, ***LOGOVIEW NT*** seeks for the best compromise between the need of applying the newest technology to factory automation and the request of safeguarding the initial investment, seen as personnel training.

In simple words this means that the ***LOGOVIEW NT*** User gets advantage from the product continuity, which while keeping unaltered the working procedures, immediately makes use of the additional features made available by the new operating systems. Under this viewpoint, the programmer acquainted to using Logoview under MS-DOS, is in the condition of writing an application under the WINDOWS environment without the need of additional training.

Even the application programs are partially reusable. Compatibility with previous versions has been maintained, to the extent allowed by the operating system itself, allowing the User to employ procedures written with the MS-DOS versions through the ASCII conversion functions of the events editor.

Also the graphic format SCR used by the DOS versions of the package is reusable by the WINDOWS versions of ***LOGOVIEW NT***, thus extending its compatibility also to the screen layout library the User might have at his disposal.

Finally, the integration is completed by the possibility of establishing communication among the various Logoview stations, independently from the operating system, allowing substantial savings when assembling large systems.

2.1 Its History...

With the addition of the WINDOWS NT and WINDOWS 95 versions, the basic Logoview versions are now five. The ***LOGOVIEW NT*** programming language has further evolved enclosing all commands which allow the management of all WINDOWS functions. The summary table of the whole *Logoview* family follows:

Name	Operating System	Ext.	Minimum Hardware	RAM min/max
LOGOVIEW Lite	MS-DOS	16bit	i286-EGA-HD20Mb	1Mb
LOGOVIEW Plus	MS-DOS	16bit	i286-VGA-HD20Mb	1/16Mb
LOGOVIEW 32	MS-DOS	32bit	i386-VGA-HD40Mb	4Mb/1Gb
LOGOVIEW NT	WINDOWS NT	32bit	Pentium-SVGA-HD540Mb	32Mb/1Gb
LOGOVIEW NT	WINDOWS 95	32bit	i486-SVGA-HD250Mb	16Mb/1Gb

As shown, *Logoview* can meet all requirements both in terms of performance and in terms of financial savings.

The various versions are in a position to communicate among themselves allowing modular composition of the various production levels.

It should not be left unmentioned that the WINDOWS NT version is not only able to take advantage of the parallel processing provided by the operating system, but is also able to run on other hardware platforms, as long as they are meeting the Microsoft NT specifications.

2.1.1 Basic characteristics

LOGOVIEW NT has been designed to take full advantage of the powerful features of the new Microsoft operating systems. This means that we are not facing here a mere conversion of the DOS package, but rather a totally new version. In practice, ***LOGOVIEW NT*** has been subjected to the same operation applied to WINDOWS 3.1 at the appearance of WINDOWS NT. The User sitting in front of the computer encounters an environment already known to him, consistent with the previous ones, without seeing the engine below.

The aim was to transfer to the WINDOWS environment the performance of *Logoview 32*, which, as far as execution speed is concerned, is placed at the top of the range. The whole package has been written in C++ using the MICROSOFT MFC 4.0. The functional testing both of the development package and of the execution core (which allows the application to run) has been carried out using all MICROSOFT tools, reaching an extremely deep package integration with the operating system.

Particular care has been taken by the designers in order that the applications written with ***LOGOVIEW NT*** look like native WINDOWS programs to the maximum extent.

2.1.2 New functionality

LOGOVIEW NT has been designed to fully take advantage the MICROSOFT operating systems resources. Even keeping all characteristics of maximum flexibility and configurability, it allows to use all tools usually found in the most common packages. This allows the developer to create applications which look and feel in line with the whole working environment. The main characteristics built into ***LOGOVIEW NT*** are the following.

- *Ole objects embedded into the layouts. This feature allows to insert graphs, documents and other types of functional objects not readily available from **LOGOVIEW NT** directly in the screen layouts. If, for instance, you want to insert an EXCEL™ sheet in a layout keeping the possibility of working in it without the need of opening the program, you can do so using this function. It is required only that the EXCEL™ program is residing in the same computer.*

- *Sharing of data processed by **LOGOVIEW NT** with other packages available under the WINDOWS environment by means of the DDE or Ole Automation protocols. This makes it possible reading the variable contents by other programs. This way you can, for instance, read the variables with a spreadsheet or with a network terminal and use them to create graphs.*

- *Network distribution of the control system by means of the NET-DDE feature. In case of large plants, you can install several **LOGOVIEW NT** stations each controlling a dedicated plant section, and still having all pieces of data generated by any station available for reading by all other network connected stations. This way any Logoview can display data acquired by others.*

- *Capability of dealing with multimedia objects readily integrated in the package. Capability of directly using sounds or video clips embedding them into the screen layouts.*

- *Capability of applying any data base, either directly by using a format compatible with ACCESS™ or Visual Fox-Pro™ or by means of protocol ODBC (optional) in client/server format by means of any among the most popular SQL-Servers (Oracle™, Ingres™ and so on).*

- *Availability of tools for the integration of gadgets such as Context Help, Tip of the day, ToolTips and Splash screen into the application in order to provide it with look and functionality in line with the MICROSOFT specifications for WINDOWS interfaces.*

- *Automatic building of the application skeleton by means of Wizard™. It consists of an application within an application. When a new application is being laid down, Wizard™ displays questions and on the basis of the answers provided by the User builds an already working application skeleton, ready to be integrated with the functions and objects one has in mind.*

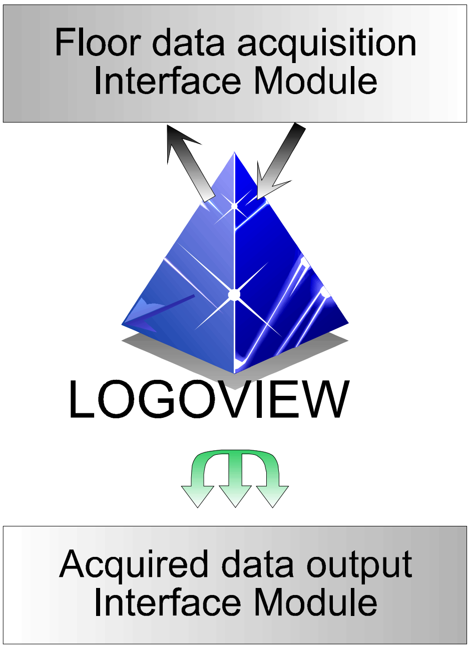
- *Full advantage taken from the window interface, with the capability of monitoring up to four concurrent screen layouts from inside any individual application, on four separate windows in MDI architecture. A dedicated graphic engine has been studied to extremely speed up the various animations refreshing, without slowing down the other **LOGOVIEW NT** functions because of that.*

2.2 How LOGOVIEW NT works

In the past, it was common practice for the buyer of a SCADA system to evaluate the system functionality and performance not considering the extent to which its architecture was open. No big deal was considered if its software implied proprietary solutions or if the hardware was practically tying the buyer to a single provider. And almost all the times the promise that the system would be capable of providing solutions not only to the present problems (in most cases true) but also the future needs would result to be false. It was not rare that the system would begin showing its own limits in a quite short time. The spread of new platforms, along with the success of operational environments entirely different both in terms of look and functionality have put the User in front of a serious predicament, the one of choosing a standard which on one side guarantees to meet all needs and on the other side assures the possibility of constantly updating it by means of the integration of new network platforms, data base management systems and graphic interfaces. This is truly the objective that the Logoview designers had put in front of them since the very first versions, which has been met to a large extent. In effect, all Users choosing it as their own development platform have not been disappointed. Version after version, they have always had at their disposal one of the most technologically advanced systems, in no case having to regret their investment, neither in terms of personnel training nor in terms of re-using code written for the old package versions. In this chapter we will try to introduce the basic concepts at the base of the package putting the User in the condition to understand its inner logic. This will make it easier to learn the subsequent programming.

2.3 Structure and Basic Concepts

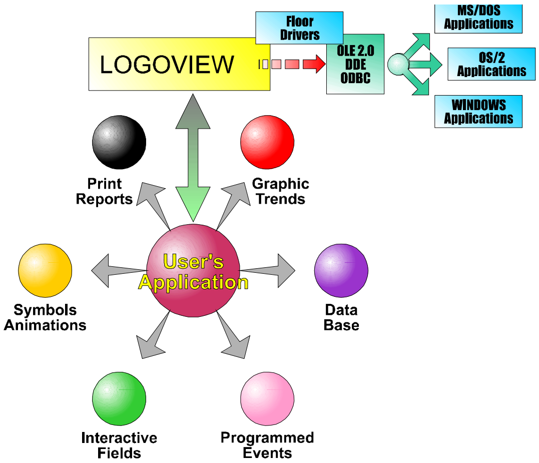
Let us first explain the structural way ***LOGOVIEW NT*** deals with the various objects the program will include, as applicable. In general, by *application* we mean a set of several objects, characterized by their own *life*. These objects are composed of floor data acquired by the system or data entered by the operator, or, more generally, data resulting from raw data processing. ***LOGOVIEW NT*** wraps up, if one can say so, these pieces of data in a way comfortable for the User. Following certain rules set forth by the package programming, it changes them into objects which will later be used for displaying, archiving and further processing.


Floor data acquisition Interface Module
LOGOVIEW
Acquired data output Interface Module

In the end, we can look at the Application as a set of pieces of data ordered according to the User specifications.

In reality, by stressing this simplification, we could say that the task of a supervision program is to read some data and display them in various ways. This sentence is certainly true, although what is meant by “various ways” makes up the difference between a package and another. Starting right from this last sentence, let us first evaluate the sketch of how the functioning engine of ***LOGOVIEW NT*** is structured. We can divide the package into three main blocks. A central module containing all data processing routines. Then, proceeding towards the floor, we find a block dealing with floor data acquisition and storage which also has the task of changing their formats to be usable also by the central block. In this broad exposition we mean by *floor* any piece of information residing outside the system, hence all input devices make part of this block, including mouse, keyboard, PLC communication drivers and network communication drivers.

Finally we have a third block having the task of preparing data according to the User’s program and putting them into the output devices, such as disks, screen and printers.



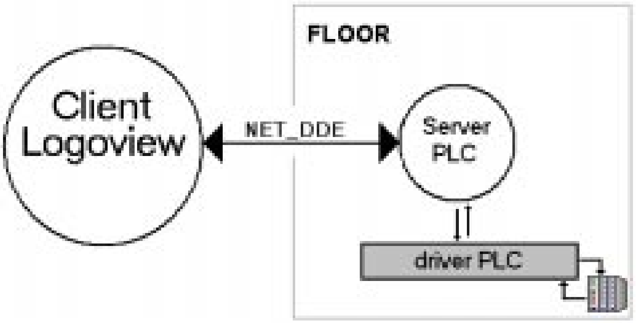
As already said, if we want to comply with such a simple sketching description, we must also simplify in an extreme way the explanation of the package functionality. However, if we were analyzing each of the three main blocks, we would discover that they are also composed of a number of subsystems, each of them dedicated to perform specific functions. The processing block, as easily understandable, is made up of the procedures defined by the developer and, consequently, will be not further analyzed. Let us instead expand the blocks dealing respectively with the floor communication and the Operator display.

2.4 Communication with the outside world

In environments like WINDOWS 95 or WINDOWS NT, the management of communications between the application and the outside world is done by the operating system, at low level. However, this is true only for the standard I/O ports, such as serial and parallel ports and keyboards. As far as all the other devices commonly utilized in the industry, the operating system is not helping at all. This ends up producing a great quantity of “dialects”, required by the various communication protocols required by the various devices. Given the current vitality of the market, every day there are new devices made available by companies specialized in instruments and accessories for automation. Not always the SCADA packages are ready to interface with them. Instead, thanks to the way ***LOGOVIEW NT*** is designed, the effort required to interface it with any kind of device is minimal. Indeed, the block for managing communications not only is capable of using the standard features of the operating system (OLE, DDE etc.) but is also provided with a protocol through which it is possible to write a communication driver between ***LOGOVIEW NT*** and the external device. The know-how required to do so is minimal. Previously, Drivers and supervisor environment were strictly connected, making them highly dependent from each other. A problem coming from the driver could easily influence the supervising environment, thus producing serious problems. ***LOGOVIEW NT*** has been designed to separate the plant controller (***LOGOVIEW NT***) from the device reading floor data (PLC), and this choice has produced a client-server architecture.

2.5 Client-server architecture

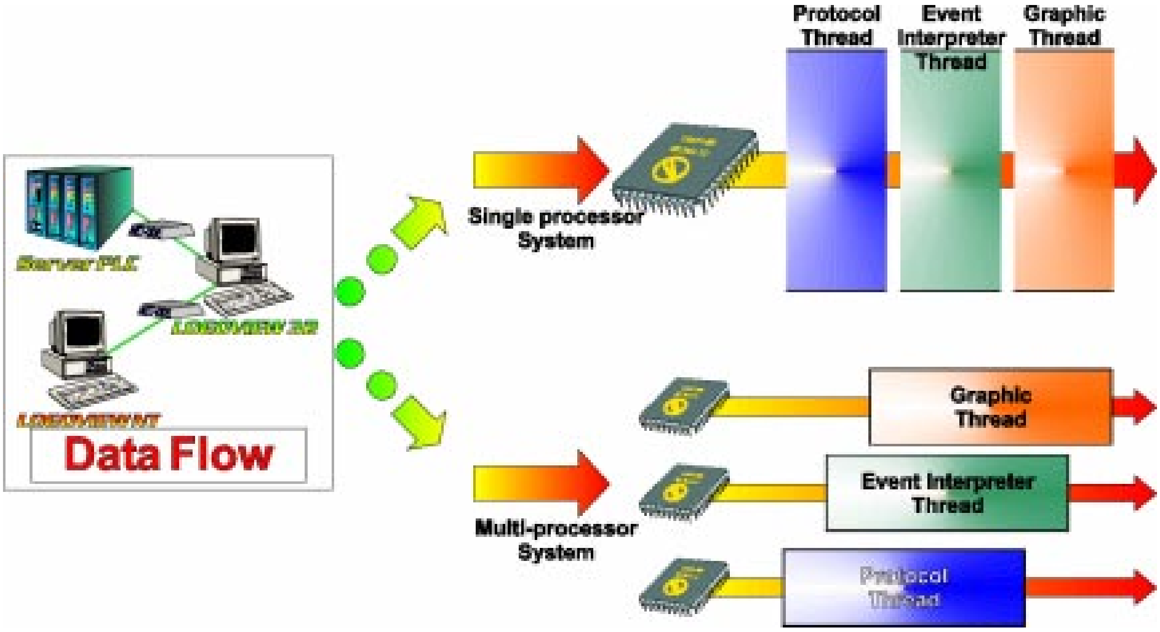
The client-server architecture devised for ***LOGOVIEW NT*** has assigned the management of PLC drivers to the communication servers. Therefore, ***LOGOVIEW NT*** communications are managed using a client-server protocol. All drivers are not controlled directly by ***LOGOVIEW NT*** but are managed by their own communication servers.



The above is a typical example of a client-server architecture: ***LOGOVIEW NT*** and PLC server communicate with a NET_DDE protocol, which manages the data flow.

Actually, all data read from the floor, both numerical and alphanumerical, is stored by ***LOGOVIEW NT*** in huge variables lists, every variable addressable individually. The point here is how to make sure that this data can be stored into the variables quickly and without possibility of errors. This is the duty of the drivers, which are programmed to read data from the devices and pass it to the server, which in turn will transfer it to ***LOGOVIEW NT***. This way data is collected asynchronously with the application. ***LOGOVIEW NT*** will acknowledge a variation in the currently displayed variables, and will update them. Evidently, every driver is a separate program, and given the type of connection between server and ***LOGOVIEW NT,*** it is not necessary for the driver to reside on the same machine running the main application. It can very well be located in a remote machine, connected via network. Furthermore, there are no limits to the number of active drivers, besides of course the hardware limits of the machine on which the application is run.

First of all, this type of architecture allows a complete separation between the application software and the kind of data collection device. Also, it allows more than one ***LOGOVIEW NT*** station to access the same external device. Furthermore, each ***LOGOVIEW NT*** station can, in turn, be both client and server, thus guaranteeing an effective operativeness under any circumstance. By using this protocol ***LOGOVIEW NT*** can make use of *DDE_POKE* and *DDE_EXECUTE,* allowing access to other programs residing on remote nodes. Within all these procedures ***LOGOVIEW NT*** fully utilizes the MULTI-THREADING capabilities of Win32, and is also capable of accessing the multi-processing of NT, if the machine has it.



2.6 Floor communication configuration

Previously we mentioned the floor communication configuration. Let us describe here what we mean by that, and which actions are required. First of all, we must remember that basically two things can be done: read and write data. The field device is irrelevant since the only duty of a driver is to manage the I/O flow of data. What is important is instead the mode and frequency of the data flow. For this purpose, the drivers developed following the specifications listed in the *Device Driver Kit* are equipped with an interface for configuring the data flow. This interface allows to choose whether the driver reads and writes in a timed manner, or asynchronously depending on the requests of the application.

The timed reading is configured by selecting the time interval for updating a selected group of variables, for example 2 seconds. The driver itself will set up a timer to satisfactorily answer this request by interfacing with the appropriate device. As soon as the timer reaches the pre-set value, the driver will send a read request, and will inform ***LOGOVIEW NT*** if the values have changed. This way the application is not slowed down by updating values which have not changed.

The asynchronous mode instead allows reading and writing groups of variables at any given time, by informing the driver of the request, which will be carried out as soon as possible. This method is slightly more complex from the application view point, but is necessary when the display of data depends on Operator request, or on external events. Obviously the two modes of operation can coexist. For writing a driver, please refer to the *Device Driver Kit* manual, sold separately. To configure a driver, also please refer to the same manual, which is peculiar to the device interfaced by your application.

2.7 Data Output

The main reasons for collecting data are: to store them for later processing; to have under control the situation described by the data themselves. Raw data, as collected by ***LOGOVIEW NT*** appear in a form which is in general meaningless for the Operator. In order to understand the meaning of each piece of information we must make use of some graphic representation, such as interactive fields, trends, alarm, etc. The system designer, when programming the application software, must make sure that all variables are correctly presented on screen and on paper. To this purpose ***LOGOVIEW NT*** can link with a number of software devices to manipulate the display of any kind of object. To simplify programming, similar objects have been grouped in families.

These families contain simple objects, using which the User can create a more complex new object, by combination.

Before examining in detail the various objects, let us take a look at the different kinds of outputs and their use. In general, there are three kinds of outputs: 1) on screen (or another type of display); 2) on a magnetic device; 3) on paper. The output on screen or another display is used to visualize the data in real-time. In this case the data read are formatted and displayed on screen lay-outs as soon as they are available to ***LOGOVIEW NT***. Every update will modify the displays, thus showing only the current situation. The data are usually displayed in a graphic manner on screen lay-outs, also named synoptic in ***LOGOVIEW NT***. These screen lay-outs usually depict in a schematic form the portion of the plant the data come from so that each variable is displayed in the vicinity of the symbol representing the physical object producing the datum itself using a color coding related to the state of the object (ON, OFF, hot, cool, etc.). Animations of the symbol can also be used. The purpose is to display the floor on screen in a most accurate way, depending on the available hardware. The magnetic support is used instead to store the various situations. As said above, on screen we have only the real-time situation, while it can be useful to compare different situations occurred at different moments in time. By saving packets of data on disk we can create Historical Archives, containing selected pieces of information, which represent the history of the plant. This allows the User to review the past events in any given moment, and also to simulate the past events. The output on paper can provide a report of the current situation, or a report of any given situation happened in the past, taken from an historical archive. The application will be made up of all of them.

3. *LOGOVIEW NT* Installation

This chapter provides information necessary to correctly perform the installation of the ***LOGOVIEW NT*** environment.

3.1 System Requirements

Logoview needs the following minimal configuration to work properly:

- *PC with 80486 CPU or higher*
  Pentium recommended

- *Graphic board VGA*
  Accelerated SVGA recommended

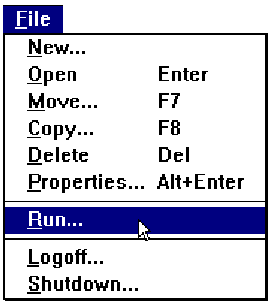
- *16Mb memory with Windows 95, 24Mb with Windows NT*
  32Mb recommended

- *Hard Disk*
  Available space at least 20 megabytes

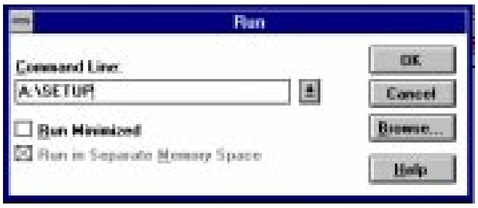
- *Operating System Windows NT 3.51, NT 4.0, Windows 95*

3.2 The Installation procedure

The installation procedure is quite simple and fully automatic. Hence, no specific knowledge is required. Before proceeding, make sure that at least 20 megabytes are free on disk.



The installation takes place under the WINDOWS operating system. Consequently, after starting the operating system, select the Run command from the File menu of the Program Manager. This action will display the application starting window. If the installation disk is located in drive A, enter “A :\SETUP”.



Command Line: A:\SETUP

□ Run Minimized

☒ Run in Separate Memory Space

Or use the “Browse...” button to locate the installation program in another drive.

3.3 The *LOGOVIEW NT* components

LOGOVIEW NT is consisting of 3 components, each included in the basic package.

Development Environment gdevelop.exe

Logoview NT Development

Allows to develop Logoview NT applications and to configure their related implementations

Runtime Environment logorun.exe

Logoview NT RunTime

Allows to run Logoview applications and interact with them

Online Help <application's name>.hlp

Help

Provides online help about all selections available in the various applications

Note. An additional application module is available, separately packaged and sold, allowing to draw screen layouts and to build vectorial objects libraries. Its name is FlashDraw. Additional information on this package is available at your nearest reseller.

3.4 The Uninstall Procedure

The uninstall procedure provided with ***LOGOVIEW NT*** is required to correctly remove all parts of the application. The procedure is fully automatic and does not require particular knowledge. Once removed, ***LOGOVIEW NT*** can be installed again by the installation procedure described above.

The removal occurs under the WINDOWS operating system. To start it up, simply double click on the icon present in the Logoview group.

Uninstal

Or, look for the uninstall program in other drives using the "Browse..." button.

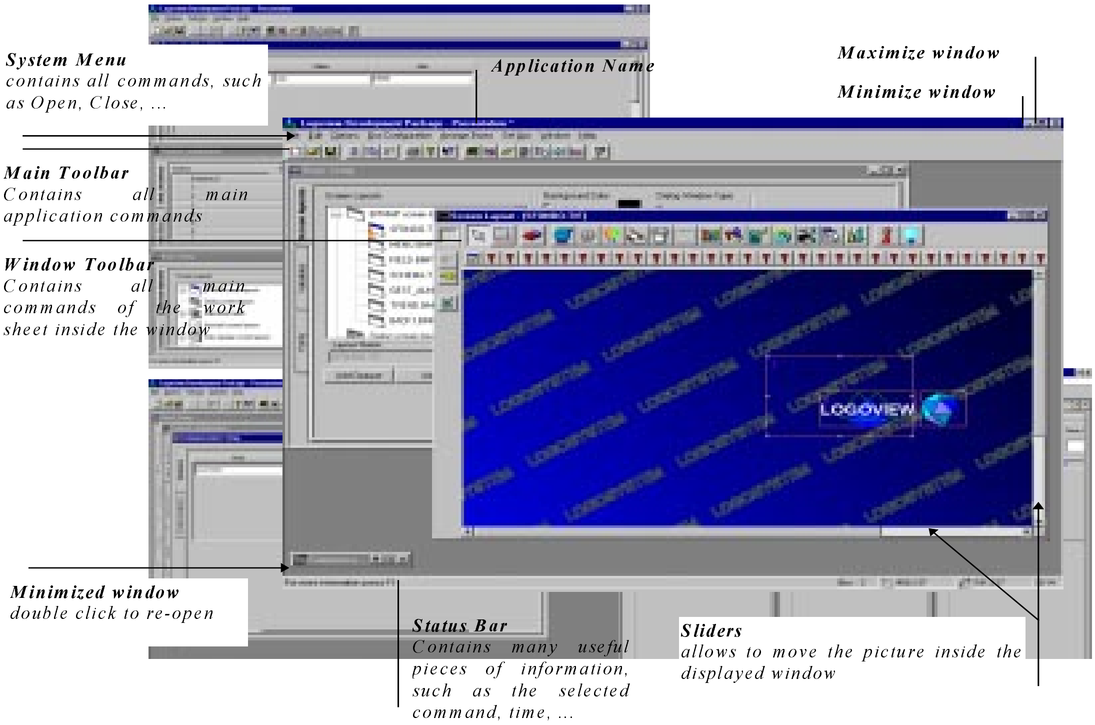
4. Using the Interface

This chapter provides the bulk of information on how to use the interface of a Window Application such as ***LOGOVIEW NT***. The information which follows should be considered as complementary to that provided by the *WINDOWS* manual. Readers already familiar with the use of the Interface could skip the next chapter.

4.1 The philosophy

LOGOVIEW NT, as all other Applications developed for *WINDOWS* makes use mainly of the mouse as User interaction resource. The mouse allows access to all Application's functions, moving, copying, re-sizing all objects present on the workbench. First of all, therefore, you should get acquainted with the mouse use. A workbench is usually contained in a window. The window constitutes the main communication medium in a *WINDOWS* application. Windows too can be moved, resized, closed ... and so on by means of the mouse. A few commands are also available to obtain automatic placement of the Application windows. Such commands are usually contained in the *Window* menu.

4.2 The application and its windows



System Menu
contains all commands, such as Open, Close, ...
Application Name
Maximize window
Minimize window
Main Toolbar
Contains all main application commands
Window Toolbar
Contains all main commands of the work sheet inside the window
Minimized window
double click to re-open
Status Bar
Contains many useful pieces of information, such as the selected command, time, ...
Sliders
allows to move the picture inside the displayed window

The application exchanges information with the User by means of windows designed and dedicated to this purpose. The main controls are placed in various toolbars for user convenience. Each button is associated to a single function control. Each toolbar control is normally available also as a menu entry.

Communications

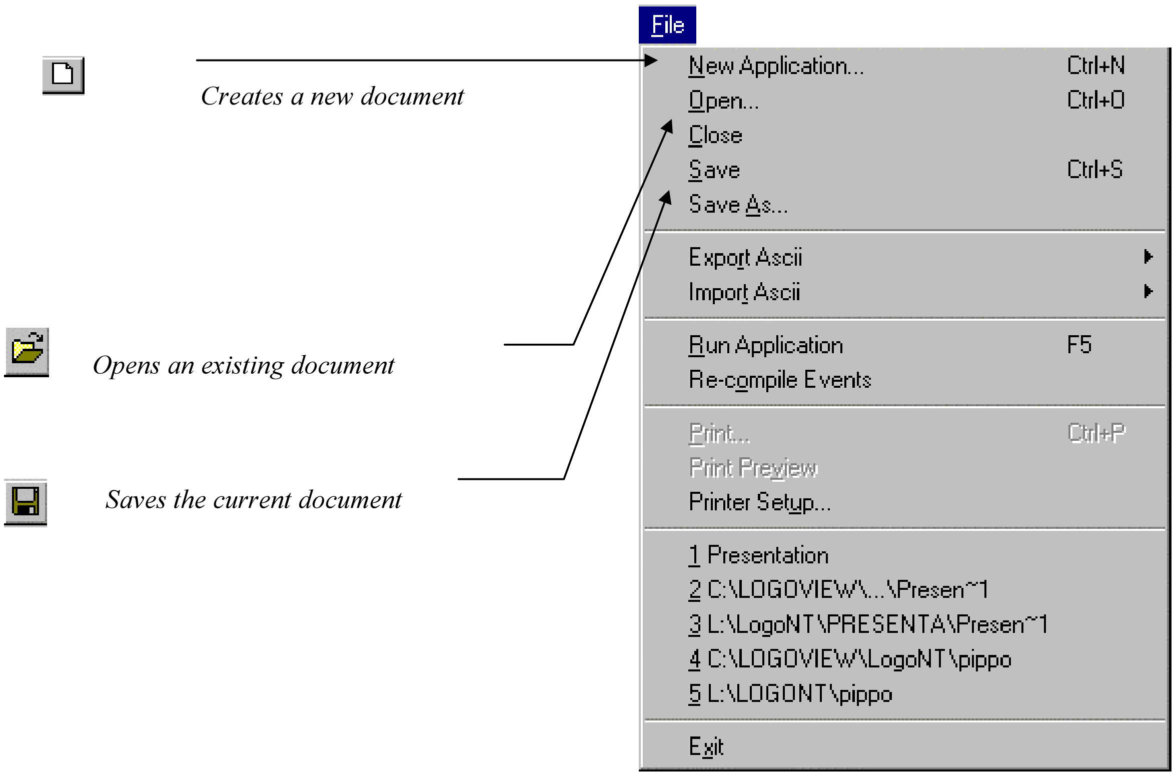
configuration

Each window, as well as the whole application, can be reduced to icon by pressing the button ▼ . To restore the previous size just double click on the icon corresponding to the window. Instead, to make the window as large as possible, press the button ▲ .

4.3 Menus and Toolbars

As mentioned, there is a tight connection between menus and toolbars. Usually the controls available in toolbars constitute a subset of those available in menus, that is the most important ones.

The most important controls present both in the File menu and in the *Main* Toolbar are the following.



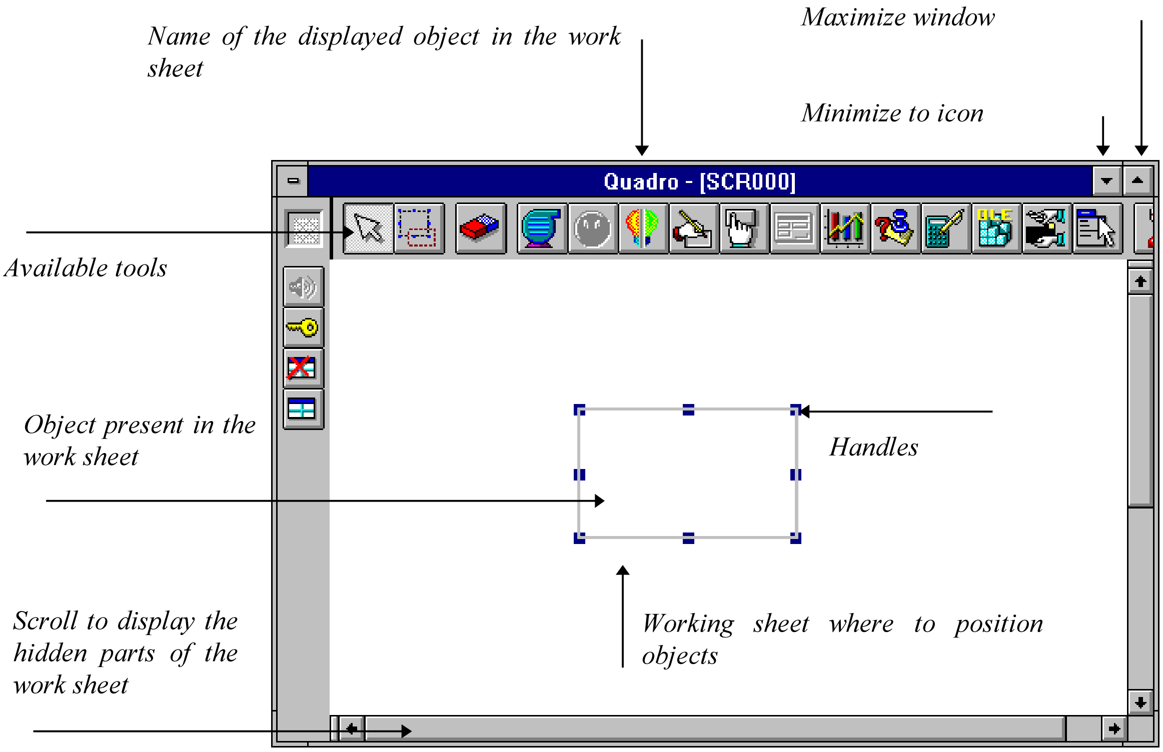
Creates a new document

Opens an existing document

Saves the current document

4.4 The Workbench

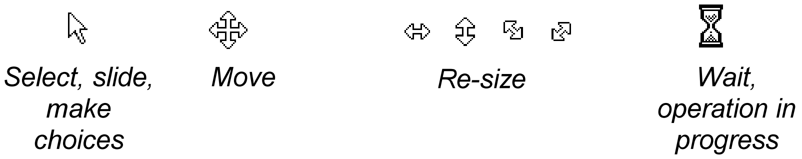
The workbench can take various forms, depending on the actual context. For example you will find a workbench allowing to build screen layouts. Anyway, whichever the workbench, using the mouse has a fundamental role, since it allows to move, modify, select and handle all contained objects in the most general way.


Name of the displayed object in the work sheet
Maximize window
Minimize to icon
Quadro - [SCR000]
Available tools
Object present in the work sheet
Handles
Scroll to display the hidden parts of the work sheet
Working sheet where to position objects

LOGOVIEW NT displays the necessary tools in the parts of the workbench window where they are most easily seen and functional.

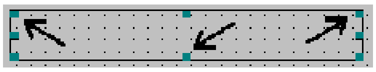
The mouse is the quickest means to interact with the workbench. We present now a broad outlook of the general as well specific techniques of mouse using and a description of the various mouse cursor formats. It needs to be taken into account, anyway, that they provide general hints, common to the majority of cases..

The mouse cursor shape denotes the action being performed.

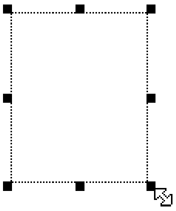


Objects located within the workbench can be selected for subsequent operations (for instance removal from screen).

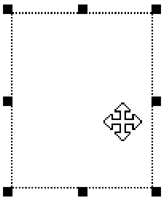
To ***select*** an object you only have to move the mouse cursor on the screen until pointing to the object to be selected and press the left mouse key. The selected object will be highlighted by some little boxes called ***handles***.



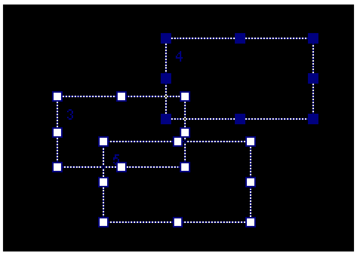
The handles will appear along the border of the selected object. One of the handles functions, besides the one of showing which object is selected, is to allow changing the object's field size. In practice, while the cursor is passing on top of any handle, the cursor shape will change and on the basis of that you will get a direct clue on the re-sizing type.



Grasping the ***corner handles*** and dragging while holding down the left mouse key, you will increase or decrease both sides. Grasping instead one of the ***side handles*** you will obtain a side slide perpendicular to the side itself.



Moving instead the cursor ***inside the selected field***, as soon as you press the left mouse key, the cursor will take the shape of a four heads arrow. Holding down the key and moving the mouse, the whole field will move along until the left key is released.

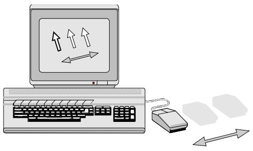


Usually, when multiple objects are present, you can perform a ***multiple selection.*** This is achieved by repeating the selection of various objects while holding down the *SHIFT* key.

The last selected item constitutes the ***reference object*** (identified by filled handles). This will be the reference object for all align operations.

The mouse allows an additional interaction function with the workbench, the ***double click.*** This operation is performed by positioning the mouse cursor on a certain object (no matter which type) and pressing twice and rapidly the left mouse key. This operation is often used to display the object characteristics.

In addition you can get quickly all the object's operational functions at your disposal on the workbench. Just position the mouse cursor on the object and press the right mouse key.



It might be convenient to recall here that if no room is left for further mouse moving, you can simply raise it from the pad and place it back where it can be moved again.

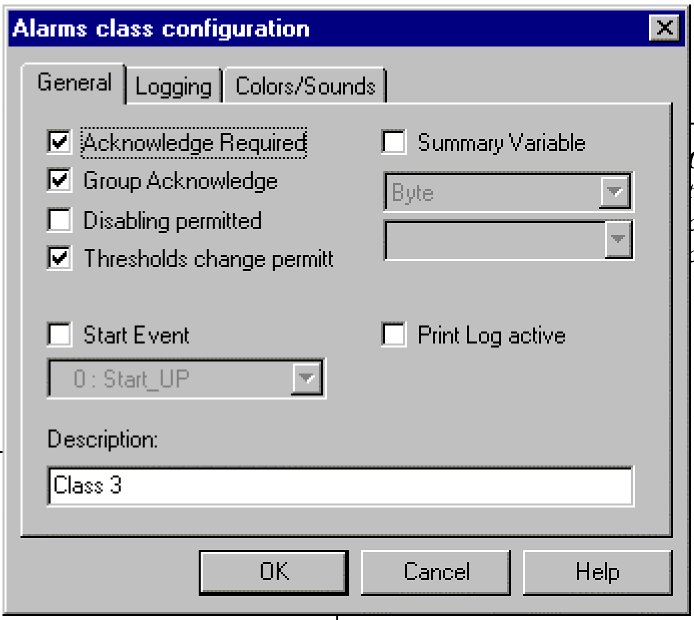
4.5 The Dialog windows

LOGOVIEW NT contains a large number of dialog windows. Their purpose is to ask the User to provide parameters and other information regarding certain actions. They constitute, therefore, another additional sort of dialog between the ***LOGOVIEW NT*** application and the User. The ***LOGOVIEW NT*** dialog windows can be used, for instance, to configure either the set of variables or the safety passwords or other items. A typical example is provided by the alarms class configuration window.

*Indicates that the window is made up of **three sections**. The active section is the one **upfront** (in this case "General")*

__Combo__: allows to choose one option among those available

__Description field__: usually allows free entering of data from keyboard, although, as in the numeric parameters case, entry is forced



Alarms class configuration

☑ Acknowledge Required

☑ Group Acknowledge

☐ Disabling permitted

☑ Thresholds change permitt

☐ Summary Variable

Byte

☐ Start Event

0 : Start_UP

☐ Print Log active

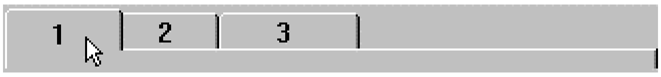
Description: Class 3

__Options__: usually to activate a certain configuration

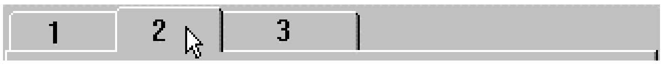
__Buttons__: usually to activate an immediate action. The ones most frequently appearing are __OK__ (to confirm), __Cancel__ (to exit without changes) and __Help__ (help on the current window).

As you can see, this window is made of various objects. First of all, it is composed of three sections (that is, it consists of a combination of three dialog windows).

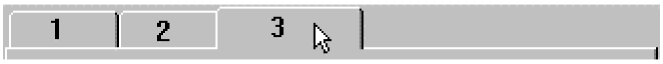
The individual window sections can be reached by positioning the mouse cursor on the index, usually appearing in the upper window part, then pressing the left mouse key, as in the following example:



By positioning the cursor on the first section index and pressing the left key, the connected dialog window will appear in front.



By positioning the cursor on the second section index and pressing the left key, the connected dialog window will appear in front.



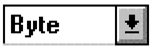
By positioning the cursor on the third section index and pressing the left key, the connected dialog window will appear in front.

As shown in the alarms window example, some buttons common to all sections are usually present.

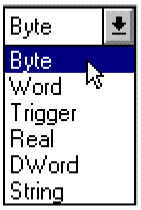
Most common buttons. Usually OK, Cancel, Help

Their activation involves some effect on all window sections.

We provide next a short description of the main objects you can find in a window and the way they are used.:

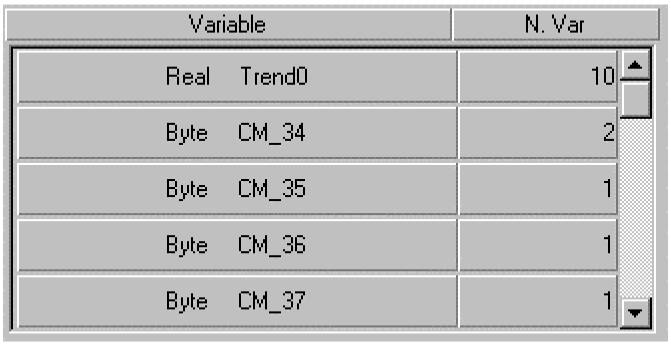


This object is called COMBO. It allows to select an item among many contained in a list. The actually selected item is the one appearing when the COMBO is closed.



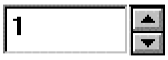
To display the list of the available options just position the mouse cursor on the button ↓ and press the left key.

This action will display all choices present in the COMBO. Next select the one you want.

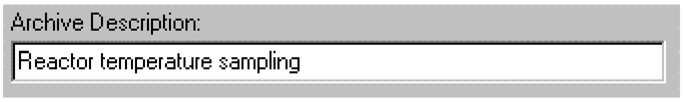


Variable	N. Var
Real Trend0	10
Byte CM_34	2
Byte CM_35	1
Byte CM_36	1
Byte CM_37	1

This object is called LIST. It contains usually a number of other objects displayed in lines. These objects can denote various things (variables, events, and so on). To select an object, position the mouse cursor on the corresponding line and press the left key. LOGOVIEW NT will highlight the selected line. To display the whole list you can make use of the sliding bar, which is automatically drawn any time the lines are so many that the available space would not accommodate all of them. The sliding bar can scroll in both directions. Upward by means of the button ↑ and downward by means of the button ↓



This object allows to enter a number. To do so you can operate in two ways. The first one consists into placing the mouse cursor inside the white box and press the left key. This action activates the keyboard, so you can enter the desired value. Alternatively you can increase or decrease the value by using the small bar, that is pressing the up arrow to increase and the down arrow to decrease.

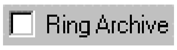


Archive Description:

Reactor temperature sampling

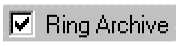
This object allows to insert a series of alphanumeric characters, usually to enter a description or a name. To insert, position the mouse cursor inside the white box and press the left key. This action will activate the keyboard. Then enter the sequence of desired characters.

This object denotes an option, assuming one of the following forms:



☐ Ring Archive

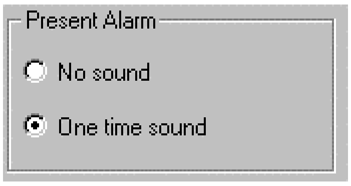
An option currently not active is shown by a white box..



☑ Ring Archive

An active option is indicated by a small mark inside the white box.

To activate or deactivate the option, position the mouse cursor inside the white box and press the left key.

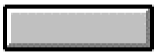


Present Alarm

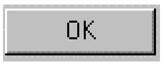
○ No sound

● One time sound

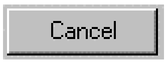
The one above denotes a preference group. This object allows to select one among many alternatives. To select a preference, position the mouse cursor on the corresponding little circle and press the left key. LOGOVIEW NT will signal the new preference by filling the circle with black color. The choice is exclusive, that is only one circle appears filled at a time.



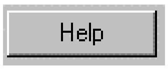
This object is a button. To press it, just position the mouse cursor on it and press the left key. The resulting effect is usually immediate. The most common buttons are:



Confirms the window choices and closes it



Cancels all last setup operations and closes the window



Provides help on the window controls

4.6 The Uninstall Procedure

The uninstall procedure provided with ***LOGOVIEW NT*** is required to correctly remove all parts of the application. The procedure is fully automatic and does not require particular knowledge. Once removed, ***LOGOVIEW NT*** can be installed again by the installation procedure described above.

The removal occurs under the WINDOWS operating system. To start it up, simply double click on the icon present in the Logoview group.

Or, look for the uninstall program in other drives using the “Browse...” button

PART II

Development Environment

5. Development Environment

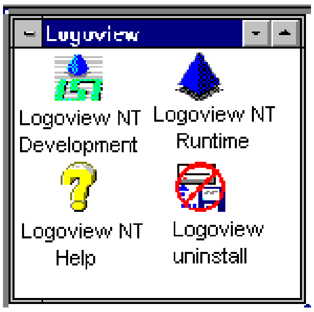
LOGOVIEW NT contains all necessary tools to develop a complete supervision environment.

5.1 Start the Application

LOGOVIEW NT environment is started like any other *WINDOWS* application: just double click on the icon.

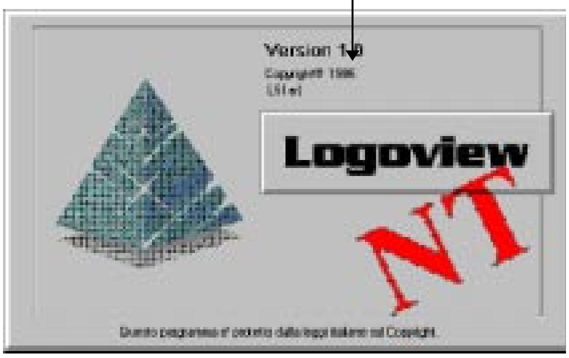
Icon for the Development package
Double click to start

The application icon can be found in the ***LOGOVIEW NT*** group created by the installation procedure. The group will appear in a shape depending on the operating system being used.

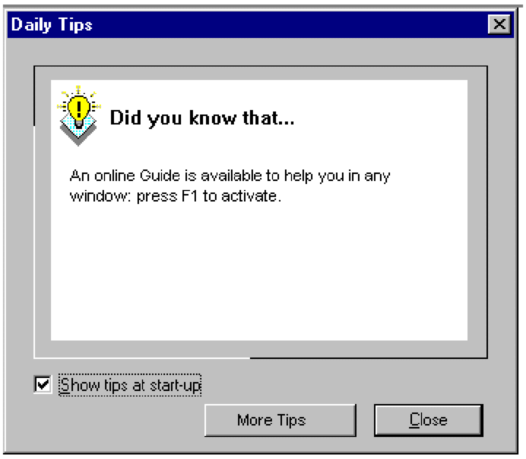


A new user to WINDOWS will find all information necessary in the Microsoft WINDOWS User Manual or in the on-line Tutorial.
At startup, ***LOGOVIEW NT*** will display the main screen containing the program logo.

Version and Copyright



Also, if the option is enabled, the daily tip is shown, containing a brief suggestion, different every time, on using ***LOGOVIEW NT***.



Did you know that...

An online Guide is available to help you in any window: press F1 to activate.

The Development Environment will be displayed upon closing the above window.

If an application has been previously saved, ***LOGOVIEW NT*** will load it, otherwise, it will begin creating a new application using the *Wizard.*

The Tip.

Brief description of a LOGOVIEW NT function. A new tip is provided any time LOGOVIEW NT is started

Choose if you want this window to be displayed at startup

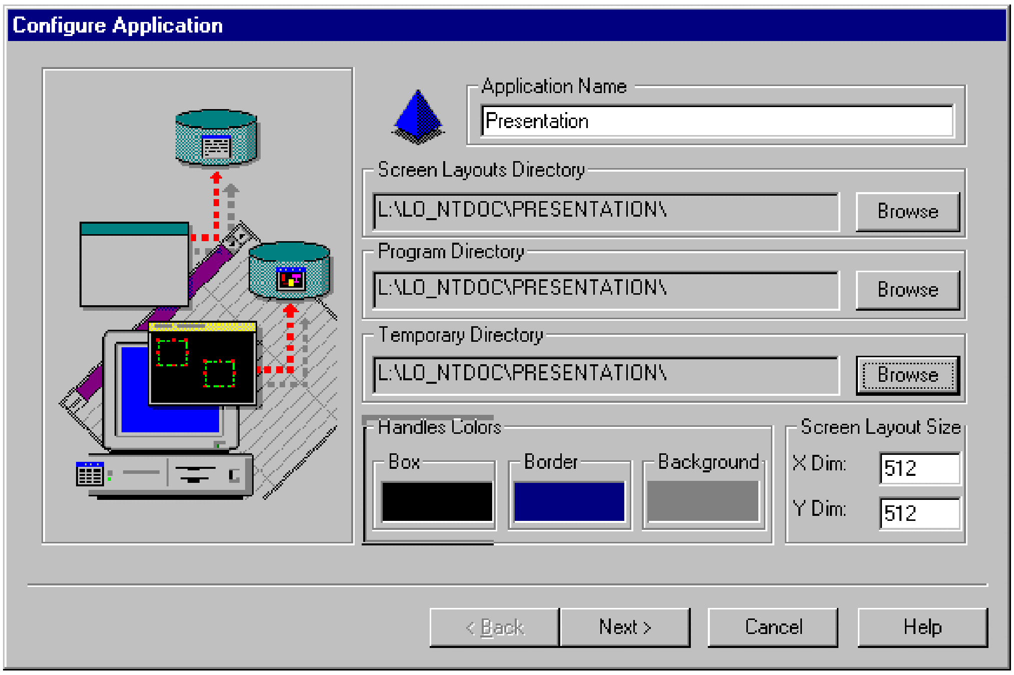
Displays another of the available tips

Close the window

If you do not want to create a new application using the *Wizard* (described later in the manual) just click on the button

Cancel

to be found in the lower part of the first window.



Configure Application

Application Name: Presentation

Screen Layouts Directory: L:\LO_NTDOC\PRESENTATION\

Program Directory: L:\LO_NTDOC\PRESENTATION\

Temporary Directory: L:\LO_NTDOC\PRESENTATION\

Handles Colors: Box ■ Border ■ Background ■

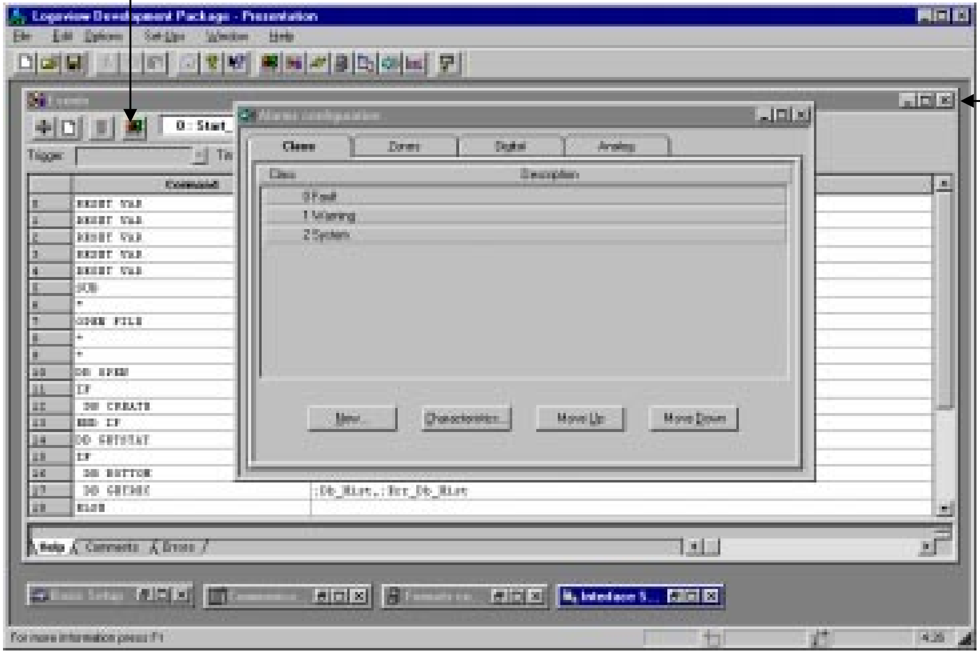
Screen Layout Size: X Dim: 512 Y Dim: 512

5.2 Environment Windows

LOGOVIEW NT is made up by various sub-environments. Each of them is displayed in a window. Every environment window allows to configure a specific part of the application ***LOGOVIEW NT*** being developed. Every environment window can be manipulated (enlarged, compressed, closed, etc.) like any other Windows window.

Toolbar
Shows window specific tools

The most important windows available are:



Maximize and minimize

Basic Setup

Allows to configure screen lay-outs, variables and fonts

Events Editor

Allows to configure the application's events

Communications

Allows to configure the servers connected

Formats

Allows to format Historical Archives, Data Base and Printouts

Custom Menus

Allows to configure the ***LOGOVIEW NT*** *application interface*

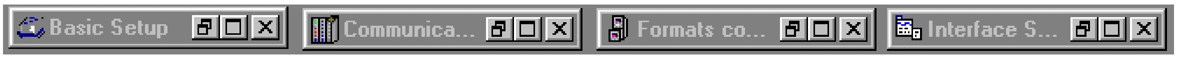
Alarms

Allows to configure the alarms displayed in case of malfunctioning

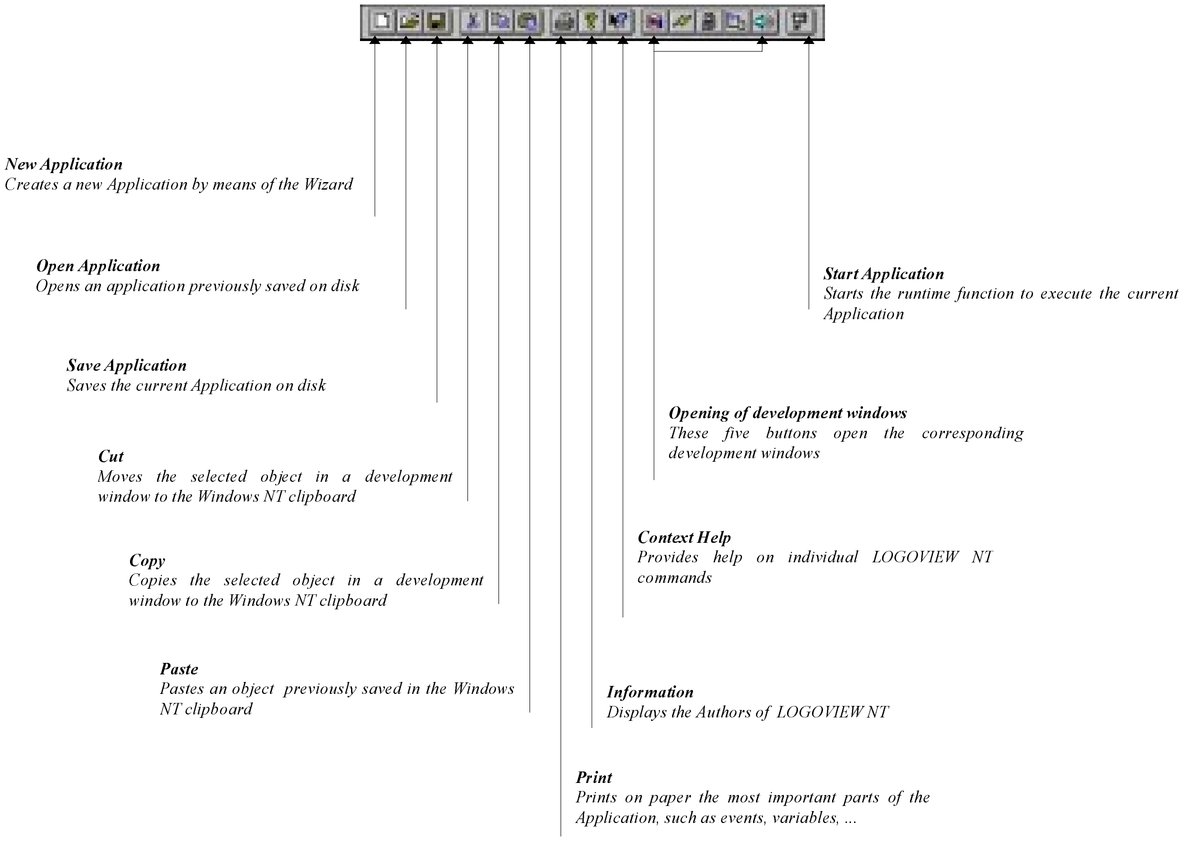
The environment windows are easily accessible by using the buttons located in the *main Toolbar,* which contains the more important commands. There are also different toolbars inside every environment window, containing its main commands.

The Basic Setup window cannot be closed. It can only be reduced to icon.

The various windows, minimized with the button ▾, will appear as follows:



5.3 Main toolbar



New Application
Creates a new Application by means of the Wizard

Open Application
Opens an application previously saved on disk

Save Application
Saves the current Application on disk

Cut
Moves the selected object in a development window to the Windows NT clipboard

Copy
Copies the selected object in a development window to the Windows NT clipboard

Paste
Pastes an object previously saved in the Windows NT clipboard

Print
Prints on paper the most important parts of the Application, such as events, variables, ...

Information
Displays the Authors of LOGOVIEW NT

Context Help
Provides help on individual LOGOVIEW NT commands

Opening of development windows
These five buttons open the corresponding development windows

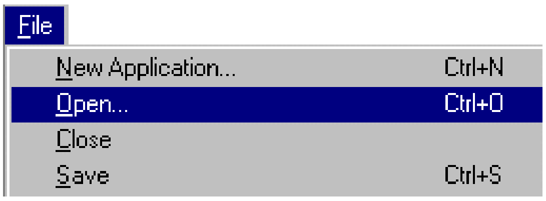
Start Application
Starts the runtime function to execute the current Application

Some of the buttons in the *Main Toolbar* may be inactive, if essential elements of their configuration are still missing. For example, if no screen lay-outs are defined, the Alarms button is not available. For this reason, please configure at least one screen lay-out and some variables before accessing the various environment windows.

5.4 Creation of a new application

As anticipated, a new application is created by pressing button 🗋 from the *Main Toolbar*. The subsequent actions are guided by the *Wizard.*

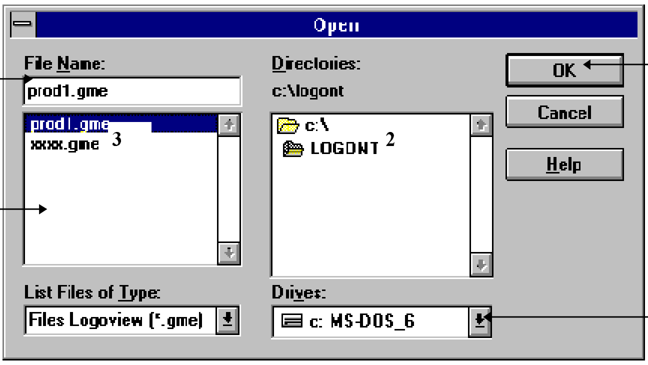
5.5 Opening an existing application



By existing application we mean an application created and saved on disk to be reused or modified at a later time. While using ***LOGOVIEW NT*** you might think it convenient to break the development work and save it. Then, using the dedicated button in the *main toolbar* (or else the appropriate entry in the *file menu*), you can open one among the applications listed and residing in one of the disk directories. The Open command selection displays the standard file window in which you can select the application to be opened.

File selected for opening

List of the applications residing in the selected directory. Choose the one you want to open



Open

File Name: prod1.gme

prod1.gme
xxxx.gme 3

Directories: c:\logont

c:\
LOGONT 2

List Files of Type: Files Logoview (*.gme)

Drives: c: MS-DOS_6

To confirm the opening of the selected file (4)

To select another unit

Hence, the application opening phases can be summarized in the following way::

1. ***Choose <u>Open</u> from the file menu (or press the corresponding button)***
2. ***<u>Select the directory</u> containing the application file***
3. ***<u>Click on the application file</u> present in the file list***
4. ***Press the<u>OK button</u>***

What if ....

The application you want to open is not found

The file could have been saved in another directory. If you can not locate the file, look for it perhaps changing the unit, helping yourself with the ***File Manager*****.**

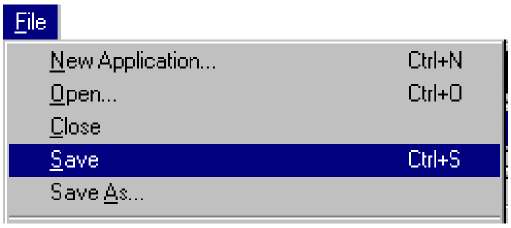
What if ...

After choosing an application, the following message appears:

"An error occurred opening file. Impossible to continue the current operation"

When this (or similar) message appears, most likely the file to be loaded is damaged (because of any reason, possibly by another application), or one or several files making up the application have been renamed or moved from their original directories. In this case the application recovery results to be quite difficult, unless a backup copy is available. In case you try to load an application previously developed on another station and later copied to the target one, the available memory might not be sufficient to load the application. In this case the only solution consists into adding new memory to the station.

5.6 Saving the current application



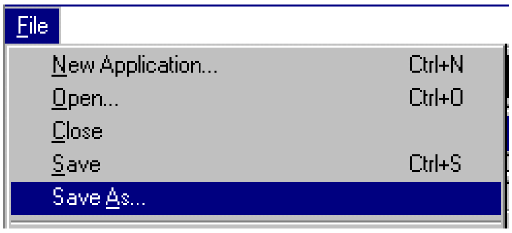
To save the current application, at the same time keeping its name, just press the button 💾 available in the *Main Toolbar*.

Alternatively select the corresponding command from the menu. ***LOGOVIEW NT*** will take care of immediately saving all the application components.

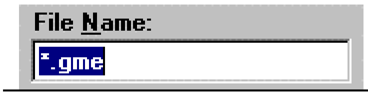
It is advisable to save your application fairly often, or at least any time significant modifications are carried out. This habit will prevent losing the entered changes in case of power breakdown or computer hardware problems..

After saving a file, you can delete it, or copy it to another directory, or to another unit, using the Windows File Manager. Although this can be done, we would like to discourage such actions, since LOGOVIEW NT, along with the program itself, saves also the location in which it resides in order to be able to find it later on. In this case problems could appear while loading. However, copying the application in this way might turn out to be useful if the original file were damaged.

If, after having saved an application, you perform some modifications to it and then want to cancel them, you can easily restore the application previous version. Just load the same application file from the hard disk and confirm you want to do so when LOGOVIEW NT asks you to do so.



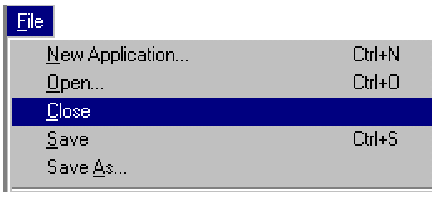
To save the application with a different name, instead, you have to select the corresponding command from the menu. This action displays the standard file window. To save the application with a different name just enter the new name in the ***File Name*** box and press the ***OK button***.


File Name: *.gme

You can not change directory from this window, since the application must reside together with all other items composing it.

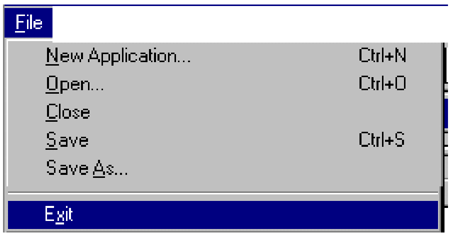
5.7 Close the Application and Exit

You can close an application without having to exit ***LOGOVIEW NT*** as well as close the application and exit ***LOGOVIEW NT*** at the same time.



- ***To close the current application***

1. *Choose* ***Close*** *from File menu*
2. ***LOGOVIEW NT*** *will ask whether to save the changes carried out since last saving*
3. *Choose Yes or No or Cancel (to cancel the saving operation and return to the current application)*



- ***To exit LOGOVIEW NT***

1. *Choose* ***Exit*** *from File menu*
2. ***LOGOVIEW NT*** *will ask whether to save the changes carried out since last saving*
3. *Choose Yes or No or Cancel (to cancel the saving operation and return to the current application*

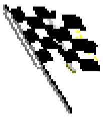
5.8 Development Environment Preferences

The development environment contains a few configurations. These pertain to the whole ***LOGOVIEW NT*** package and are not linked to each individual application.

6. Wizard

A quite flexible tool is made available by ***LOGOVIEW NT*** for the automatic application generation: the Wizard. This tool allows creating a complete ***LOGOVIEW NT*** application skeleton before the developer completes the job according to the application's specific needs. This approach allows creating quite organized and readable applications. This is also the first step towards object oriented programming. Anyway, all the Wizard made setups can be changed later at convenience.

6.1 Create a new application



To create a new application you only have to press the button available in the Main Toolbar, or select the command ***New Application...*** from the ***File*** menu. These actions will activate the Wizard.

The wizard is composed by a certain number of Dialog Windows, each dedicated to configure one particular application section. The User task is to scroll through all dialog windows and configure them according to his own needs. Each dialog window, excluding the first and the last ones, presents two buttons:

The first one to proceed to the next dialog window

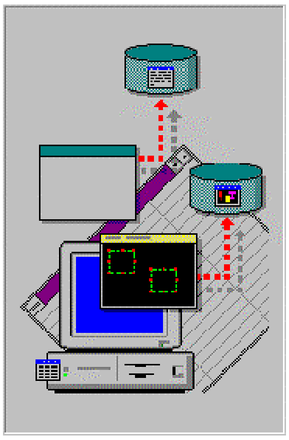
The second one to return to the previous one

Once a window has been configured, you have just to press the button to display the next.

In addition, ***in any window you can always cancel the creation process.***

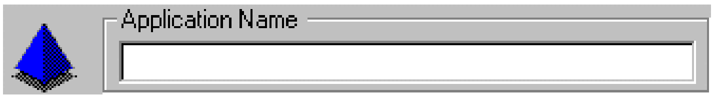
The last window has the Finish button (which actually generates the Application) taking the place of the next window button.

There are eight windows you have to configure in the Wizard before proceeding to the application building.

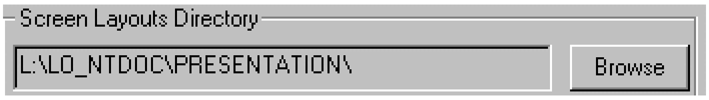


6.1.1 First configuration: Application Configuration

In the first window you ought to setup the main application characteristic, namely its name, working directory, the handles color and the screen layout size (in pixels).



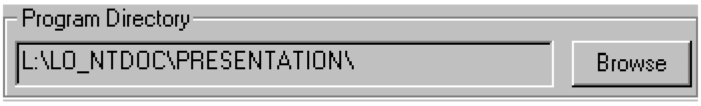
In this box you ought to enter the application name. Proceeding to the next dialog window without doing it is disabled. The maximum field length is eight characters.



Screen Layouts Directory

L:\LO_NTDOC\PRESENTATION\

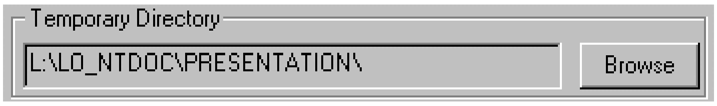
In this box you ought to enter the application's SCREEN LAYOUTS directory. This directory will be used to contain all screen layouts and the graphic parts making up the application along with the files having the extension GMB. Choosing the directory is achieved by means of the Browse button.



Program Directory

L:\LO_NTDOC\PRESENTATION\

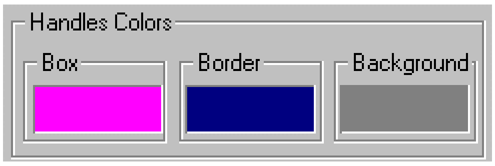
In this box you ought to enter the Application Program directory. This is where the application file (.GME) will be placed, together with the archives and all other files needed by the application but not of graphic type. Choosing the directory is achieved by means of the Browse button.



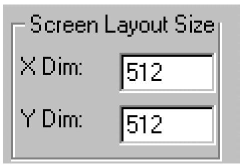
Temporary Directory

L:\LO_NTDOC\PRESENTATION\

In this box you ought to enter the application's SERVICE directory. In this directory all files needed only during development, but not being part of the final application, will be temporarily saved. Choosing the directory is achieved by means of the Browse button.



In this part you can setup, according to your needs, the color of the handles appearing around the selected objects and the XOR color of boxes. To choose a color, just position the mouse cursor on the applicable box and press the left key. This action calls the color window, where you can choose the one you want.



Screen Layout Size

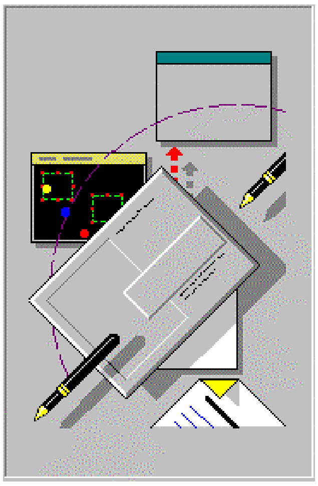
X Dim: 512

Y Dim: 512

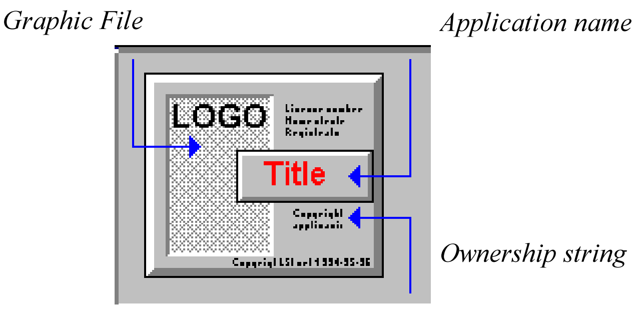
In this section you can setup the application's screen layout size, both height and width. This size is just a reference. It will be possible to change it later on. Values must be between 160 and 1024 pixels.

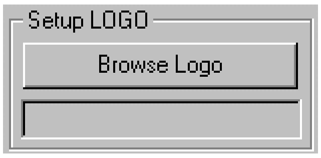
Once this window setup is complete, in order to proceed to the next dialog window, press

6.1.2 Second Configuration: Using the opening Logo

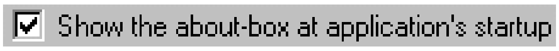


In the second window you have to setup the characteristics of the screen layout which will appear just after the application loading. This screen layout contains several parts: a graphic file, the Logo, the application name entered in the previous dialog window and an ownership string.



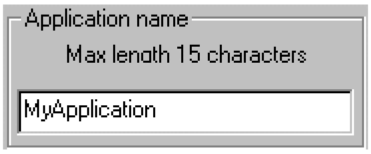


In this section you choose the Logo you want to appear in the first screen layout. The graphic file you want to use as Logo must be located in the Screen Layouts directory. To look for the Logo file use the Browse Logo button. This section will not be active unless the option shown below has previously been selected.



☑ Show the about-box at application's startup

This option makes the application's first screen layout active.



Application name

Max length 15 characters

MyApplication

In this section the application's name is displayed. This name will be displayed inside the screen layout at startup. The name is the one entered in the previous dialog window (First configuration). This can not be changed in this window. To do so you have to return to the first dialog window.

In this box you can optionally enter a short description to be displayed at the startup of the application being created. This might, for instance, contain Copyright information.

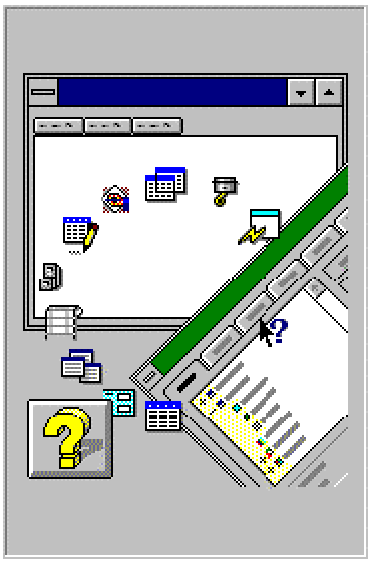
This button shows the startup window the way it has been configured using the present window.

Once this window configuration is complete, press

To proceed to the next dialog window

To return to the previous window

To cancel the creation of the application

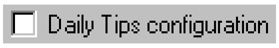


6.1.3 Third configuration: Help and Daily Tips

In this window you can choose if you want use *Context help* and *Daily Tips* in the application being created. The *Daily Tips* will be displayed in a window at the application startup, usually providing short descriptions of the application main features. An application developed with ***LOGOVIEW NT*** can provide such a particular help practice.

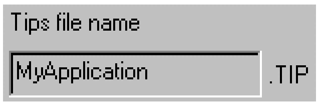
What said above applies also to *Help*: the application being created could provide the User of the application being created some help.

This help makes use of the engine managing the ipertext help under *WINDOWS*. ***LOGOVIEW NT*** allows to create an index file required to link the *help* to the application being created. Then the developer will have to build, following the Microsoft standard, the actual help (which will use the index file) using either a standard editor (for instance Word) or one of the dedicated tools available from the market..
The daily tips file, instead, is a text file in which each line contains a tip. This file can be changed by any text editor (for instance NotePad).



□ Daily Tips configuration

This option, if active, allows defining the Daily Tips contents for the application being created.



Tips file name

MyApplication .TIP

In this box the tips file name is displayed. This file has the same name of the application but with extension .TIP. This is why you are not allowed to enter a new name in this box. The file can be modified using any text editor.

Tips number: 0

In this box you enter the number of tips the application will contain. This number is not unchangeable anyway. During development it will be possible to change it at a later time.

□ Context help config.

This option activates the use of Context Help for the application being defined.

Compilation file name

MyApplication .LHP

In this box the help file name is displayed. This file has the same name as the application but extension .HLP . This is why you are not allowed to enter a new name in this box.

Topics number: 0

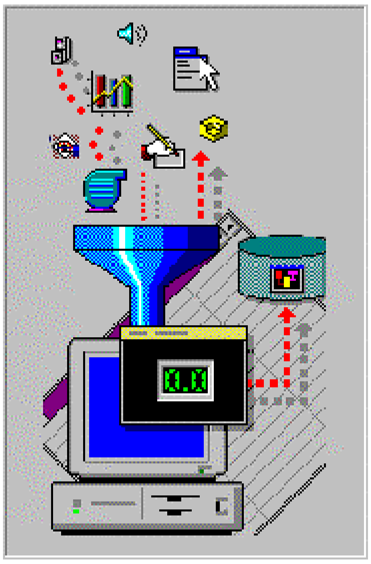
In this box you ought to enter the number of links to the WINDOWS hypertext help engine. This number is not unchangeable anyway. During development it will be possible to change it at a later time.

Once this window configuration is complete, press

To proceed to the next dialog window

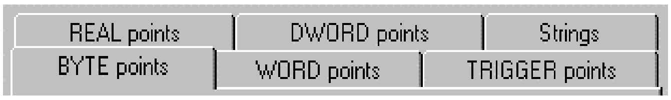
To return to the previous window

To cancel the creation of the application

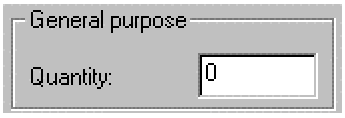


6.1.4 Fourth Configuration : Control Points

In a SCADA program, the control points are the variables used for certain purposes. This dialog window task is to define the quantity and, above all, the purpose of the used variables. Six types of control points are defined (corresponding to the six variable types):

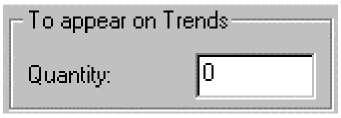


To proceed to the next configuration window at least one control point must be configured. ***LOGOVIEW NT*** will assign different names for each variable purpose. The quantity definition is similar for each available type.


General purpose

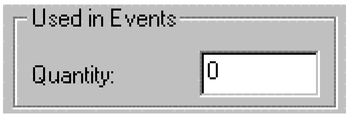
Quantity: 0

Enter in this box the number of variables for generic use you intend to utilize in your program. Variables assigned to this purpose will receive the name CM_(from 0 to the configured quantity).


To appear on Trends

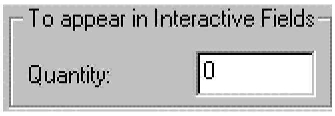
Quantity: 0

Enter in this box the number of variables which will be displayed on trends. Variables assigned to this purpose will receive the name TR_(from 0 to the configured quantity).


Used in Events

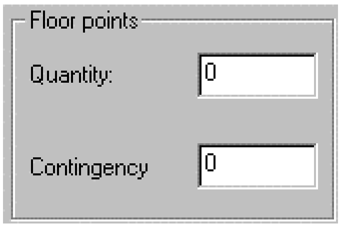
Quantity: 0

Enter in this box the number of variables which will be used in the events. Variables assigned to this purpose will receive the name EV_(from 0 to the configured quantity).


To appear in Interactive Fields

Quantity: 0

Enter in this box the number of variables which will be displayed in interactive fields. Variables assigned to this purpose will receive the name CI_(from 0 to the configured quantity).


Floor points

Quantity: 0

Contingency 0

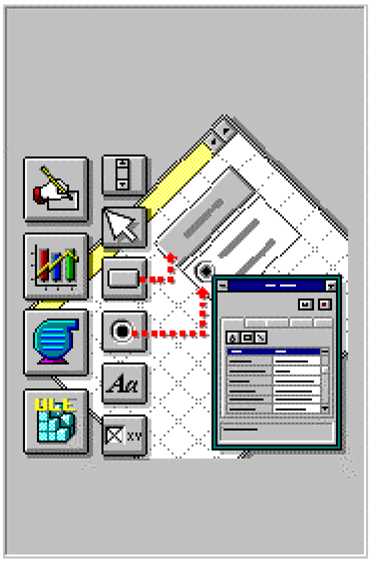
Enter in this box the number of variables which will be used to communicate with the plant floor. In addition you can optionally specify an extension, that is a number of additional variables the system is allowed to configure if necessary. The normal variables of this kind will receive the name PLC_(from 0 to the configured quantity). Instead, the extension variables will receive the name TPLC_(from 0 to the configured quantity).

Once this window configuration is complete, press

To proceed to the next dialog window

To return to the previous window

To cancel the creation of the application

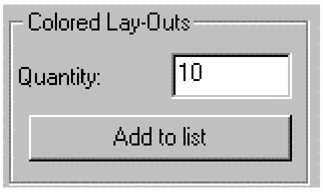


6.1.5 Fifth Configuration : Screen Layouts Configuration

In this configuration window you can define the screen layout structure (as application pictures are called in ***LOGOVIEW NT***).

As mentioned previously, a ***LOGOVIEW NT*** application can be seen as a network graph, in which nodes stand for screen layouts. The Wizard allows to quickly define the application screen layouts by inserting shortcuts. A shortcut consists into a jump to another layout. This way you can build an application without writing any event managing calls among screen layouts.

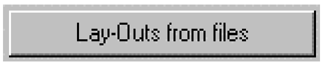
The shortcuts built by the Wizard will find their place in a toolbar appearing in the upper part of the layouts themselves.
In order to apply this technique, you have to define the first screen layout of the sequence. This is the screen layout that ***LOGOVIEW NT*** Runtime (the execution environment) will display just after the application loading.



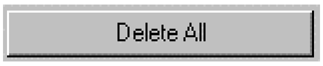
Colored Lay-Outs

Quantity: 10

In this section you can enter the number of the application's opaque screen layouts. To insert, enter first the number in the white box and then press the add button.

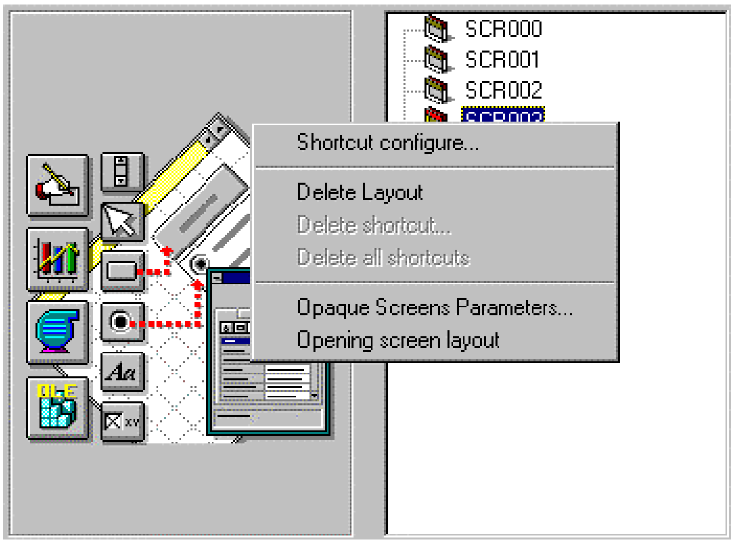


This button allows adding to the application new screen layouts already prepared in files.



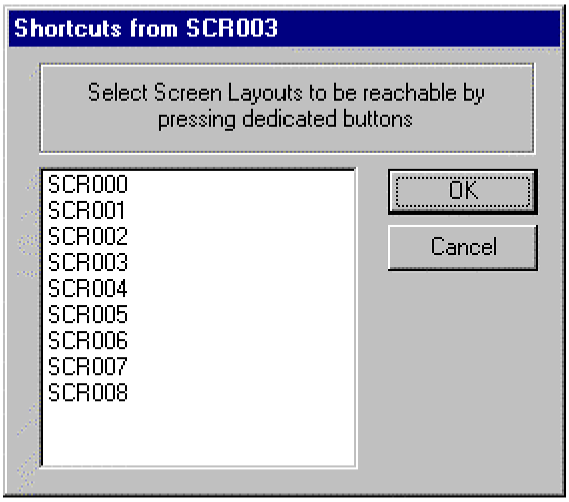
This button deletes all application's screen layouts.

To organize the layouts structure you can proceed as follows. First add the desired screen layouts, using the *opaque* button or *the file resident* one.



Position the mouse cursor on the screen layout in which you want to insert a shortcut and double click.

The appearing Popup will contain the ***Shortcut Configure...*** option. By selecting this option another window will appear where you can select the screen layout to be linked.

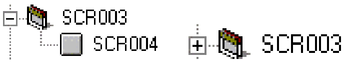

Shortcuts from SCR003

Select Screen Layouts to be reachable by pressing dedicated buttons

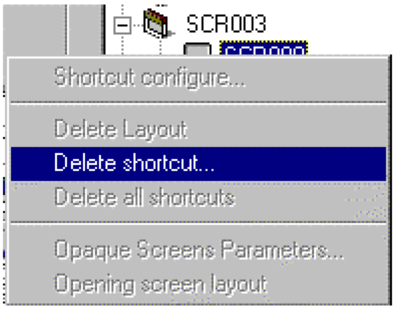
SCR000
SCR001
SCR002
SCR003
SCR004
SCR005
SCR006
SCR007
SCR008

In this window you can carry out a multiple selection. In such case, an equal number of shortcut buttons will be added to the screen. The *OK* button shall confirm the shortcut definition and complete their implementation.

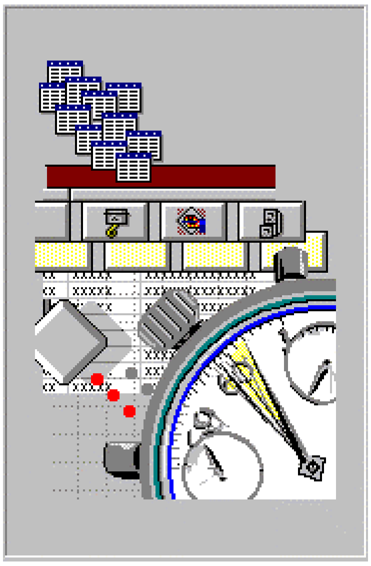
To highlight the screen shortcuts position the mouse cursor on the small icon ⊞. To hide the shortcuts, press ⊟



The shortcut is depicted in the layout list by means of a button icon □

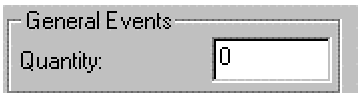


To erase the shortcut just position the mouse cursor on the shortcut icon, double click and select the "***Delete Shortcut*** " entry.



6.1.6 Sixth Configuration : Setup the events structure

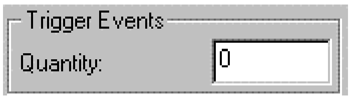
This configuration window allows creating the application events according to the purpose they have to serve.



Quantity: 0

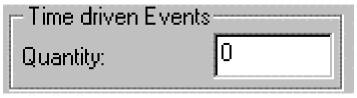
In this box you can setup the generic use events number you have evaluated will be required by the application.

The events defined for this purpose will be given the name GEN(from 0 to the configured number).



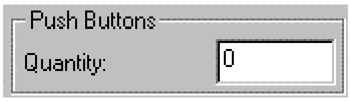
Quantity: 0

In this box you can setup the number of the events associated to trigger variables. These events will be started any time the associated trigger variable takes the high value (one). The events defined for this purpose will be given the name TRG (from 0 to the configured number).



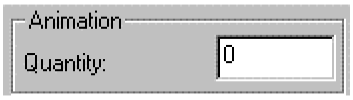
Quantity: 0

In this box you can setup the number of time driven events. These events will be started any time their set time has elapsed. The events defined for this purpose will be given the name TIM (from 0 to the configured number).



Quantity: 0

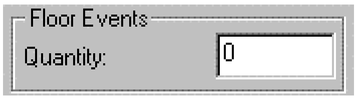
In this box you can setup the number of button started events. The events defined for this purpose will be given the name BUT (from 0 to the configured number).



Animation

Quantity: 0

In this box you can setup the number of animation associated events. The events defined for this purpose will be given the name ANM (from 0 to the configured number).



Floor Events

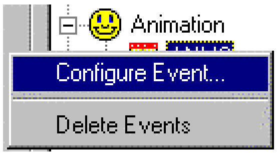
Quantity: 0

In this box you can setup the number of plant floor interaction events. The events defined for this purpose will be given the name PLC (from 0 to the configured number).

By means of this button, following further confirmation, you can delete all configured events.

This button allows to get access to the events library.

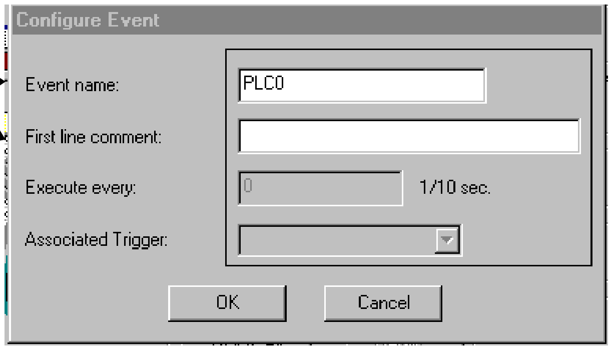
To configure the event characteristics you only have to position the mouse cursor on the icon pointing to it and double click. Then select the ***Configure Event...*** entry.



This action causes the appearance of the event configuration window.

These two fields can be entered for all events

Address of the trigger variable associated to the trigger event



Configure Event

Event name: PLC0

First line comment:

Execute every: 0 1/10 sec.

Associated Trigger:

The window contains four fields. Two of them are applicable to all events (***Event name and First line comment***). One for the events associated to triggers (***Associated Trigger***) and the last one for the Timer associated events (***Execute every***).

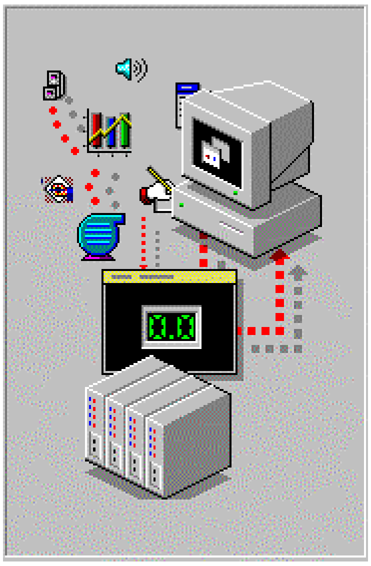
Once this window configuration is complete, press

To proceed to the next dialog window

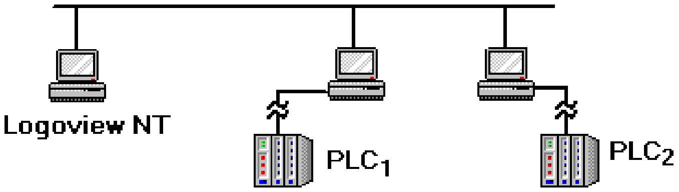
To return to the previous window

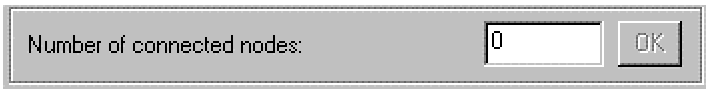
To cancel the creation of the application

6.1.7 Seventh Configuration : Communication with the Plant Floor



In this configuration window you can define the number of stations that will perform communications with the ***LOGOVIEW NT*** application being created. In addition you can map variable blocks present in the remote stations to corresponding variable blocks in the application. ***LOGOVIEW NT*** can perform communications with one or more stations. Each of them resides on a node (this can even be local, when ***LOGOVIEW NT*** and server reside in the same computer).

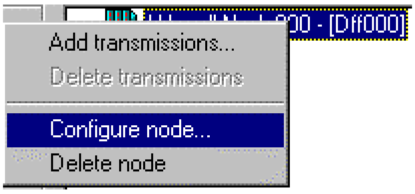




Number of connected nodes: 0 OK

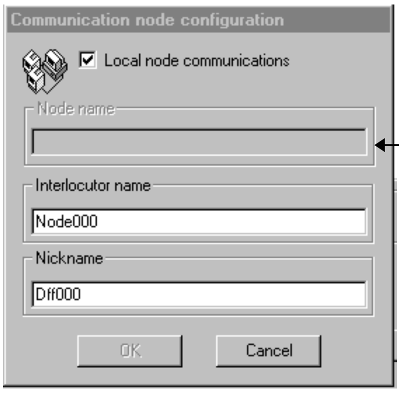
In this box you can setup the number of nodes on which communication stations are residing. Enter the required number and then press the OK button.

This button deletes, following further confirmation, all configured nodes.



To configure a station present on a node, position the mouse cursor on the icon (initially showing a local node) of the node to be configured and double click. Then select ***Configure node...*** in the displayed menu.

This action calls the node configuration window.



☑ Local node communications

Node name:

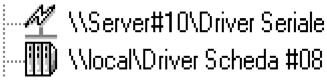
Interlocutor name: Node000

Nickname: Dff000

In case the node is not local, enter into this box the name assigned to the node itself within the network .

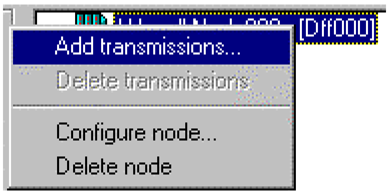
In this window you have to enter the location of the station with which the ***LOGOVIEW NT*** application will have to communicate. The station node can be local, that is reside on the same computer. Or else it can refer to a given address in the installed network.

In the available nodes list, a local node is depicted by an icon different from the remote node one.



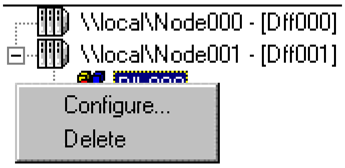
Remote node

Local node

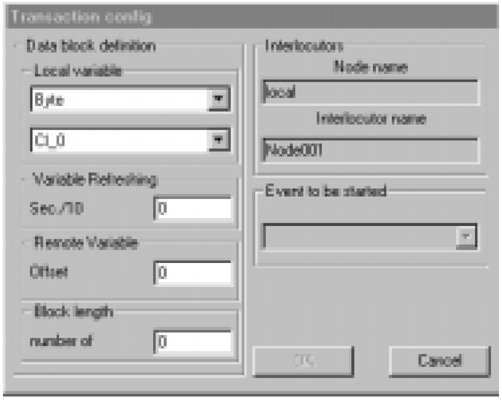


To configure read operations of variable blocks from a given server station, just double click on the corresponding icon and select the ***Add transmissions...*** entry in the displayed menu.
This action will add a sub-tree to the node list, likewise the layouts one, constituting a transmission block to that node. This block is made of a series of variables read from the server station.

To configure transmissions position the mouse cursor on the corresponding icon and double click on it. Then select the entry ***Configure...*** in the menu appearing next.



This action will cause the window for transactions configuration to appear. Each server variable block reading constitutes a transaction.


Transaction config

Data block definition

Local variable: Byte, CI_0

Variable Refreshing — Sec./10: 0

Remote Variable — Offset: 0

Block length — number of: 0

Interlocutors — Node name: local; Interlocutor name: Node001

Event to be started

First variable offset in the Logoview application of the block to be read

Station characteristics from which transactions will occur

Time interval between readings

Event to be started at any change of any variable in the block

First variable offset in the server of the block to be read

Block word count

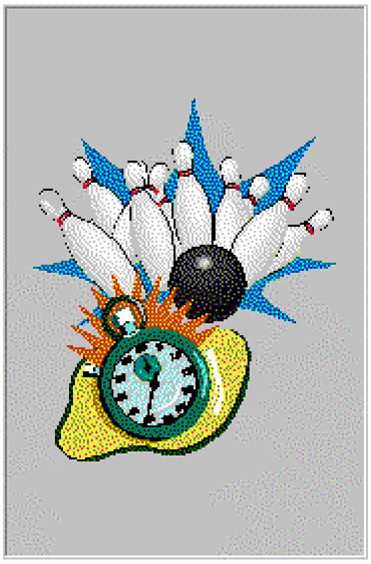
transfers a server variable block into a corre ck in the
T application. Therefore, this function al chronous
application's data blocks from connected se
configuration is complete, press

Next > ***To proceed to the next dialog window***

< Back ***To return to the previous window***

Cancel ***To cancel the creation of the application***

6.1.8 Eighth Configuration : End setup



This configuration window allows to get a summary of all Wizard made configuration choices and to save the created application.

The window allows the usual choices:

To return to the previous window press

To cancel the application creation activity press

The only meaningful difference is given by the button you press to confirm that the application just configured should be saved by the system.

To end the configuration process and save it press

The Wizard, after creating the Application in memory, will display the *files window* to save it on disk.

Once you confirm, the application will be saved in the entered directory and you are now in the position to complete it using the ***LOGOVIEW NT*** development system.

7. Output on Screen

A supervision application developed using ***LOGOVIEW NT*** utilizes the video monitor as its main communication channel with the operators. Through the screen the operator interacts with all significant process parameters. The typical ***LOGOVIEW NT*** application is designed as a sequence of screen layouts. The order within the sequence is decided by the designer. The way a ***LOGOVIEW NT*** application works is therefore strictly connected to the screen layouts, which schematically represent the plant and all its components. Trough communication with the external world ***LOGOVIEW NT*** acquires all the data which allow it to display on screen the plant processes in a graphic manner, using the screen layouts as background. The way the screen layouts are designed is of great importance for optimizing ***LOGOVIEW NT*** functionality's.

7.1 Fonts

LOGOVIEW NT can make use of all fonts included in your *WINDOWS* operating system. The font's usage in ***LOGOVIEW NT*** is not straight forward; it is necessary to define new fonts inside ***LOGOVIEW NT*** (which will be called ***LOGOVIEW NT*** fonts, to tell them apart from the operating system fonts, called *WINDOWS fonts*), name them and then use them in the statements within the Events.

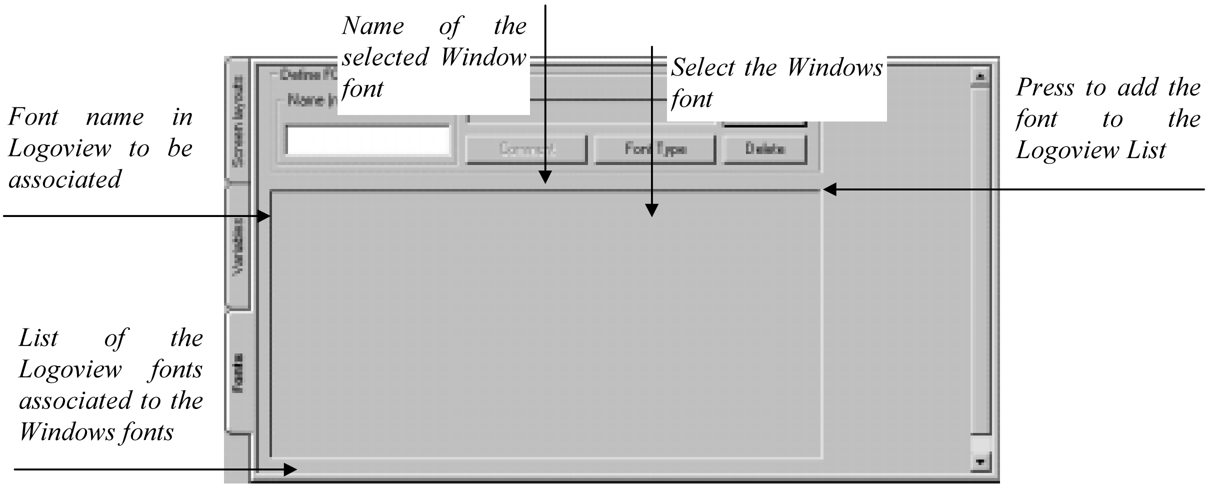
This trick has been devised to maintain compatibility with previous versions of ***LOGOVIEW***. A ***LOGOVIEW NT*** font is defined by a fixed set of characteristics, such as size, shading, italic, etc. as it is in *WINDOWS*.

Once defined, each ***LOGOVIEW NT*** font will be addressed by its name in all statements using it.

If, for example, you want to use in your application the “Courier” WINDOWS font, size 10, all you need is to create a LOGOVIEW NT font with the same characteristics, named for instance “my_Cour”. This is the font you need to recall in your statements.

7.1.1 *LOGOVIEW NT* font definition

As said above, ***LOGOVIEW NT*** makes use of fonts of its own, derived from the operating system fonts. To define one or more fonts, just call ***Fonts*** from ***Basic Setup.***


Name of the selected Window font
Select the Windows font
Press to add the font to the Logoview List
Font name in Logoview to be associated
List of the Logoview fonts associated to the Windows fonts
Define FO
Name i
Connect
Font Type
Delete
Screen layouts
Variables
Fonts

This window allows the user to define all the associations between *WINDOWS* fonts and ***LOGOVIEW NT*** fonts. After creating the association, every time you need to use the font, just refer to it by its defined name.

Example: if the font “my_Cour” has been defined as explained, to use a “Courier” WINDOWS font, size 10, then the statement to make use of it will be: FONT “MY_COUR”

7.2 Screen layouts skeletons

Screen layouts skeletons are background graphic objects used by ***LOGOVIEW NT*** to build screen layouts. The screen layouts are therefore background pictures plus other objects providing operator interaction. It is the task of the developer to load into the application all required backgrounds, then insert all other objects.

7.2.1 Screen background types

Many types of screen backgrounds exist, in order to allow optimal solutions to various types of problems. Formally, the various types are different. The available screen backgrounds are of the following types: ***Bitmap***, ***Dialog***, ***Transparent***, ***Vectorial*** and ***Opaque***.

7.2.1.1 *Bitmap type*

LOGOVIEW NT has been conceived to manage bitmap files in an optimal way. A bitmap picture is composed by a block of numbers describing the screen picture.

To precisely understand what a bitmap is, you can imagine having to lay down a mosaic. The number of colors depends on the color of the tiles you have at your disposal. If you have 16 types of tiles, you make a 16 colors mosaic, if you have 256 types of tiles you make a 256 colors mosaic and so on. The picture detail level you can achieve, instead, depends from the tile sizes. The smaller the used tiles, the more detailed the achievable picture features. Using this analogy, we can say that the mosaic represents our bitmap, where the tiles stand for the small points of the picture, technically called pixels. The bitmap pictures can be built using various systems. The easiest is to apply our editor Flash Draw, which is a tool particularly suited to lay down background picture for your screen layouts. Since there is not a single standard graphic format, each drawing program makes use of its own format. ***LOGOVIEW NT*** has the capability of reading the formats of the most used programs, also because now the choice has shrunk to 4 or 5 types, the most efficient ones.

To read the most exotic formats, and anyway those not supported, you should contact your dealer checking if a program is available on the market able to convert the file generated by your editor into one readable by ***LOGOVIEW NT***. The supported formats are the following:

- **SCR (Gost)**: This is a format generated by GOST-PAINT (G-PAINT), which has been kept for compatibility reasons with precious old *Logoview* versions. It is extremely fast at loading, while picture compression is not so effective. It does not allow loading pictures with a number of colors different from the one actually defined in the selected graphic mode. Instead, there are no problems if graphic resolutions are different (colors being the same). It does not allow saving 24 bit pictures.

- **BMP (*WINDOWS* Bitmap)**: This is the format generated by Paintbrush in the *WINDOWS* environment. This format is one of the most ineffective as far as data compression is concerned, however, it consists of a “must” for people determined to use the *WINDOWS* internal editor to create bitmaps.

- **GIF (Compuserve)**: This is a format allowing remarkable disk saving, using a LZW compression type, achieving an optimal picture compression without detail loosing. On the other side, given the high workload to decompress picture, it is rather slow at loading. Neither this format allows managing 24 bit pictures.

- **PCX (Paintbrush)**: This is one of the oldest file formats, dating back to the time of monochromatic pictures. The compression is similar to the one of the SCR files, but is capable of handling 24 bit pictures. This is one of the paintbrush generated formats.

- **CUT (Dr. Halo)**: This is another classic (old) format. It is maintained and used since it is generated by one of the most popular existing graphic editors: Doctor Halo. In this case too, as in the PCX one, the file compression is rather low. It provides a peculiar feature, however: the color file is separately generated with extension PAL. This requires to move or copy two files instead of one when you want to transfer a picture. Also in this case, as far as we know, 24 bit files can not be handled. We do not recommend using it because it seems that the newest package versions handle now other formats certainly more effective and better compressed.

- **TIF (TIFF)**: This is the presently most preferred format. It can be generated practically by all medium and high level graphic editors. As in the GIF case, it makes use of an LZW compression type, but, unlike the other, allows handling 24 bit pictures. Consequently also this format is slow both in reading and in writing

- **TGA (Targa)**: This is a format used exclusively for 24 bit pictures. It is generated by editors used mainly in the broadcast field. Pictures are practically not compressed, therefore they take a lot of disk space. Should you absolutely need to make use of such pictures, we suggest to convert them into TIF or PCX format before actually using them in the application.

7.2.1.2 *Dialog type*

The background files of dialog type allow to build screen layouts containing all main objects usually present in a *WINDOWS* dialog window, such as sliding bars, options, buttons and so on. These screens turn our to be quite useful to build masks dedicated to data entry by the operator.

7.2.1.3 *Transparent type*

The transparent type do not erase the background screen. These allow therefore a given screen layout to possess various different configurations according to varying application needs. The bitmap type screens make use of a graphic technique which does not describe objects or picture elements, but rather the whole picture as a single entity. This is the reason why all graphic objects needed to animate screen layouts are superimposed to the background screen, as if they were laid down on a transparent plane parallel to the background screen. This operational capability helps the developer needing to have several screen layouts with the same background but with different configurations. This feature allows to save memory, by sharing background. They constitute, in practice, items not associated to any specific screen layout which can be loaded on a screen already present in memory. Strictly speaking, you can understand that this is not a true screen layout, however it has been included in this paragraph since it can be used anyway as if it were a screen skeleton like the others.

7.2.1.4 *Vectorial type*

By vectorial type we mean any file type generated by vectorial graphics programs, also called object oriented graphics. Pictures are in this case saved in the form of statement lists describing the way the picture can be built starting from elementary geometric elements. These statements are always in the form of numbers inside the computer. The list of statements therefore constitutes the mathematical description of the picture. The drawing is conceived as a collection of lines, circles, rectangles and so on. Each of these items is called *object.*

Any individual object in the picture has its own set of primitives. Each object, therefore, can be enlarged, reduced, copied, moved and changed without any influence on the other picture objects. In addition, when proceeding to the object reduction, there is no resolution loss and the lines are smoother then the ones drawn with bitmap technique. The vectorial formats that ***LOGOVIEW NT*** has the capability to read are two:

- **SLD (Autocad Slide)**: this type of file is generated by the *AUTOCAD* package. It does not consist of its native file (the one normally applied by the package to save drawings). This is instead a file fixing a picture frame. In practice it consists of a part of the whole drawing. This part, which if required could comprise the whole drawing, can be selected using suitable controls available in the file generating phase. Our choice of reading this file type instead of the standard DXF has been recommended by the observation that, while the SLD format has never been modified in the 12 *AUTOCAD* versions, no version came out without changes to the DXF file. This would have forced us to modify also ***LOGOVIEW NT*** accordingly. To get detailed instructions on how to generate SLD (slide) type files, please check with the documentation supplied with the *AUTOCAD* package.

- **WMF (*WINDOWS* Metafile)**: This is the *WINDOWS* standard vectorial format. This can be generated by almost all vectorial graphic packages running in this environment, even the ones normally applying their own proprietary format. By means of this format *CORELDRAW* or *Mircrografx DRAW* generated drawings can readily be imported. Also in these cases, to get further details check with the documentation supplied with your drawing package.

7.2.1.5 *Colored Opaque type*

Within ***LOGOVIEW NT,*** by opaque type we refer to totally empty screen layout background. Any time a bitmap or vectorial file is being used, ***LOGOVIEW NT*** loads and stores a predefined picture, reading it from disk. On the contrary, when an opaque screen layout is being used ***LOGOVIEW NT*** does not read any disk file, instead creates directly the bitmap in memory, assigning it a suitable size and color. These and other screen attributes are generated on the basis of configuration procedures (we shall see more details later). Then you can begin working on such a uniform background as if it were a normal screen layout.

7.2.2 Laying down the screen layout background

Laying down the background picture of the screen layouts can be achieved using any graphic editor, as long as it allows saving files in one of the ***LOGOVIEW NT*** supported formats. Anyway it is advisable use *Flash Draw*. This program, separately purchasable, has been created just to implement the graphic parts of ***LOGOVIEW NT*** applications.

7.2.3 Using screen layout background

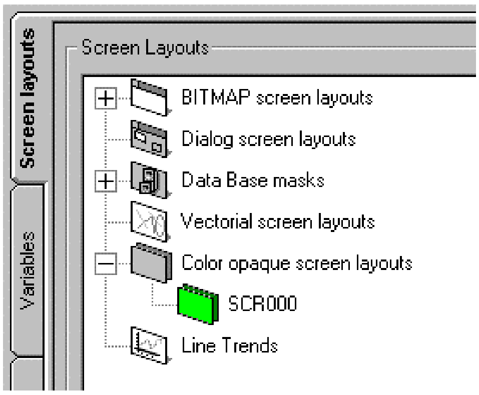
LOGOVIEW NT is capable of supporting all file types mentioned above. In addition ***LOGOVIEW NT*** allows to get access to all *WINDOWS* graphic driver resolutions. Particularly, it can handle pictures having any number of colors, starting from the 16 standard VGA colors till to the 16 million colors of boards such as TARGA/ILLUMINATOR.
The problem of converting pictures with different palettes has been solved. In fact you can handle pictures even if the graphic board color handling capability is lower then the picture number of colors. Thanks to a sophisticated algorithm based on the Floyd-Steinberg theory, ***LOGOVIEW NT*** performs an image dithering allowing it to be displayed also in low capacity graphic systems.
Regrettably, this brings about two drawbacks. First some additional loading time is required by ***LOGOVIEW NT*** in such cases, due to the related conversion time, obviously proportional to the picture size. The second drawback is more severe. After dithering the picture colors are no longer simple. They are composed of a fine pixel grid mixing various colors. This, on the other hand has side effects on certain graphic operations, hindering their results.

In summary, in order to build a screen layout you have first to draw a background screen, load it into ***LOGOVIEW NT***, then insert in it all objects (trends, interactive objects ...) useful to display the whole plant floor status and interact with the Operator. The whole of these objects is called screen layout. We can state, therefore, that many types of screen layouts exist, depending on the task assigned to each of them. These tasks belong to one of three following general classes: supervision, menus and dialogs. The screen layouts dedicated to supervision constitute the work base in any practical application, while the other two types can be considered as accessories.

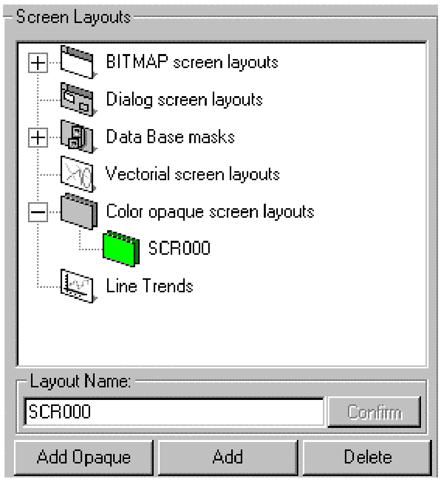
To position these items on the screen, ***LOGOVIEW NT*** makes available an editor. The screen layout editor allows to place on screen all operator interfacing objects. The screen layouts constitute therefore the foundation of any application.

7.3 The Screen Layouts

After defining the Application’s background screens (which could also be thought as empty screen layouts) you ought to configure them. In order to do so you have to open the ***screen layout editor,*** double clicking on the name of the application screen layout appearing in the ***Screen Layouts*** section of the ***Basic setup*** window.



Screen Layouts
- BITMAP screen layouts
- Dialog screen layouts
- Data Base masks
- Vectorial screen layouts
- Color opaque screen layouts
  - SCR000
- Line Trends



Screen Layouts
- BITMAP screen layouts
- Dialog screen layouts
- Data Base masks
- Vectorial screen layouts
- Color opaque screen layouts
  - SCR000
- Line Trends

Layout Name: SCR000

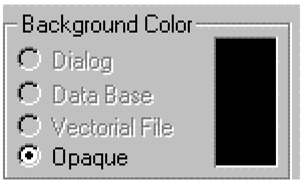
7.3.1 Adding and deleting a screen layout

The addition of a screen layout can be done in the *screen layout* section of the *Basic setup* window.

Double click

An empty screen is created by pressing the button

Next you can enter the screen layout type you desire, selecting it from the available types.



Background Color
- ○ Dialog
- ○ Data Base
- ○ Vectorial File
- ● Opaque

Other screens layouts can be loaded from files by means of the button

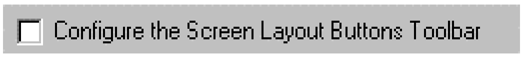
LOGOVIEW NT identifies the screen layout file you are attempting to load from the file extension and, consequently, adds it to the corresponding category.

To erase a screen layout you simply select it from the list and press the button

This delete operation does not erase the file from disk, but simply makes it no longer present in the Application.

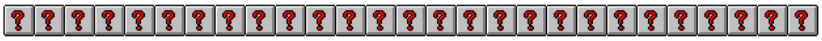
7.3.2 Hierarchy building

There are mainly two ways to link screen layouts together. The first, the classic one, is achieved by placing in the screen some buttons directly calling other screens. Sometimes however placing buttons on screen can result to be inconvenient, since the screen layout could already be filled with objects. To solve this problem ***LOGOVIEW NT*** allows you to place buttons in a toolbar outside the screen layout itself, in the upper part of the window containing it. To create such a toolbar on a screen layout, select it from the screen layout list and then activate the

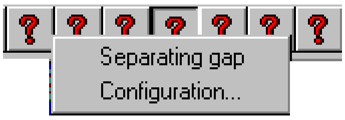


□ Configure the Screen Layout Buttons Toolbar

Upon activating this option a non configured toolbar will appear in the upper window part once you open the screen layout editor.

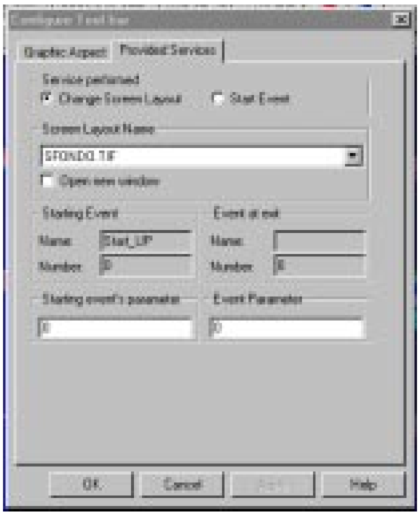


This toolbar's buttons can be individually configured by means of a click callable menu.



The configuration window allows to associate either the loading of a screen layout or the start of an event to each individual button. This allows to quickly build the screen layout sequence making up the ***LOGOVIEW NT*** application.

The button configuration phase involves two sections. One to configure the button's aspect (icon, name and so on). The other to select the associated logic action, change screen layout or start event..



Configure Tool bar

Graphic Aspect | Provided Services

Service performed
● Change Screen Layout ○ Start Event

Screen Layout Name
SFONDO.TIF
□ Open new window

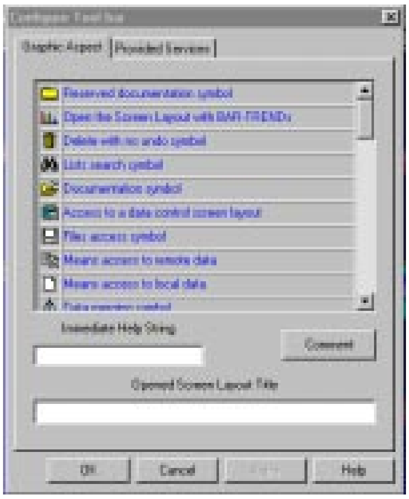
Starting Event
Name: Start_UP
Number: 0

Event at exit
Name:
Number: 0

Starting event's parameter
0

Event Parameter
0

OK | Cancel | Apply | Help



Configure Tool bar

Graphic Aspect | Provided Services

- Reserved documentation symbol
- Open the Screen Layout with BAR-TRENDs
- Delete with no undo symbol
- Lists search symbol
- Documentation symbol
- Access to a data control screen layout
- Files access symbol
- Means access to remote data
- Means access to local data
- Data insertion symbol

Immediate Help String

Comment

Opened Screen Layout Title

OK | Cancel | Apply | Help

Graphic Aspect

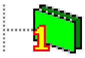
Provided Services

7.3.3 First screen layout of sequence

The sequence first screen layout is the one loaded at the application's starting (at runtime). To select the first screen layout select it from the list and then activate the

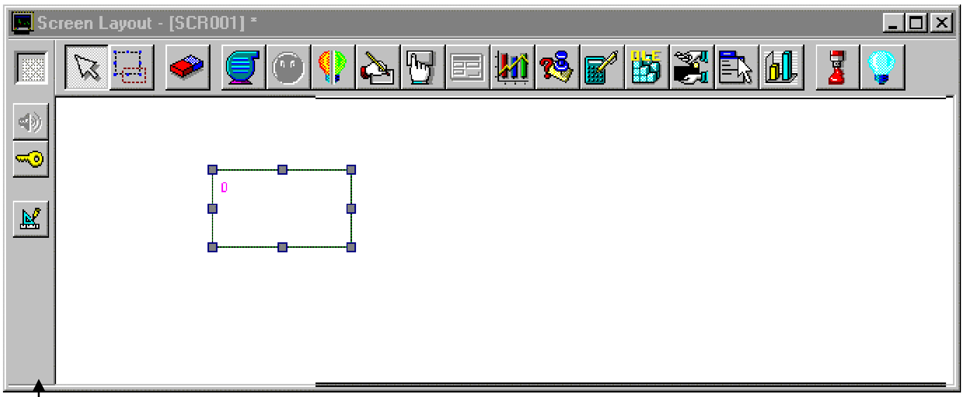
☑ first screen layout of programmed sequence

The sequence first screen layout is shown in the list by a dedicated icon, as shown below.



7.3.4 The screen layout editor

This editor allows to place on the screen the *operator interaction* objects. To open the editor on a specific screen layout you only have to double click on the corresponding icon. You find it in the screen layout list in the Screen layout section of the basic setup window.



Screen Layout - [SCR001] *

7.3.4.1 *Inserting a reference box*

Emergency Gem. Can be used as a sentinel. To configure, click on it.

utton allows inserti box. Any screen object m d into a ce box defining its step to place an object is t fine the olding it. Once the box has been placed on screen, can be resized and moved ling to specific needs.

7.3.4.2 *Deleting objects*

This button allows deleting all objects selected in the screen layout.

7.3.4.3 *Moving and resizing*

This button changes the cursor into its classic arrow form allowing to select, move or resize (by pulling the handles) one or multiple objects present in the worksheet. To select multiple objects just keep the SHIFT key pressed while selecting.

7.3.4.4 *Vectorial objects*

This button is active only if at least one box is selected. It allows inserting a vectorial object previously defined in a library. Vectorial objects are graphic objects, linked to variables, extremely useful in ***LOGOVIEW NT*** applications. Typical examples of vectorial objects are symbols depicting plant machine tools, motors, valves and so on. A vectorial objects library can be created by means of the program *Flash Draw* (separately purchasable).

7.3.4.5 *Sprite animation*

This button is active only if at least one box is selected. It allows inserting an *animation* into a box. An animation is composed of a sequence of fields of view. Such a sequence could be used, for instance, to display in a perceptive form a production cycle. An animation of this type can be built using the program *Flash Draw* (separately purchasable).

7.3.4.6 *Color animation*

This button is active only if at least one box is selected. It allows creating a *color animation* into a box. A color animation is defined as object color changing following value changes of an associated variable. A common example of such animation type is to represent a warning flag. Each flag color encodes a running state of the associated piece of machinery.

7.3.4.7 *Interactive fields*

This button is active only if at least one box is selected. It allows defining an interactive field inside a box. This type of field allows the operator to control a variable. The interactive field associated variable can be displayed in different colors, depending on its values. In addition, if the developer purposely configured the field accordingly, the operator will be able to change the variable value at the application runtime.

7.3.4.8 *Buttons*

This button is active only if at least one box is selected. It allows defining a button inside a box.. An event start or a screen layout loading can be associated to the created button.

7.3.4.9 *Controls*

This button is active only if the screen layout is of dialog type and at least one box is selected. It allows defining a dialog window typical control inside a box.. These controls can be placed inside the box in such a manner as to build an input/output user mask.

7.3.4.10 *Trend*

This button is active only if at least one box is selected. It allows inserting a trend into the box. A trend is the display in some graphic form of a variable. The available forms are the following three: Bar trends, Analog meter and Digital meter.

7.3.4.11 *Context Help*

This button is active only if at least one box is selected. It allows inserting context help into the box. This help will be at the operator disposal to get information on the way the current screen layout can be used, assuming the developer provided that information.

7.3.4.12 *Tables*

This button is active only if at least one box is selected. It allows inserting a variable table into a box. Such tables turn out to be quite convenient if you want variables to appear on screen in the form of lines and columns.

7.3.4.13 *OLE objects*

This button is active only if at least one box is selected. It allows inserting an OLE object into the box. This feature turns out to be quite convenient if you want to use in the screen layout external applications such as *WORD* or *EXCEL*. In fact, these applications can be imported following the *OLE* standard into other applications. This allows the developer to transparently insert a foreign application into a ***LOGOVIEW NT*** screen layout

7.3.4.14 *Multimedia Objects*

This button is active only if at least one box is selected. It allows inserting any object with MCI interface into the box. ***LOGOVIEW NT*** has consequently the capability of playing back sounds, video clips and similar items. For instance, using a video acquisition board ***LOGOVIEW NT*** can put on screen picture taken by a video camera.

7.3.4.15 *Popup menus*

This button is active only if at least one box is selected. It allows inserting a Popup menu into the box. This will in turn be activated by pressing the right mouse key on the selected box itself. This menu will contain configurable, application convenient control entries.

7.3.4.16 *Graphs*

This button is active only if at least one box is selected. It allows inserting a graph into the box. Graphs can be built by means of a powerful editor allowing to configure in a remarkably flexible way all variables and parameters contained in them.

7.3.4.17 *Local blink table*

This button allows defining the screen layout specific blinking colors. This feature provides individual screen layouts with special blinking colors on top of the default ones.

7.3.4.18 *Local color table*

This button allows defining the screen layout specific colors on top of the default ones.

7.3.5 Interactive fields

Since, as already extensively remarked, purpose of monitoring applications is to acquire data and present them in front of an operator, paramount importance has the feature of displaying numeric data. We mean data regarding plant floor quantities, not just plant states. Tightly linked is the capability for the operator to enter and modify such values. In other frequent cases the need might occur of displaying information in various colors and fonts to provide more clarity and of checking input data to prevent the operator from entering wrong data. Tools providing such functions are called interactive field in ***LOGOVIEW NT.*** They carry a number of peculiar characteristics.

- ***Immediate configuration.*** In fact they can be easily configured without the need of additional code, just associating them to variables.

- ***Automatic refreshing.*** Immediate refreshing of variables on screen.

- ***Blinking.*** Configured relationships between display colors and values of floor quantities, possibly making use of blinking (alternating) colors for out of range values.

7.3.6 Trends

Trends are certainly among the most applied ways of displaying complex data. Trends provide effective display of consistent quantities in the same scale and frame. There are three trend types you can put on a screen layout.

They consist into bar charts in which any bar length shows a value in a given scale. In this trend type thresholds and colors can be configured. The main purpose for such trend type is to show real time quantities, since their refreshing is immediate.

This trend type provides sound alternatives to interactive fields, the main difference being that they do not allow change of data by the operator. This trend depicts a digital meter with numbers appearing as in the LCD segment technique. Consequently it can show just one quantity on a given meter. Also this trend type if fully configurable, namely to choose thresholds and colors. This trend type is particularly suitable for real time data display.

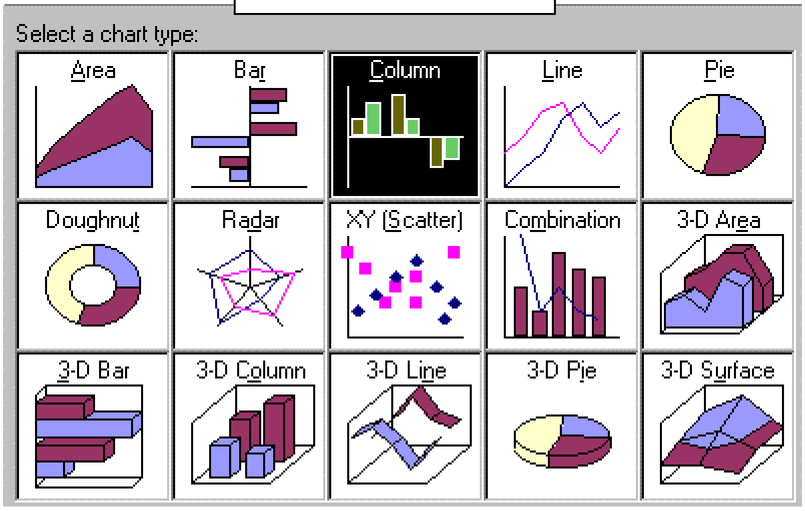
This trend provides the analog version of the previous type, but can display up to three different variables. This tool is particularly suitable to bring on screen traditional meter dials, imitating the way quantities are normally displayed by real meters on board floor equipment. Also this trend is normally used to display quantities in real time.

In addition a separate application is available to provide line trends, that is graphs displaying data series as lines joining points. This application allows using graphs in a simple and flexible way without any display area size limits.

7.3.7 Graphs

In addition to the trends ***LOGOVIEW NT*** makes available a powerful tool for graphs creation, both t dimensional.

For each chart type you can setup the axis orientation, scale,colors,.... according to your needs.

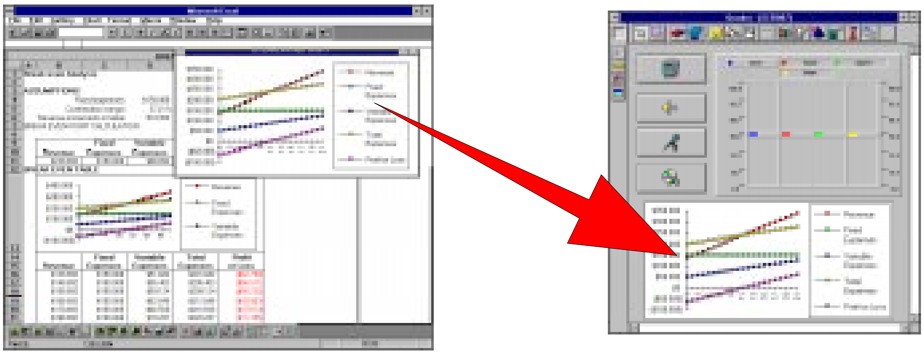


These graphs are not suitable for real time display of data, since their refreshing is rather slow. Refreshing these graphs is achievable by dedicated statements you can code in events.

7.3.8 OLE objects

Besides the official ***LOGOVIEW NT*** objects, you can put into screen layouts certain generic objects, controlled by other *WINDOWS* applications, comprehensively called OLE 2.0 servers. OLE 2.0 is one of the protocols defined by Microsoft for data interfacing and data exchanging among miscellaneous applications. One of the most remarkable features is to include in a document a call to another document totally different in nature from the first.

This function allows the user to run a program calling it from inside another program in a client / server mode. ***LOGOVIEW NT*** takes full advantage of this feature allowing to include into screen layouts documents compliant with the OLE 2.0 specifications, in the so called *embedded* mode. This way you can complete and integrate ***LOGOVIEW NT*** applications with spreadsheets, graphs, Word™ documents and so on.



Such applications, not included in the package, can be separately purchased or purposely programmed by the User to serve very specific needs, using any C++ compiler. OLE 2.0 objects can safely share resources with standard ***LOGOVIEW NT*** objects.

7.4 Colours

A rather meaningful role is taken in any ***LOGOVIEW NT*** application by color management. Purpose of this paragraph is to introduce the basic ideas at the basis of color management under ***LOGOVIEW NT***.

7.4.1 The palette

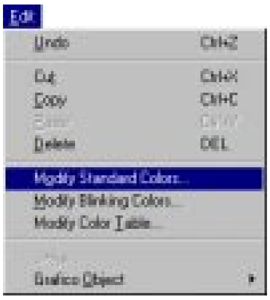
Any screen layout, regardless its type, has a palette. The palette consists of the color set available in the given screen layout.

Any screen layout has its own palette, which can be different from each other. You can draw objects in a screen layout using only its own palette colors. Each palette is made of 256 colors. You must be fully aware that a given color in a screen layout might be not available in another one.

7.4.2 The color table

LOGOVIEW NT makes available various tables to configure all color attributes required by applications. These tables will be used by events by means of suitable statements. As said, colors available in a screen layout are linked to their respective palette. A color configured in a screen layout might be not available in another one. Hence, table colors might show differences among screen layouts. ***LOGOVIEW NT*** performs always a “best-match” between the color configured in a table and those available in the screen layout’s palette. According to the best match criteria ***LOGOVIEW NT*** will choose in the screen layout’s palette the color best approaching the color configured in the table. This feature applies to all color configurations in ***LOGOVIEW NT***.

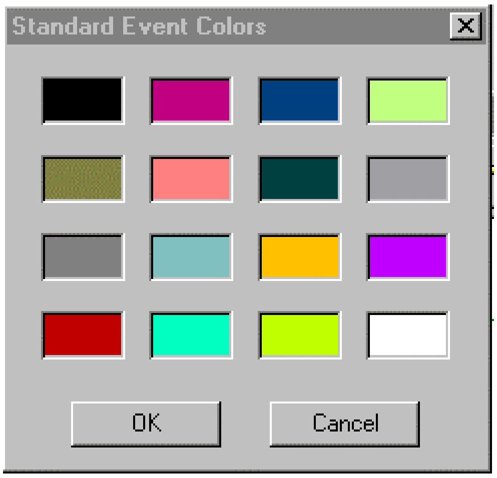
7.4.2.1 *Standard colors*



The standard color table is a table containing 16 colors available in events by means of graphic statements. The table starts with the 0 color and ends with the 15.

To define the standard color table (which is common to all application’s screen layouts) you have to open the ***Screen editor***

and select the entry ***Modify Standard Colors...*** from the ***Edit*** menu. This action will put on screen the ***Standard Event Colors***



In this window you can configure 16 colors which will be available in all drawing statements. The number / color correspondence is shown in the picture.

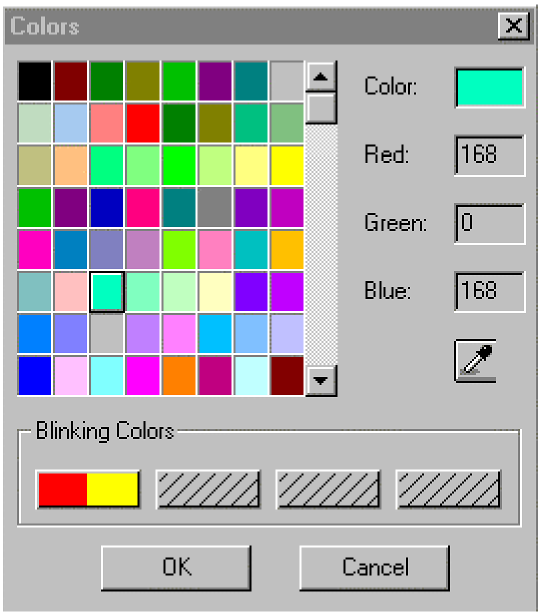
It is also possible to get access to the 16 standard WINDOWS colors. These colors are mapped by numbers from 16 to 31 respectively.

To refer to a given color in this table you only need to specify its number as parameter of the graphic statements.

The BOX statement draws a certain color rectangle in the presently loaded screen layout. Such statement accepts five parameters. The first four ones provide the rectangle coordinates, while the fifth one the color of the rectangle itself.

BOX x1,y1,x2,y2,color

A way for expressing the color parameter is to indicate the color of the table.



Color:

Red: 168

Green: 0

Blue: 168

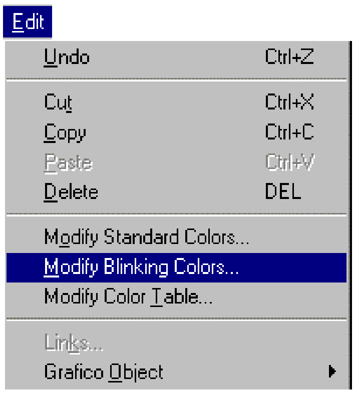
To change a color, position the mouse cursor on the table box corresponding to it and press the left mouse key. The ***Colors*** window will then appear, where you can pickup the color you want.

Colors configured in the ***Standard Event Colors*** window are available in all application's screen layouts, simply indicating the corresponding number in any graphic statement.
As previously seen, not always happens that all screen layouts have the same palette. A color available in a screen layout could be not available on another one. The ***LOGOVIEW NT*** applied method is the best match one explained in the previous chapter. The defined colors might therefore appear not the same in all screen layouts.

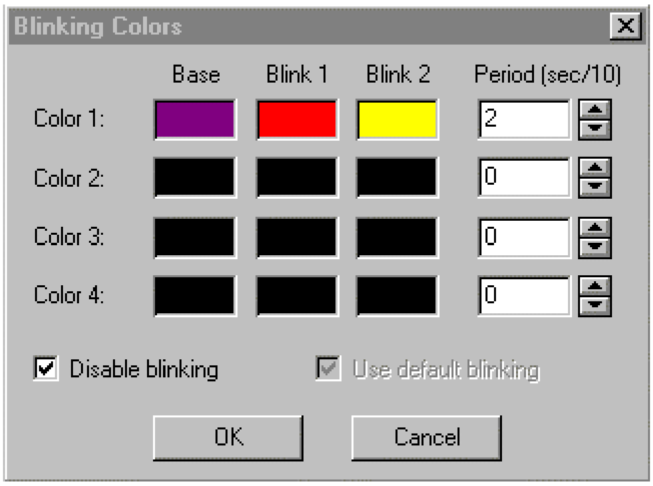
7.4.2.2 *Blinking colors*

LOGOVIEW NT allows handling some particular color types: blinking colors. They consist of colors not fixed on screen but rather alternating between two colors this way providing a blinking effect.
The blinking colors are setup by defining a basic color (or the reference color) which will have the blinking effect and two other colors which will be the two alternating colors of the blinking effect.
For example, if you define green as basic color and red and yellow as blinking colors, any time the green color is put on screen, the alternating display of red and yellow will appear instead of green. As should appear clear, the basis color is just a reference color, in place of which the two other colors will be displayed in alternate sequence of o given time interval. Usually the blinking basic colors are chosen among the unusual ones, not used for other purposes throughout the application.
LOGOVIEW NT provides four blinking colors. Any individual screen layout can have its own local blinking colors. Anyway default blinking colors can be defined once and for all. The default colors can be used by all screen layouts.
In each screen layout you can select if the default colors or the local ones are being used. The blinking local colors are available only for that particular screen layout, consequently they have to be setup individually for each screen layout. Although this will seldom occur, any screen layout could have four different blinking colors. Usually, however, the blinking colors are the standard ones, that is the default blinking colors are mostly used in the whole application.
At creation all screen layouts make use of the default colors by default. It is the developer task to call on screen the applicable screen layouts and change them, making use of the procedures explained in the following paragraphs.

7.4.2.2.1 Default blinking colors



To define the blinking colors you have to open the ***Screen Layouts Editor*** and select the entry ***Modify Blinking Colors...*** in the ***Edit*** menu. This action calls on screen the ***Blinking Colors*** window.


Blinking Colors

	Base	Blink 1	Blink 2	Period (sec/10)
Color 1:	🟪	🟥	🟨	2
Color 2:	⬛	⬛	⬛	0
Color 3:	⬛	⬛	⬛	0
Color 4:	⬛	⬛	⬛	0

☑ Disable blinking ☑ Use default blinking

where you can define the four default blinking colors.

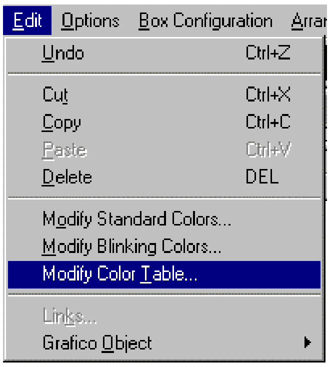
7.4.2.2.2 Local blinking colors

As already mentioned, in addition to the default colors, you can define screen layout specific blinking colors. These can be selected by means of the button shown aside, available in the ***Screen Layouts Editor*** toolbar. The configuration process is achieved again by means of the ***Blinking Colors*** window.

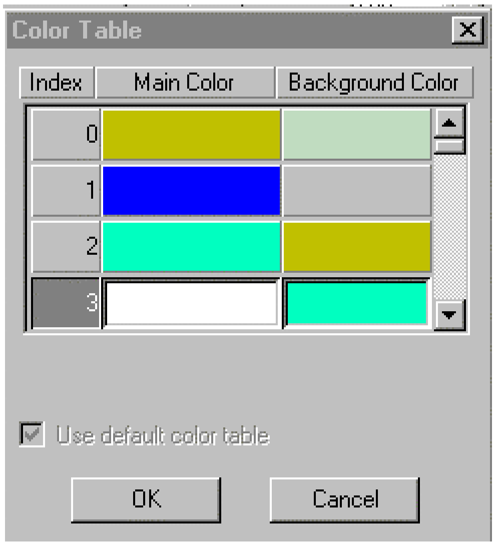
7.4.2.3 *Color table*

The last table made available by ***LOGOVIEW NT*** is the Color table. This table allows defining up to 256 individual colors for use in interactive fields and color animation.
Also in this case two tables exist. The first is the default one, used by all screen layouts, the second is the local one, used only by the associated screen layout and configured to meet its peculiar functions. A screen layout can therefore make access and use both the default and its own colors. The initial screen layout default setup at creation is to use the default colors. It is the developer task to load the screen layouts of interest and change setup according to the procedures explained below.

7.4.2.3.1 Default color table



To define the default color table you have to open the ***Screen Layouts Editor*** and select the entry ***Modify Color Table...*** in the ***Edit*** menu. This action calls on screen the ***Color Table*** window.



☑ Use default color table

where you can configure all 256 available colors.

7.4.2.3.2 Local Color Table

As already mentioned, in addition to the default colors, you can define screen layout specific color tables. These can be selected by means of the button shown aside, available in the ***Screen Layouts Editor*** toolbar. The configuration process is achieved again by means of the ***Color Table*** window.

7.5 Alarms

LOGOVIEW NT provides a quite flexible alarms management function. This is performed in two separate environments. The first in the configuration phase using the development package, explained in the following chapter, the second in the application running phase, using the ***LOGOVIEW NT*** runtime package. The configuration phase allows to setup all alarms parameters. The runtime phase allows to monitor all plant alarms and acknowledge them.

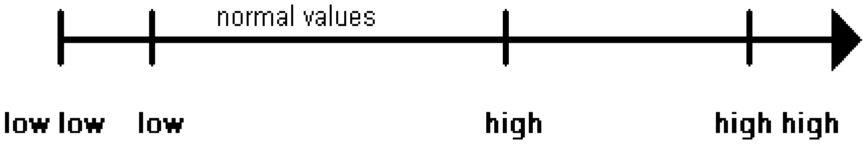
7.5.1 Classes and Zones

An alarm signal should indicate some malfunction. The ***LOGOVIEW NT*** Developer should precisely know and define all quantities the monitoring of which is considered critical by the Process Engineer. These quantities will be stored in variables. It is therefore the task of the Developer to associate alarms to variables.

The alarms can be seen in general and grouped under two main perspectives: classes and zones. An alarm class includes all alarms pertaining to the same category. For example, it might make sense to state that all temperature alarms pertain to the same class, although the final word will be said case by case by the Process Engineer. The second perspective groups alarms according to plant zones, that is physical plant parts. For instance, all alarms of a given cabinet could be said to belong to the same zone.

7.5.2 Analog and Digital Alarms

After associating some control quantities to corresponding variables, we can associate alarm conditions to out of threshold values of such variables. Alarms can be of two types: Analog and Digital. As their names suggest, the first ones are controlled by thresholds, being applicable to analog variables (BYTE, WORD, DOUBLE WORD, REAL), the second ones are generated by binary values (0 and 1), being applicable to digital variables (TRIGGER). Thresholds define value intervals inside which individual variables can range without generating alarm. There are four thresholds, in order to better specify the allowable variable ranges. These four threshold values are called (LOW LOW), (LOW), (HIGH) and (HIGH HIGH). Their representation on an increasing scale can be depicted as follows.



A variable acceptable value is by definition lower then high and greater then low thresholds. The other two threshold values can be used for finer assessments, such as flagging plant safety conditions. As we shall see, you are not forced to use all four values. These four alarm thresholds are each controlled by four consecutive trigger variables. They take the 1 value any time the corresponding threshold is crossed and 0 otherwise. Hence each analog alarm is mapped to four digital alarms.

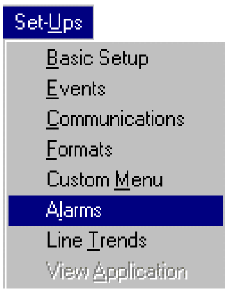
Thresholds are meaningful only for analog variables. In case of trigger variables the alarm conditions are coinciding with values (for instance 0 = no alarm, 1 = alarm).

7.5.3 Alarms acknowledge

Alarms acknowledgment occurs at application's runtime. For this reason the subject will not be discussed in detail here. We only recall here that this operation consists on ensuring that the Operator was actually made aware of the alarm occurrence. Obviously, explicit alarm acknowledgment could not be required for some specific alarms, as we shall see later.

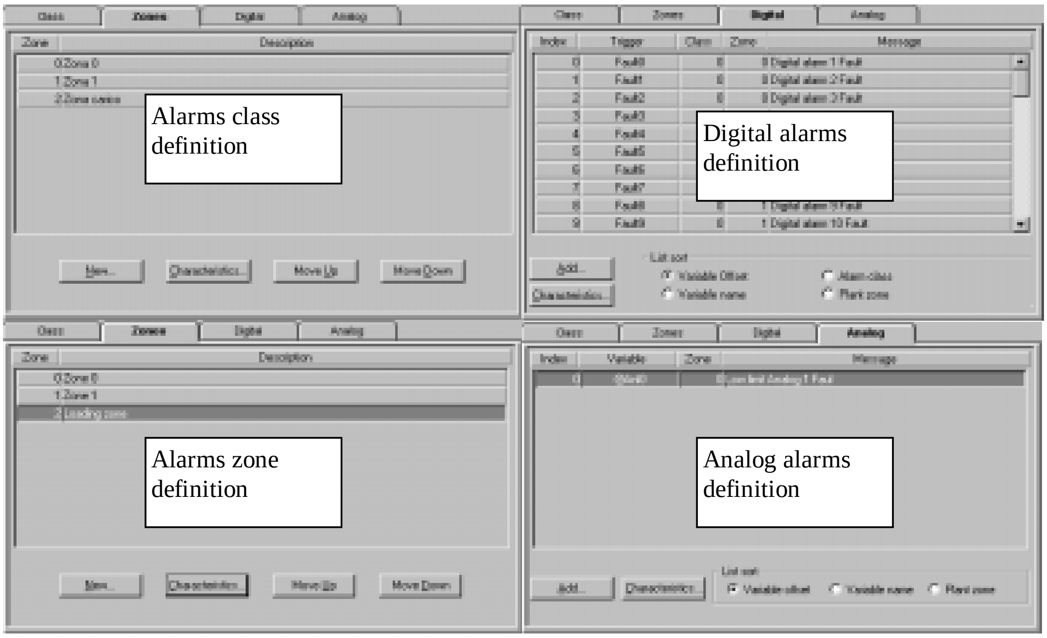
Acknowledgment is a class attribute, hence if acknowledgment is active for the class, then all alarms of that class will be considered acknowledged.

7.5.4 Alarms configuring



To define alarm classes you have to enter the ***Alarms Configuration*** window. You reach this window by selecting the ***Alarms*** entry of the ***Set-Ups*** menu, or pressing the dedicated button present in the *Main toolbar.*

This action puts on screen the main alarms window, where all required alarm parameters can be setup. The window contains four sections:



7.6 Interface Setup

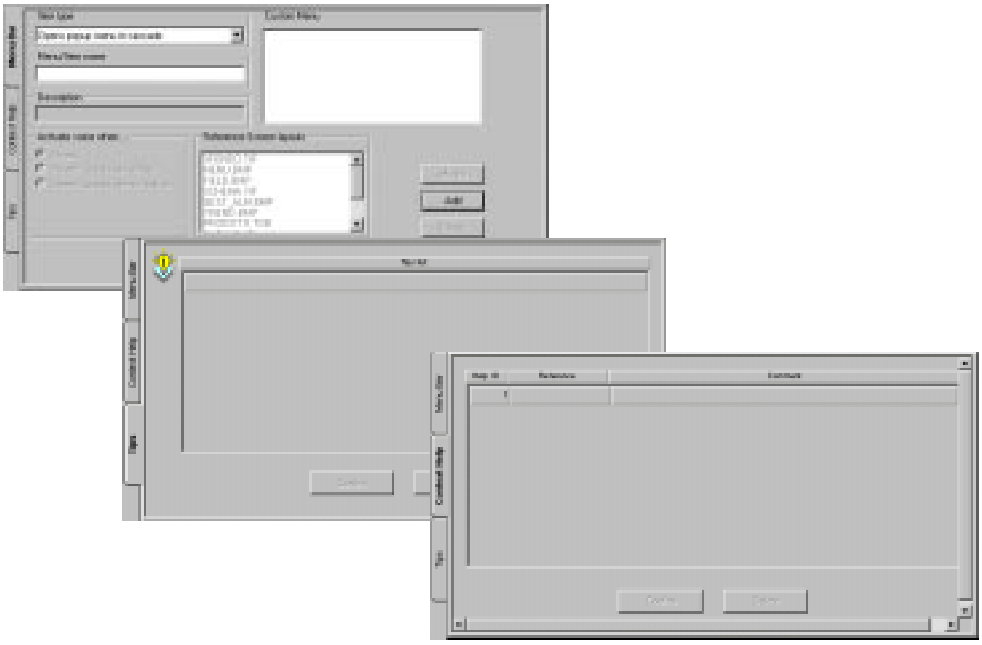
LOGOVIEW NT allows to create user interfaces suitable for the particular needs of individual applications. This interface includes three components.

Menus : They can be added to the runtime menu bar. Each menu contains one or several entries, fully configurable by the User.

Context Help : this is like a manual for the operator actually using the ***LOGOVIEW NT*** application. To this purpose you can configure the indexes required by certain commercial compilers (not included in the package) which will consequently create the actual help.

Tips : They are short indications appearing occasionally at runtime in front of the operator to help him using at best the ***LOGOVIEW NT*** application.

To setup the interface you only need to press the button present in the main toolbar. This action calls on screen the configuration window. This contains three parts, one for each of the three components described above.



8. Programming

The spreading of electronic computers and the significant success of the computer professionals has nourished in many people the desire to deal with programming. Unfortunately the approach to computers and computer programming has been confronted in the past till today in a not quite systematic way. The basic idea was that to know programming means to know a programming language.

Granted that the knowledge of a language constitutes a qualifying asset, the issue to be considered here is "why and how" it should be used.

The observation at the basis of the answer is that "first place should be assigned to computers and problems you can meet and solve by means of them, then the choices to be carried out in order to solve such problems and last the creation of tools (that is programs) actually providing solution means." This way programming languages are placed into their real frame. They are mainly convenient facilities available to instruct the computer, in a more or less simple and standard way, on the tasks the computer itself has to carry out in order to help us coping successfully with our problems.

The question truly at the core is, therefore, to split the computer task in a series of elementary jobs.

This section will gradually present some basic concepts useful to meet simply and correctly the issue of building program basic skeletons.

The chapter development is linear and the style definitely didactic. Nowhere easy shortcuts are promised neither fixed rules imposed. Our attempt here is to provide a trace for problems highlighting and solutions planning, this way acquiring analysis capabilities in problems approaching.

8.1 Writing an application

LOGOVIEW NT is an interpreter program, that is a program being able to execute the commands provided by the programmer. In its essence it is a dull program, since does not have its own, independent reasoning capacity. Rather it is conceived to accept the reasoning imposed to it by the programmer and to execute commands accurately, without obstructing the application logic with actions devised by itself.

If we can say from one side that ***LOGOVIEW NT*** is "logic-dependent", from the execution side it consists, instead, of an extremely reliable program. If it will not provide the desired results we can reasonably be certain that some programming error must be present.

Depending on the program implemented in ***LOGOVIEW NT***, the computer will carry out whichever job is to be done, in practice substituting the human actions, repeating at high speed and accuracy the particular tasks it has been instructed to perform.

To write an application program means therefore to provide LOGOVIEW NT with the procedure to be followed in order to reach the problem solution.

All this means that we ought to devise an algorithm (reasoning) to process the input data and generate output data (meaning results). It happens often in everyday life that we have to describe the procedure to be followed to solve a problem. In so doing we normally do not experience particular difficulty, since the person we are talking to needs usually only broad information. He will be able to positively cope with the situation even if, while applying the procedure, he faces circumstances not foreseen by its originator.

To teach a computer is different, since you must define all elementary steps and take into account all conditions that will possibly occur.

Only in case if very simple problems we manage to build into our memory (that is remember) a complete reasoning skeleton. In the majority of cases we are forced to help ourselves with paper notes and graphs allowing us to build solutions and verify their completeness and consistency. Only at this point we are in the position to start the application programs, which now appear naturally to consist just into translations of our reasoning previously put on paper, following the rules of the chosen programming language (or the language the computer to be programmed has available).

Hence, given a problem to be solved by computer, one ought to draw first the program design. Only after the correctness and completeness of the reasoning has been verified the program code can be written down.

8.2 The programming tools

In the previous chapter we introduced a fundamental component of ***LOGOVIEW NT*** applications: the screen layout. Now we would like to talk about the three tools which make ***LOGOVIEW NT*** a complete development environment, capable of meeting the needs of the most demanding developer. These three tools are *variables, events and communications.*

8.2.1 Variables

In any industrial environment, as well in everyday life, we are constantly facing items continuously changing their appearance, shape and value while time is passing. We interact regularly with many entities not remaining constant in time. Even at the risk of playing with words, we can state that everything varying is a variable. Obviously our purpose here is to limit our analysis to the variables of the plants ***LOGOVIEW NT*** is supposed to supervise.

What kind of variables can be found in a plant? Many, for instance pressures in pipes, motor rotational speeds, furnace temperatures, valve states and so on.

As can easily be observed, all these constitute variables, although they differ from each other because of their related meaning.

Besides these *Plant Variables,* other variables exist which will be used exclusively by the ***LOGOVIEW NT*** application. These variables make up the program memory. They can be used for the widest purposes, such as keeping an cycle index, saving a piece of data to be later put into an archive and so on. We can not make a list of everything will be required inside a given ***LOGOVIEW NT*** application.

Whichever purpose a variable might have, the substantial matter remains the same: a variable must be defined and used any time a changing quantity must be stored.

In the attempt of coping with all possible cases ***LOGOVIEW NT*** puts to the User's disposal six types of variables.

A ***BYTE*** type variable is integer, ranging between **0** and **255**. It provides therefore good space saving in case you deal with integer quantities not oscillating outside this interval.

A ***WORD*** type variable is also integer, but its oscillating range is bigger than the Byte variable one. Its values can range between **-32768** and **32767.**

A ***TRIGGER*** type variable can take only two values, 0 and 1. No other value is physically possible. It is therefore a Boolean variable.

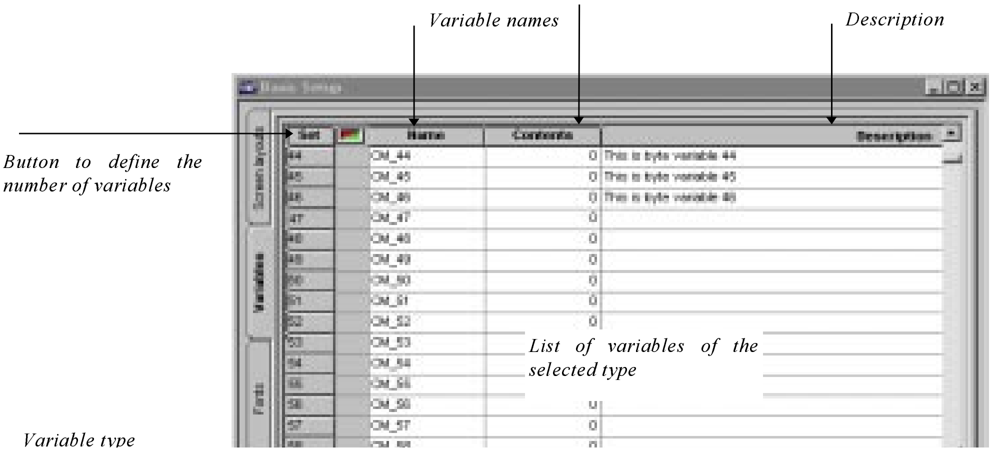
A ***REAL*** type variable is real. Therefore it can take values in the range **-1e+38** and **1e+38**.
A variable of such a type is the only one able to represent rational numbers.

A ***DOUBLEWORD*** variable is integer ranging between **-2147483648** and **2147483647**. It is therefore longer than a Word, representing a wider value range.

A ***STRING*** variable contains alphanumeric characters. It is considered in a different way with respect to the other variables. Many arithmetic operators can not be applied, since this variable does not represent a quantity, but contains only a text.

8.2.2 Define variables

As said, ***LOGOVIEW NT*** can manage up to 6 types of variables. To define one or more variables, you have to call the Variables section in the *Basic Setup* window.



Set		Name	Contents	Description
44		CM_44	0	This is byte variable 44
45		CM_45	0	This is byte variable 45
46		CM_46	0	This is byte variable 46
47		CM_47	0	
48		CM_48	0	
49		CM_49	0	
50		CM_50	0	
51		CM_51	0	
52		CM_52	0	
53		CM_53		
54		CM_54		
55		CM_55		
56		CM_56	0	
57		CM_57	0	
58		CM_58	0	

Variable names

Description

Button to define the number of variables

List of variables of the selected type

Variable type

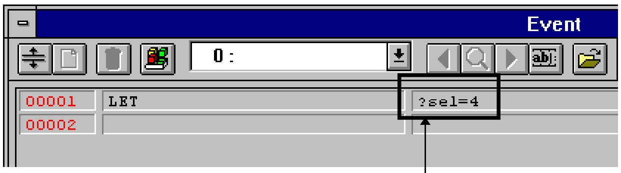
This window allows to define all ***LOGOVIEW NT*** variables.

8.2.3 Using variables in Events

Using variables in events is an extremely frequent operation. To get access to a variable you only need to specify its name or its offset. Each of these two attributes uniquely identifies a variable. Any reference to a variable for whatsoever purpose can therefore be done by specifying its offset or its name, just making sure to add also the prefix identifying the variable type. The available prefixes are listed in the following table.

TYPE	*PREFIX*
TRIGGER	?
BYTE	!
WORD	#
DOUBLE WORD	:
REAL	@

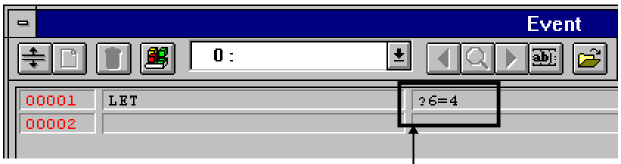
If, for instance, we want to make an assignment (LET statement) to the Trigger variable of name sel and offset 6, we could write indifferently:



```
00001   LET   ?sel=4
00002
```

Using the variable name

? identifies the variable type, in this case trigger
sel *name of the variable*
=4 assigns the value 4.



```
00001   LET   ?6=4
00002
```

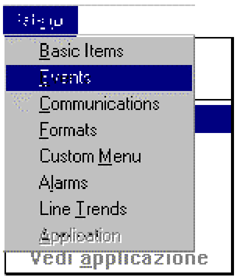
Using the variable offset

? identifies the variable type, in this case trigger
6 variable offset
=4 assigns the value 4.

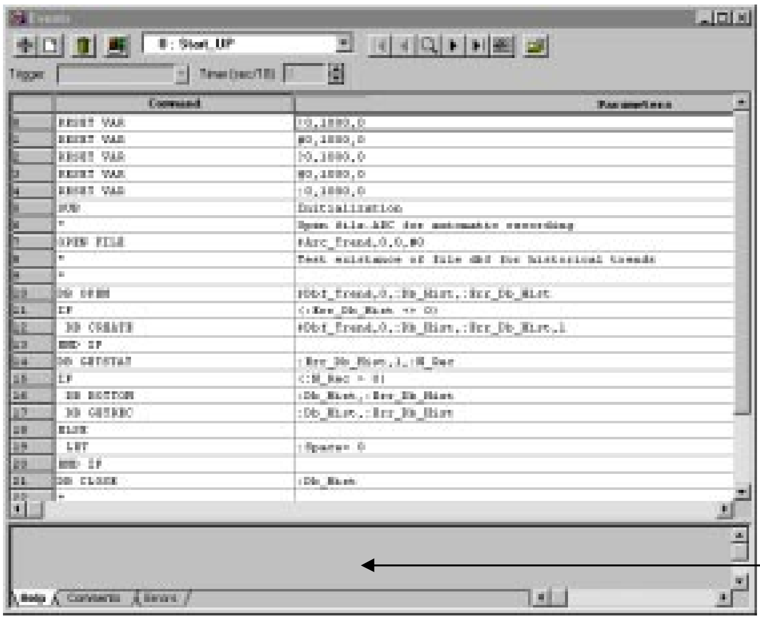
8.2.4 The Events

LOGOVIEW NT makes available an internal programming language which allows to create the application events. The events are constituted by program items (similar to procedures) which can be run in parallel taking advantage of the ***LOGOVIEW NT*** really multitask architecture.

8.2.4.1 *The Events Editor*



To get access to the Events Editor, just press the button available in the *Main Toolbar,* or activate the corresponding menu entry.



	Command	Parameters
0	RESET VAR	:0,1000,0
1	RESET VAR	#0,1000,0
2	RESET VAR	?0,1000,0
3	RESET VAR	#0,1000,0
4	RESET VAR	:0,1000,0
5	SUB	Initialization
6	*	Open file.ARC for automatic recording
7	OPEN FILE	#Arc_Trend,0,0,#0
8	*	Test existance of file dbf for historical trends
9	*	
10	DB OPEN	#Obj_Trend,0,:Db_Hist,:Err_Db_Hist
11	IF	(:Err_Db_Hist <> 0)
12	DB CREATE	#Obj_Trend,0,:Db_Hist,:Err_Db_Hist,1
13	END IF	
14	DB GETSTAT	:Err_Db_Hist,1,:N_Rec
15	IF	(:N_Rec > 0)
16	DB BOTTOM	:Db_Hist,:Err_Db_Hist
17	DB GETREC	:Db_Hist,:Err_Db_Hist
18	ELSE	
19	LET	:Spare= 0
20	END IF	
21	DB CLOSE	:Db_Hist
22	*	

Events toolbar

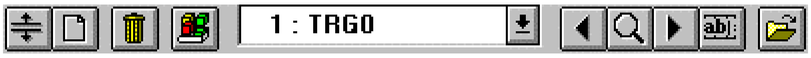
Trigger associated to the event start

List of statements

Help, comments and errors in statements

8.2.4.2 *Events Toolbar*

This toolbar summarizes the most important controls present in the events editor:



TOOLBAR

This button allows to insert a statement into any arbitrary event's line.

This button allows selecting all statements present in the current event.

This button allows deleting the selected statements.

This button allows configuring the event's parameters and local variables.

1 : TRGO This list contains all events defined in the application. To display another event just select from this list.

This button displays the event preceding the one presently displayed.

This button allows performing inquiries on an event's object.

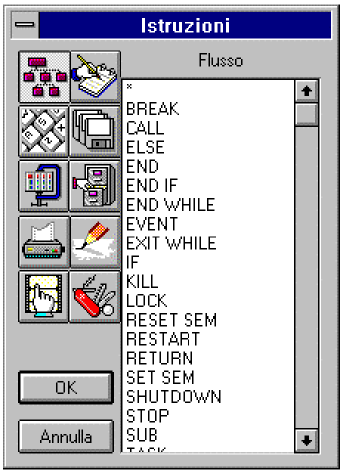
This button displays the event following the one presently displayed.

This button allows defining the label of the event presently selected.

This button allows defining the number of events present in the application.

8.2.4.3 *Adding statements*

The event statements are displayed with the addition of line numbers in the main list. To add a statement just position the mouse cursor on the last line and press the left key. This action calls on screen the statements window, where all statements defined in ***LOGOVIEW NT*** are listed.



In this window you can find all ***LOGOVIEW NT*** statements separately listed according to their classes:

Flow : This class contains all statements regarding the event flow control, including statements to implement cycles, conditional software paths and so on.

Variables : This class contains all statements regarding variables management, including assignment, initialization, string concatenation and so on.

User Interface : This class contains all statements regarding the management of the Operator interface, including alarms management, user inputs, passwords and so on.

Files : This class contains all statements regarding files management, including file creating, deleting, historical archives record management and so on.

Plc : This class contains all statements regarding the management of communications with the plant floor, including reading and writing to the connected servers.

Data Base: This class contains all statements regarding the management of Data Bases, including configuration of Data Base formats, creation, addition, search and so on.

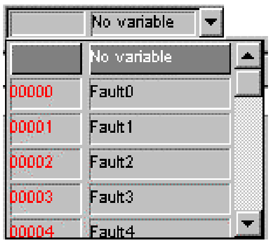
Printer: This class contains all statements regarding printing, including printing of the configured forms.

Graphics : This class contains all statements regarding graphics, including drawing, box management and so on.

Animations : This class contains all statements regarding the animations management, including MCI, sprites and so on.

External Commands : This class contains all statements added by means of the ***LOGOVIEW NT*** extensions.

8.2.4.4 *Association of a trigger*



No variable

No variable

00000 Fault0

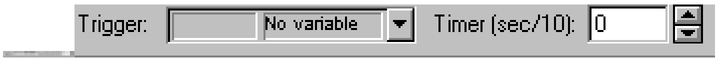
00001 Fault1

00002 Fault2

00003 Fault3

00004 Fault4

LOGOVIEW NT allows to associate the start of an event to value change of a trigger. This allows to conceive events linked to certain plant floor state changes. The trigger association is managed by the toolbar present in the upper part of the editor window.



Trigger: No variable Timer (sec/10): 0

To associate a previously defined trigger, just select it from the list. In addition you can also associate a timer to start the event as soon as the time has elapsed.

8.2.5 Communications

In a large size plant many communication problems are present. The communication external device can be either a server driving a PLC or another ***LOGOVIEW NT*** station. There are mainly two reasons why ***LOGOVIEW NT*** needs communication with external devices. First to keep updated the display of all plant floor data allowing the operator to monitor how satisfactorily the process is performing. This is therefore a reading problem. Second to change values of plant floor physical quantities. This is a quite peculiar function which will be discussed in chapter 11, since regards a rather delicate aspect of the ***LOGOVIEW NT*** running conditions. In this chapter we deal with linking blocks of variables external to ***LOGOVIEW NT*** to corresponding blocks of variables defined inside ***LOGOVIEW NT.*** A supervision system could ask for rather complex configurations. There could be a number of servers driving interface boards connecting devices such as PLCs as well as several ***LOGOVIEW NT*** stations communicating to each other.

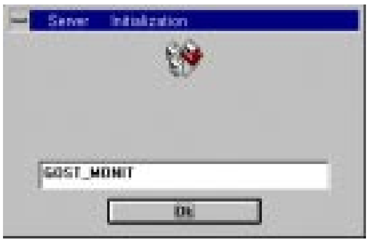
8.2.6 Server and Client

Servers are objects providing services. Clients are objects taking advantage of such services. In our case a logical configuration could include a number of servers providing services of floor connecting devices, usually boards connecting PLCs, to a client, which might be a ***LOGOVIEW NT*** station. Also, not just one client might require services from a given server, but several ones at the same time. Servers and clients must be connected in a network or must run locally on a single computer.

In our case the server task is to grant access to variable blocks mapped on a PLC. ***LOGOVIEW NT*** allows to map server variables on own internal variables, consequently available to develop the application. It is important to highlight that this configuration creates two conceptual links.

The first is the one created by a server mapping PLC memory blocks on own variables located at sequential addresses starting from 0.
The second is the one created by ***LOGOVIEW NT*** mapping this variable block into an internal block of own variables. While developing the application you need therefore to pay attention to how server variable blocks are configured before proceeding to mapping such blocks on ***LOGOVIEW NT*** variables. ***LOGOVIEW NT*** is capable of communicating with servers developed according to the specifications described in the servers development manual.

Multiple servers can run on a single computer. Hence you must indicate in a unique way the server with which you want to communicate.
A server is identified by two parameters: the node name and the station name. The node name is the name identifying the server hosting computer in the network, in case the server is located on a computer different from the client one. In case server and client reside both in the same computer we talk of local connection. In this case the node name parameter is not relevant to the remaining of our discussion. The station name, instead, is meaningful in both cases since it identifies the server among the ones residing on that node with which communication has to occur. Usually a server allows configuring the station name. This way multiple instances of the same server can run with different names on the same node.

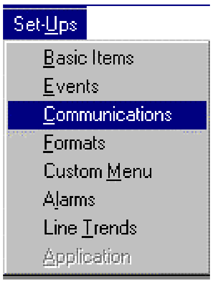


GOST_MONIT

Usually servers will ask to enter the name to be given to the station at their startup.
To get more information on how to get this parameter you need to read the server user manual.

8.2.6.1 *Configuring Communications*

Before adding new server stations you need to know the node name on which they reside (if this is different from the node on which ***LOGOVIEW NT*** resides). You need also to know the station name.



To add a server station select the ***Communications*** entry in the ***Set-Ups*** menu, or press the button present in the main toolbar.

This action will activate the ***Communications Configuration*** window, where you can configure both stations and server variable blocks to be connected to ***LOGOVIEW NT*** variables.

To know more about using communication features read chapter 11.

9. Output on other supports : Formats

LOGOVIEW NT offers other tools to interact with the operator. These tools increase the possibilities of managing the plant floor. They are the formats: *Data Base formats, Historical Archives* and *Print formats.*

9.1 Historical Archives

Historical archives allow to store variables in a most flexible and user friendly manner. The archiving can be managed directly by ***LOGOVIEW NT*** using configurable automatic procedures. A correct use of these archives allows monitoring of the important plant parameters in time.

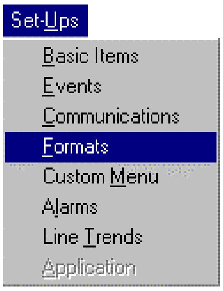
9.1.1 Introduction

The historical archives are files that ***LOGOVIEW NT*** generate to allow reconstructing past occurrences in the plant. Usually the data are examined formatted in a line-trend manner, since this way allows a quick visual evaluation. The internal format of the archives is proprietary and is not therefore directly usable but, by converting the data by the driver ODBC, it is possible to examine the same data using other programs which include this protocol.
The configuration of these archives is done through a window which allows specifying all parameters for the record format and its use. ***LOGOVIEW NT*** allows to configure and open at the same time up to twenty different formats. Each historical archive has its peculiar record format to fulfill specific requirements. The extension of ***LOGOVIEW NT*** archive files is .ARC. The formats are defined in a specific section of the application, while its usage is done by statements included in the application's events.

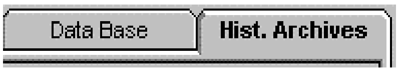
9.1.2 Using Historical Archives

Historical archives are very useful to monitor the plant in time. This way the operator can keep under scrutiny significant parameters for plant production. This is the main reason for having historical archives.

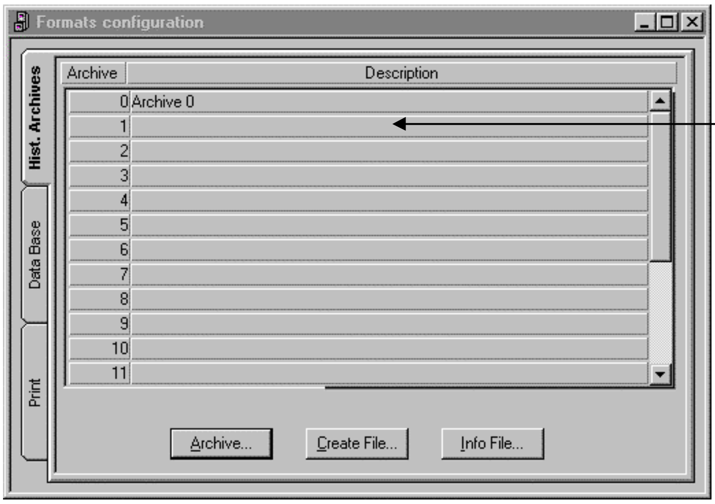
9.1.3 Creating Historical Archives



To define a Data Base format just select ***Formats*** from the ***Set-Ups*** menu or press the button from the main toolbar. This action activates the ***Formats Configuration*** window, through which you can configure ***Historical Archives, Data Base*** and ***Prints***.



The ***Historical Archives*** window allows to define up to twenty different archives, each one with its own record format.



Archive	Description
0	Archive 0
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Available formats list. Double click to configure a format

9.1.4 Configuring Records

Each historical archive is constituted by records, in turn made up by sets of variables. To configure a record is therefore necessary to specify which blocks of variables have to be saved in the archive.

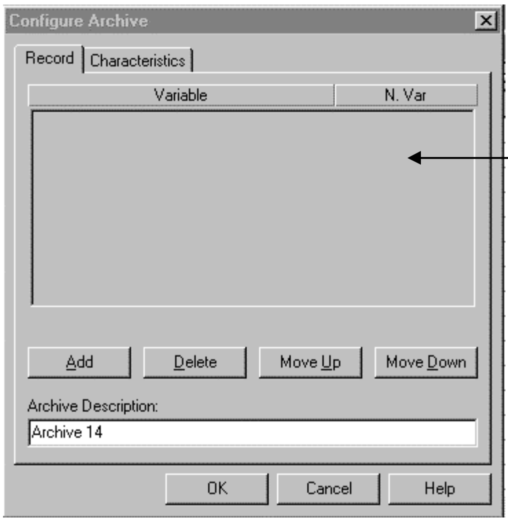
To configure a record just press the button

Archive...

to be fount in the format window. Following this action the window ***Configure Archive*** is displayed, where you can specify composition and characteristics of the record, and mode of data writing in the file.

Record | Characteristics

The record definition is done through the section ***Record***, where you will be able to specify the variable list.

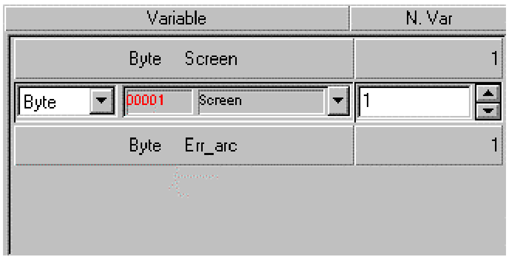


Archive Description: Archive 14

List of variables making up the record

To insert a variable block just press the button Add

By this action you add one line in the list; select it to specify its characteristics.



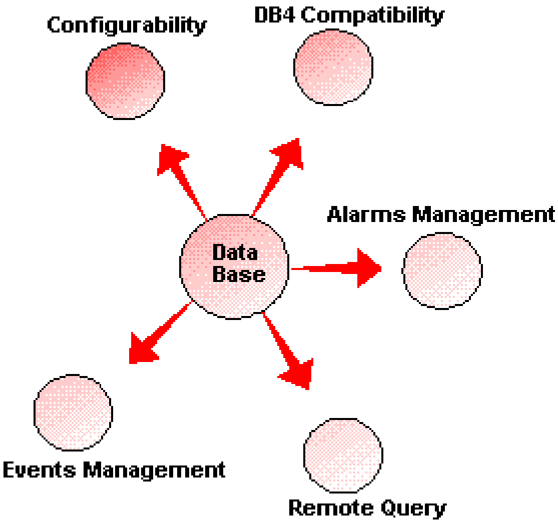
Variable	N. Var
Byte Screen	1
Byte ▼ 00001 Screen ▼	1
Byte Err_arc	1

A variable block is a set of contiguous variables, defined by three characteristics: variable type, offset and length. To select these three characteristics ***LOGOVIEW NT*** makes available 3 combo-boxes. The first two contain the list of available variable types and the list of the defined variables of the specific type.

9.2 Data Base

The subject Data Base is rather wide and complex, therefore we will not be able to examine it exhaustively in this introductory manual. We will only supply a summary of the tools ***LOGOVIEW NT*** make available, assuming the basic concepts are well known by the reader.

9.2.1 Data Base Usage

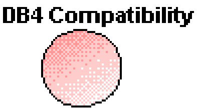


In this paragraph we will briefly explain the reason why a complete data base support, partially configurable, to be utilized within the events, has been included in ***LOGOVIEW NT***.

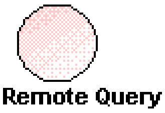
At the beginning, SCADA programs like ***LOGOVIEW NT*** always included some archiving procedure, using proprietary formats, therefore different from one program to the other. This forced the user to utilize the program itself only to examine all the data during the normal running of the plant. This was occasionally a great limitation, since data are archived sequentially, and often the tools available for further processing were not adequate.

This has forced programmers to devise complex ways to format the data, to have them in a generally readable aspect, for other programs to use them, at times written ad hoc. ***LOGOVIEW NT*** is quite advanced in this matter: in fact, the designers have given the package a complete relational data base system, compatible with the standard DBF.

This makes ***LOGOVIEW NT*** a very good package also for writing data base management programs; also, other powerful tools are available, to write archives readable by other programs such as FOX pro, Access, Clipper etc. These tools allow to sort data in any way, use filters to see only specific records, etc.



The main characteristic of the data base engine found in ***LOGOVIEW NT*** is, as said above, the compatibility with the DBF format. This format, derived from the DBase program, is nowadays a standard in the PC environment. The choice was therefore obvious for ***LOGOVIEW NT***: organized this way, its data are accessible by the vast majority of



existing data base programs. The few users still utilizing programs not compatible with this standard can convert their data using packages readily available on the market.

It is of the outmost importance the possibility of accessing ***LOGOVIEW NT*** archives from a remote station. ***LOGOVEW NT*** allows to generate an archive on one station and at the same time read and even update it from a remote station, connected via network, without interrupting the application program running on the first station. There are many other possibilities...

Configurability

Regardless of the complexity of the functions included in this system, ***LOGOVIEW NT*** allows to configure the most commonly used functions via some dialogs. Mainly, the configuration is for creation and updating record formats, selection filters and sorting criteria. All record fields are temporarily stored inside ***LOGOVIEW NT*** variables, and this allows sharing the data base fields with the various output objects available in the system.

Events Management

Besides the configurability, the great flexibility is given by the wide choice inside the event statements. Two levels of programming exist: the first makes use of the configured formats and allows a very easy creation and management of archives; the second level, slightly more complex, can be used to access all functions proper of relational archives. The functions available are many, and allow to resolve many different situations. Furthermore, being a multi-task system, ***LOGOVIEW NT*** makes use of its peculiar advantages without penalizing the system performances.

Alarms Management

Last but not least, also the historical management of alarms is achieved through the data base engine. This translates into the fact that also the alarms archives can be accessed by external programs or, if necessary, by ***LOGOVIEW NT*** itself, from a remote node, to achieve the greatest usage flexibility.

9.2.2 Data Base Structure

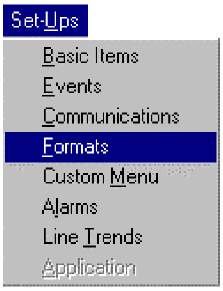
LOGOVIEW NT can manage up to 32767 different types of Data Base. Each Data Base has a specific record format, built to specifications. Every Data Base is saved by ***LOGOVIEW NT*** in a file with extension .DBF; complete compatibility with applications supporting this format is guaranteed.

The Data Base record fields defined in ***LOGOVIEW NT*** are associated to variables: each field is tied to a variable. This way, when adding a record, the values will be taken directly from the associated variables.

It is also possible to open a DB of unknown structure from ***LOGOVIEW NT***. In this case the only limitation is the impossibility of utilizing the statements of read and write of the entire record (***DB PUTREC*** and ***DB GETREC***).

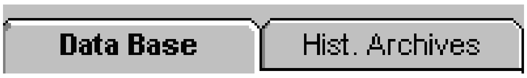
To manage Data Base not defined inside ***LOGOVIEW NT*** statements for reading and writing single fields have been implemented (***DB PUTFIELD*** and ***DB GETFIELD***); these statements do not make reference to specific variables, leaving great flexibility to the user who this way can build and manage very articulated Data Bases.

Formats are defined inside a specific section of the application; the use of Data Base is instead done through statements of the application events.

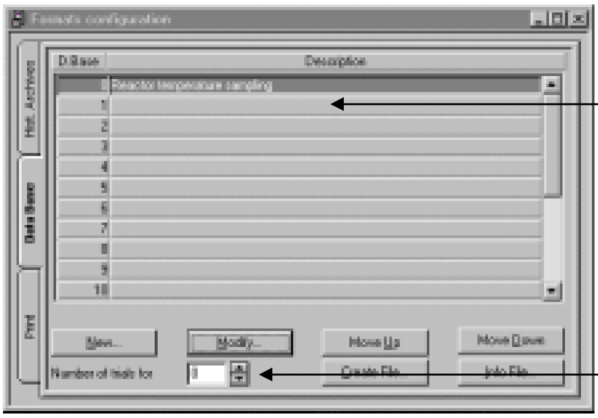


9.2.3 Creating a Data Base

To define a Data Base format, select ***Formats*** from the ***Set-Ups*** menu, or press the button in the main toolbar. This action activates the ***Format Configuration*** window.



The ***Data Base*** section of this window allows to define the record formats used by ***LOGOVIEW NT*** Data Base.



D.Base	Description
0	Reactor temperature sampling
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

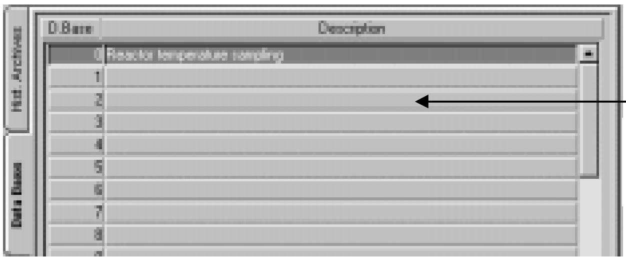
List of the available Data Base formats

Number of trials Logoview NT is allowed to make on accesses to a locked file

Each configurable Data Base format corresponds to one line in the main list contained in this window.

9.2.4 Configuring Records

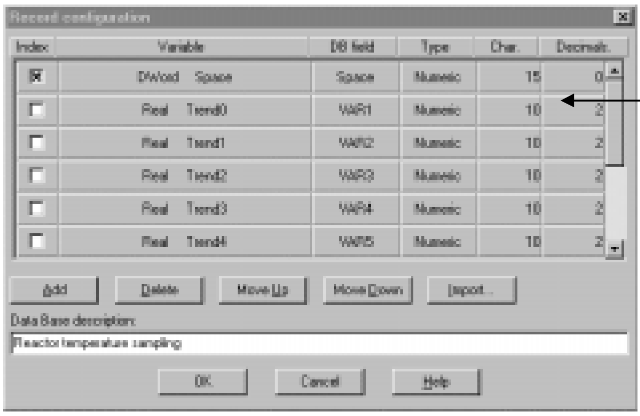
To add a new record just press the button New.... To modify the configuration of an existing record select the corresponding line in the main list present in the ***Data Base*** section of the ***Formats configuration*** window,



A double click can also be used to configure a record

then press the button Modify...

Following this action the window ***Record configuration*** is displayed, where one can build the record by specifying its characteristics.



Index	Variable	DB field	Type	Char.	Decimals
☒	DWord Space	Space	Numeric	15	0
□	Real Trend0	VAR1	Numeric	10	2
□	Real Trend1	VAR2	Numeric	10	2
□	Real Trend2	VAR3	Numeric	10	2
□	Real Trend3	VAR4	Numeric	10	2
□	Real Trend4	VAR5	Numeric	10	2

Data Base description: Reactor temperature sampling

Fields making up the record

To add a field, just press the button Add.

9.2.5 Using Records

Records are mainly used in events, through Data Base statements. Records are stored in the Data Base sequentially. So we can think of a data base as a list of records.

It is important to remember that when opening a file, ***LOGOVIEW NT*** positions its pointer to the first record found according to the sorting method specified (TAG), not necessarily record 0.

The pointer can move inside the list in one of two ways:

- ♦ By using the statement DB GO, which allows to move to an absolute address, independent from the selected tag. This can be used, for instance, to return to a record whose index number has been previously memorized.

- ♦ The second method uses the current tag and allows to move to the beginning of the DB (***DB TOP***), to the end (***DB BOTTOM***), or a relative movement with respect to the current *record* (***DB SKIP***).

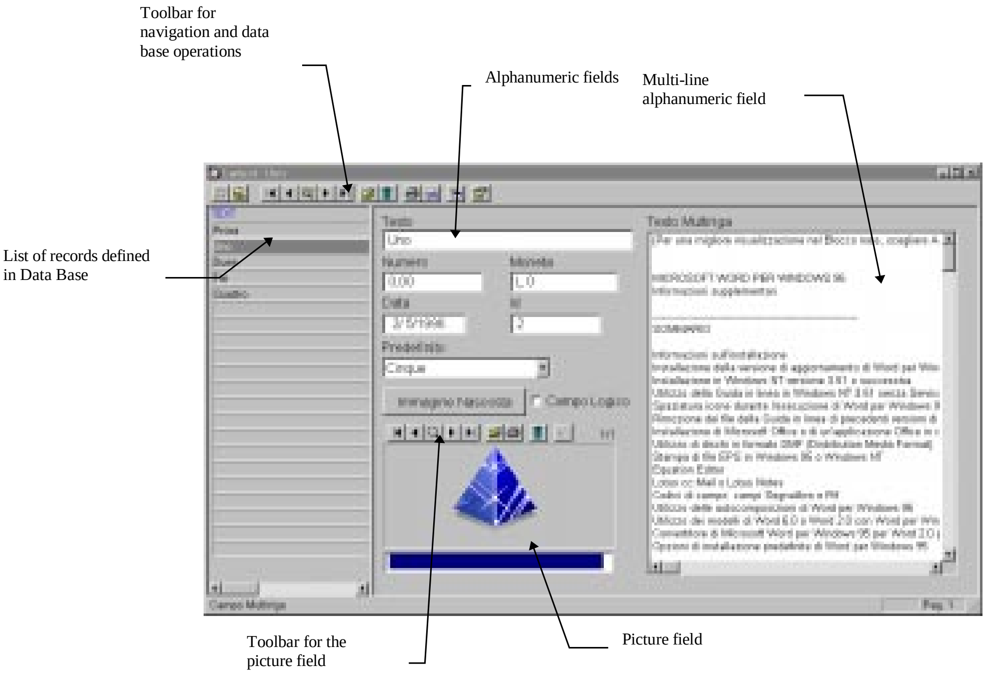
Every time the pointer is positioned onto a *record*, ***LOGOVIEW NT*** automatically writes on disk the content of *records* previously loaded in memory, then reads and makes available the new *record*.

9.2.6 Data Base Masks

Specific windows have been developed inside Logoview to make displaying Data Base files easier. Data Base masks are managed in a way analogous to screen layouts and, at runtime, can be opened by the usual change screen procedures. Data Base masks can not be displayed or modified inside the development environment. To create them you must use the package **Titano** (to be bought separately).

Using **Titano** you can create masks, Data Base print reports, and even the Data Base itself. Data Bases managed through masks allow to archive also pictures and documents in standard format, such as Word or Excel documents. Logoview runtime can read and use objects created by **Titano** exactly the same way **Titano** itself does, therefore from the Data Base masks you can:

- ■ Navigate inside the Data Base
- ■ Modify/Delete the content of the Data Base
- ■ Search
- ■ Query
- ■ Print pictures and masks as they are
- ■ Reports print


Toolbar for navigation and data base operations
Alphanumeric fields
Multi-line alphanumeric field
List of records defined in Data Base
Toolbar for the picture field
Picture field
Prima
Uno
Due
Tre
Quattro
Immagine Nascosta
Campo Logico
1/1
Campo Multiriga
Pag. 1

Testo: Uno

Numero: 0,00

Moneta: L.0

Data: 2/5/1996

Int: 2

Predefinito: Cinque

Testo Multiriga:

(Per una migliore visualizzazione nel Blocco note, scegliere A

MICROSOFT WORD PER WINDOWS 95
Informazioni supplementari

SOMMARIO

Informazioni sull'installazione
Installazione della versione di aggiornamento di Word per Win
Installazione in Windows NT versione 3.51 e successiva
Utilizzo della Guida in linea in Windows NT 3.51 senza Servi
Sparizione icone durante l'esecuzione di Word per Windows 9
Rimozione dei file della Guida in linea di precedenti versioni di
Installazione di Microsoft Office e di un'applicazione Office in
Utilizzo di dischi in formato DMF (Distribution Media Format)
Stampa di file EPS in Windows 95 o Windows NT
Equation Editor
Lotus cc:Mail e Lotus Notes
Codici di campo: campi Segnalibro e Fill
Utilizzo delle autocomposizioni di Word per Windows 95
Utilizzo dei modelli di Word 6.0 e Word 2.0 con Word per Win
Convertitore di Microsoft Word per Windows 95 per Word 2.0 (
Opzioni di installazione predefinite di Word per Windows 95

9.3 Print

We have given appropriate consideration to the fact that it is quite useful to be able to put on paper the display of significant plant variables, to allow a greater group of people to evaluate production. Especially for big plants, where the quantity of significant parameters is just as big, "memory on paper" becomes very useful. It is a tradition, and therefore easier, to look at trends on paper, and the historical picture of production will always be an important factor for plant managers.
LOGOVIEW NT allows to build quite elaborate printouts in a very simple and flexible manner through the Print Formats.

9.3.1 Structuring Print

Print under WINDOWS is structured in pages. This is the main difference between printing under WINDOWS and printing under DOS. It is quite common, under DOS operating system, to print one line at the time, while this is not possible under WINDOWS. WINDOWS requires to structure the whole page to be printed before sending it to the output device.

The ***LOGOVIEW NT*** section described below takes care of structuring all pieces of information to be printed in a formatted page, which will be printed. ***LOGOVIEW NT*** allows configuring an unlimited number of print formats. It is therefore possible to have a specific format for each group of variables to be printed. The printing operation is made up of two phases: in the first the page to be printed is formatted; in the second the print actually takes place, using the available print statements.

The format is therefore the structure of the printing page.

9.3.2 Print Formats

Each print format has the size of one page. The format can contain two kinds of objects: Static objects and Dynamic objects. With static objects you can build the user friendly part of the printout, such as inserting frames, outline portions of the page, insert strings. The static objects available are Lines, Rectangles, Fixed texts, Pictures, OLE objects and system variables (time, date, etc.).

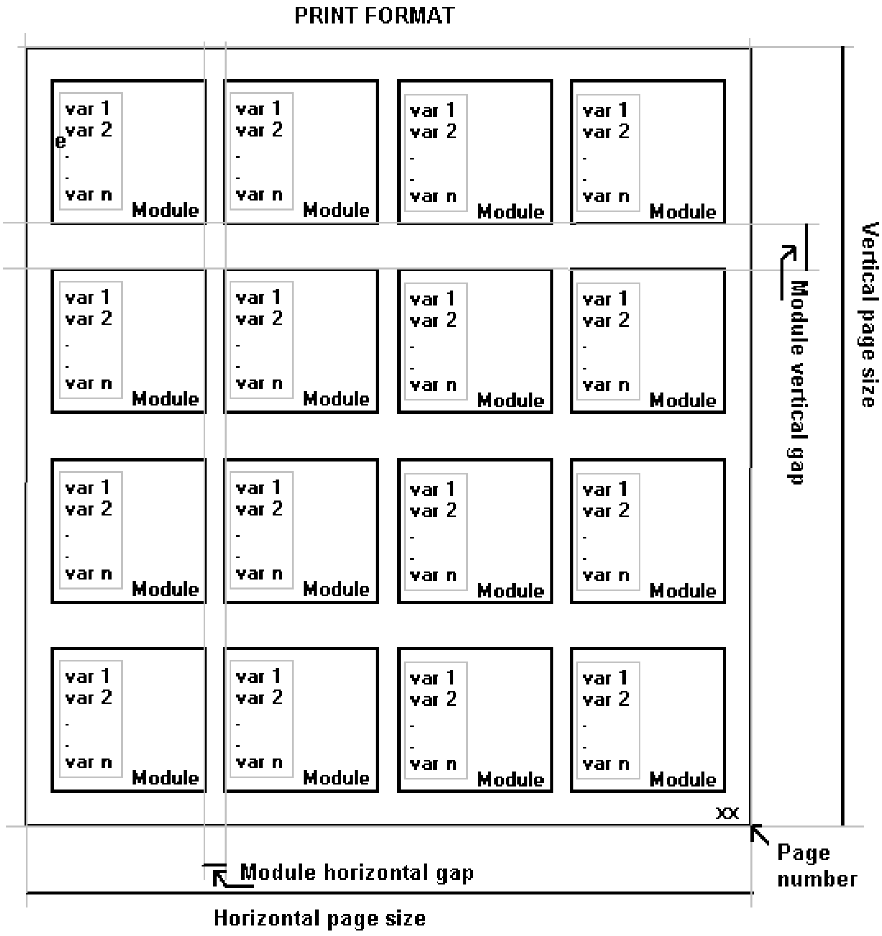
The dynamic objects are the application's variables. When defining a print format only the reference to the variable is specified (address). The actual value will be put in place by ***LOGOVIEW NT*** at the moment of printing.

Static objects can be positioned anywhere in the page, while dynamic objects are grouped in Modules.

9.3.3 Modules

A format can be made up by one or more modules, containing each one or more variables.

The following is an example of printout configuration:



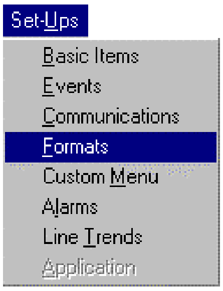
In this example, every module contains the same variables.

This a logical way of proceeding: first of all the programmer defines the content of the module (variables to be printed). Then he defines the quantity of modules that can fit in a page, horizontally and vertically. This way every time one wants to print a set of variables one can send the command to print a module. ***LOGOVIEW NT*** will fill the module with current variable values, and “park” it inside the still incomplete page. Only when all modules in the format have been compiled, the print will actually take place. This is because, as said before, WINDOWS only prints full pages. Using this procedure you will obtain on one page the values that a set of variables had at the specified moments in time, when each print module statement was issued.

Besides variables you can put static objects on the page. In particular it is always very useful to have date and time printed on the module. This way you can know for sure when each print module statement was issued, regardless the time when the page is actually printed.

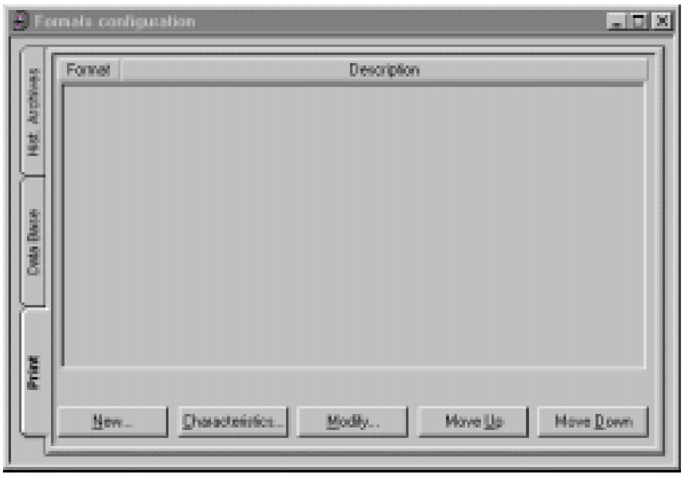
Each format is identified by its name, to be specified in the print statements.

9.3.4 Creating a Print Format



To define a Print Format select ***Formats*** from the ***Set-Ups*** menu or press the corresponding button in the main toolbar. This action activates the ***Formats configuration*** window.

Section ***Print*** of this window allows to define all printing formats necessary to the application.

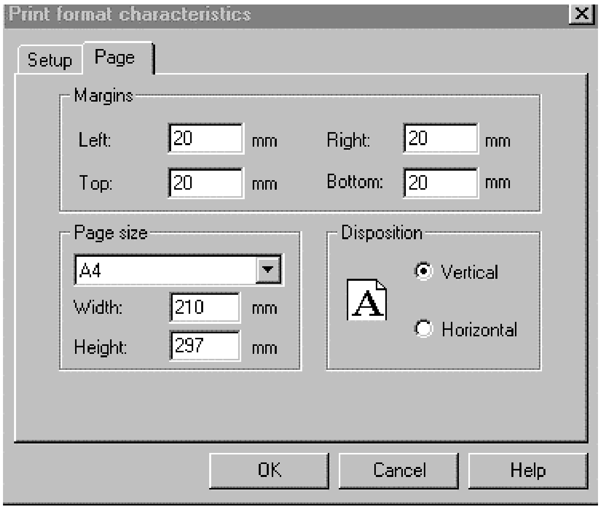


List of defined formats

To create a new print format position the mouse on button and press the left mouse button. The window ***Print format characteristics*** will be displayed. This window is made up by two sections: ***Setup*** and ***Page***.

Section ***Setup*** allows to specify the format characteristics.

Section ***Page*** allows to specify the size of the page containing the format.



Print format characteristics

Margins

Left: 20 mm Right: 20 mm

Top: 20 mm Bottom: 20 mm

Page size

A4

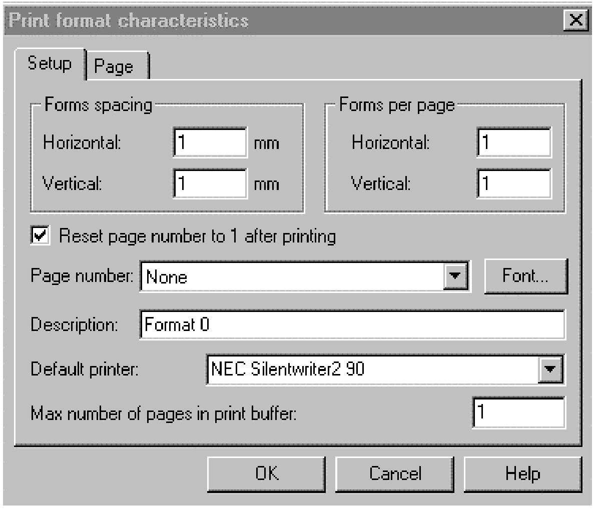
Width: 210 mm

Height: 297 mm

Disposition

● Vertical

○ Horizontal



Print format characteristics

Forms spacing

Horizontal: 1 mm

Vertical: 1 mm

Forms per page

Horizontal: 1

Vertical: 1

☑ Reset page number to 1 after printing

Page number: None

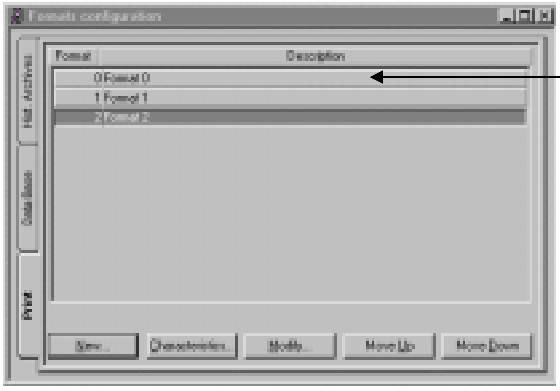
Description: Format 0

Default printer: NEC Silentwriter2 90

Max number of pages in print buffer: 1

9.3.5 Print Format Editor

The Print Format Editor allows to position objects to be printed within the page. To open the editor procedure position the mouse over the line corresponding to the format you want to edit and double click.



Format	Description
0 Format 0	
1 Format 1	
2 Format 2	

Double click on the line corresponding to the format

or, after selecting the line, push the button Modify...

The format editor allows to insert all the objects you want to copy on paper in a simple and flexible manner. The editor works in a window.
The window is divided in various areas, each of which has a specific role in the definition of the printing format. The editor can work full page or visualize one module at a time.
The type of display depends on the button
This button can be found in the toolbar of the same window.

Button up: ***editor in page mode.***

Button down: ***editor in module mode.***

When the editor works in page mode, it displays the whole page, thus providing a broad picture of the formatting taking place. In page mode you can only insert static objects.
When the editor works in module mode, it displays only the module on which you are working. In module mode you can insert both static and dynamic objects (variables).

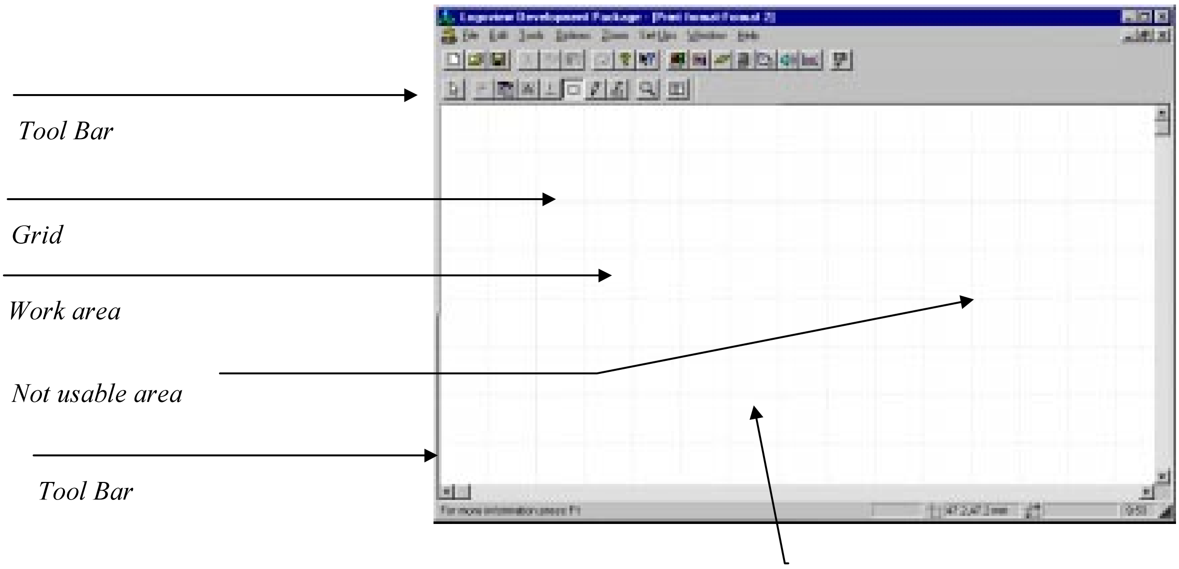
Dynamic objects are inserted via the button , which therefore is active in module mode only.

9.3.5.1 *Page Mode*

In page mode the entire page with its defined modules is displayed.

To switch to page mode make sure the button is up.
With the button down the editor is in module mode.

The window in page mode looks like this:



In this mode you can insert static objects only. The ayed exactly how it will be d. The working area is the zone inside the ***page margin***.

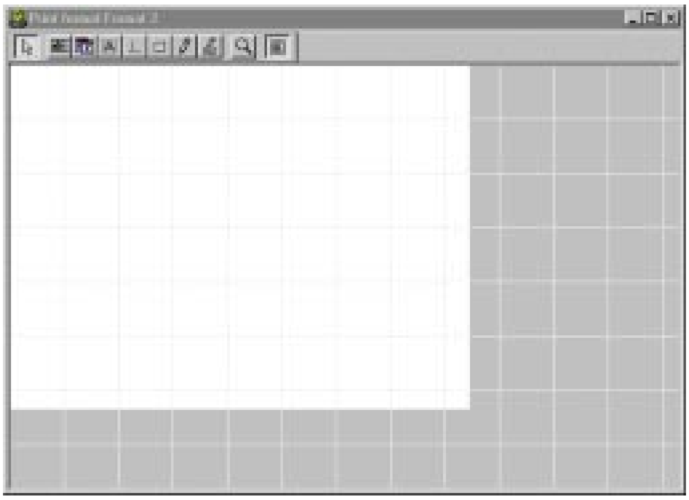
9.3.5.2 Module Mode

e mode the printing format is displayed outlining the it.

To activate this mode press the button .

Once pushed, the button will look like this and will remain this way till next time it is hit.
As long as it is down, the editor is in module mode.

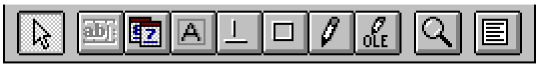
The window in module mode looks like this:



The module area is the zone in white. This is the work area. you should never invade the gray area, because objects put in it could overlap others in other modules.
In a module you can insert both static and dynamic objects (variables).
The tools available are the same you have in page mode, plus the possibility to add dynamic objects with the button

9.3.5.3 *Formats Editor Toolbar*

The tools available in the window allow to insert and modify objects in the work area, both in page and in module mode. They all work the same way. The only difference is the availability of the button to insert dynamic objects, not active in page mode.



TOOLBAR

The toolbar of this window contains all tools necessary to create a printing format in the selected mode. The tool to insert dynamic fields is not active in page mode.

This button allows to edit, select, re-size or move objects in the work area.

This button allows to insert dynamic objects (variables) in the work area. It is available only in module mode.

This button allows to insert system variables in the work area (such as date and time).

This button allows to enter static text in the work area.

This button allows to draw horizontal, vertical or oblique lines of different sizes and colors.

This button allows to insert rectangles of different sizes and colors.

This button allows to insert pictures of different formats in the work area.

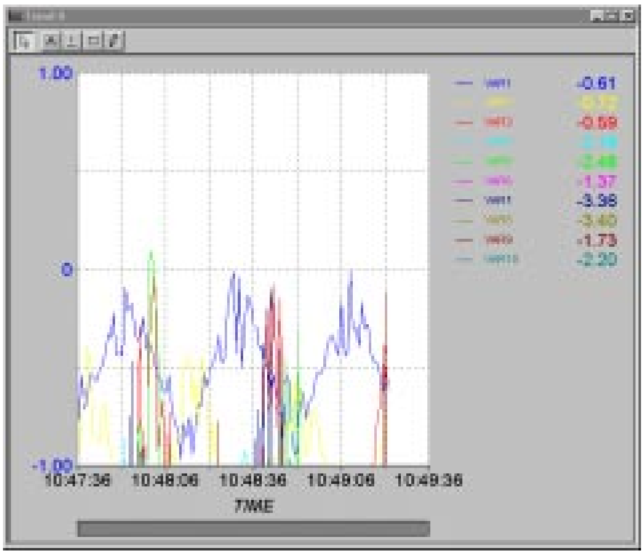
This button allows to insert OLE objects in the work area.

This button allows to zoom in an area of the work area.

With this button you can switch between page and module mode.

10. Line Trend

LOGOVIEW NT offers the possibility to build time pictures of up to 32 variables as line graphs. This type of picture is called Line Trend. Basically there are two kinds of line trends: real time and historical. The majority of configurations are common to the two, such as axes, captions and channels, while some characteristics are peculiar to one of them.



10.1 Real Time Line Trend

Real Time line trends allow to display the variation in time of one or more variables, by saving their values in the PC memory. While running, ***LOGOVIEW NT*** periodically samples the variables defined in a trend graph and saves them in a buffer. If the trend is being displayed, it will also update the plot. Since the buffer is located in the RAM memory of the PC, this type of plot is useful when displaying variables that do not have to be permanently saved. In any case the temporary buffer can save up to 32767 samples each variable.

10.1.1 Basic Characteristics

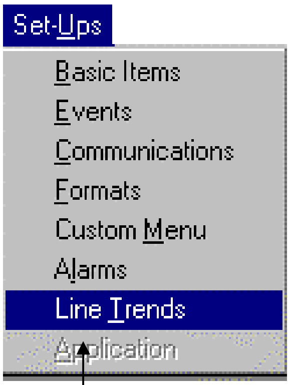
Each line trend is displayed inside its own window, and can be configured through a number of parameters:

- Sampling period (min. 1 second)
- Time span to display
- Time span to save in the temporary buffer
- Caption
- Current variables values
- Location and characteristics of the Y scale
- Characteristics of the X scale (time)
- Grid characteristics for both axes

For each trend you must define the list of variables the operator can display. Up to 32 can be displayed at the same time. ***LOGOVIEW NT*** requires a distinct configuration for the variables associated to the trend and for the 32 channels.; This way you can foresee a great

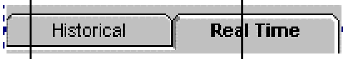
number of variables to be displayed in the trend (greater then 32), then select which ones you want to see. Limits and scaling are peculiar to each variable, regardless the way the channels are configured. The graphic appearance is configured for the channels.
At runtime the operator can modify scaling and displayed variables. These modifications can be under password. It is also possible to display the same variables in a numeric table.

10.1.2 Creating a Line Trend



To create a new line trend select ***Lin*** *List of configured trends* ***et-Ups*** menu or push the button to b e main toolbar.

This action will display the window ***Line Trend configuration***, which allows to create and configure both real time and historical line trends.

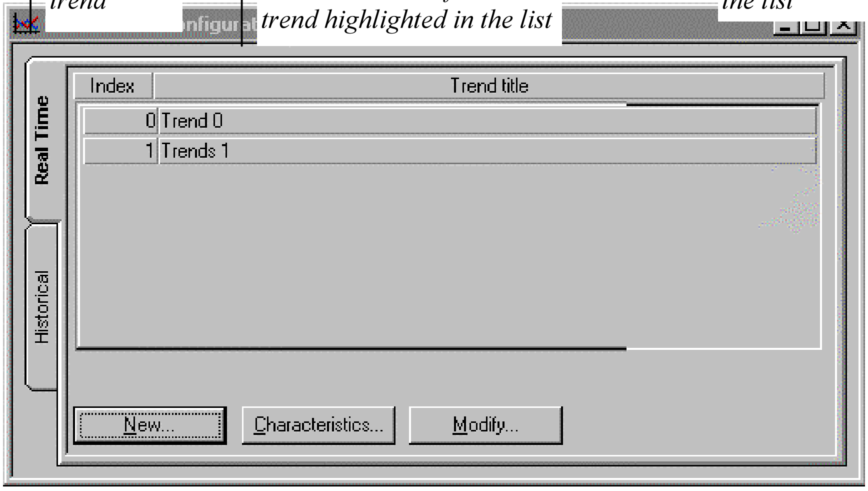


Section ***Real Time*** allows to create and configure real time trends. To create a new trend just click on button ***New***. This will open the configuration dia

To create a new line trend

To modify the characteristics of the line trend highlighted in the list

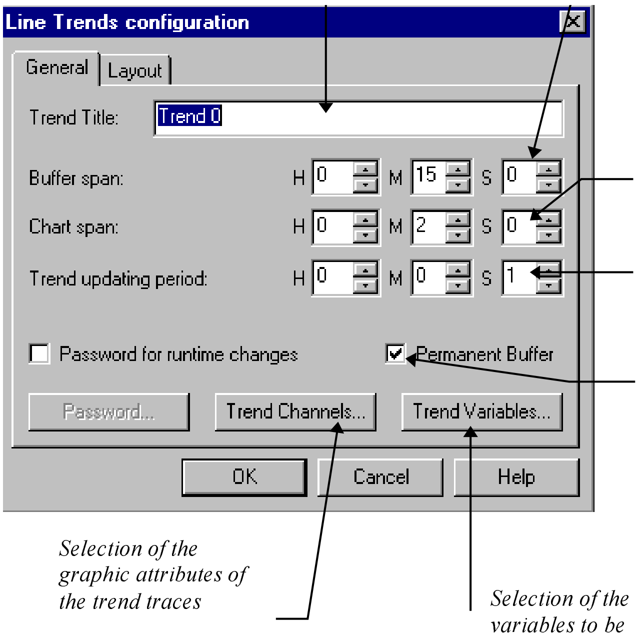
To modify the graphic aspect of the line trend highlighted in the list



Index	Trend title
0	Trend 0
1	Trends 1

10.1.3 Line Trend configuration

To configure a line trend you need to specify a number of parameters, defining both the working of it and its graphic appearance. The majority of parameters can be set by selecting the desired trend and then click on button ***Characteristics***. This button will open the window ***Line Trend configuration***, where you can enter all required parameters.



Line Trends configuration

General | Layout

Trend Title: Trend 0

	H	M	S
Buffer span:	0	15	0
Chart span:	0	2	0
Trend updating period:	0	0	1

□ Password for runtime changes

☑ Permanent Buffer

Time span of the line trend

Sampling rate

If the buffer is marked as permanent, the sampling process will occur even when the trend is not on display

Selection of the graphic attributes of the trend traces

Selection of the variables to be displayed in the trend

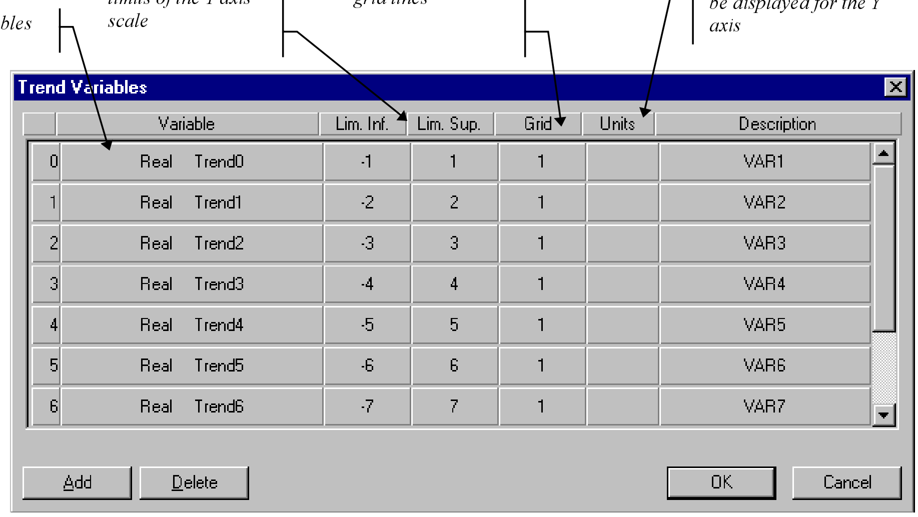
Before closing the window, please proceed to define the variables to be associated with the trend (button ***Trend Variables***) and the channels to be displayed (button ***Trend Channels***). The window for configuring variables allows to choose which variables can be utilized in the trend. For each of them you must define the min and max value, the grid amplitude for Y axis and the description to be displayed in the trend caption.

Variables

Upper and lower limits of the Y axis scale

Y axis step between grid lines

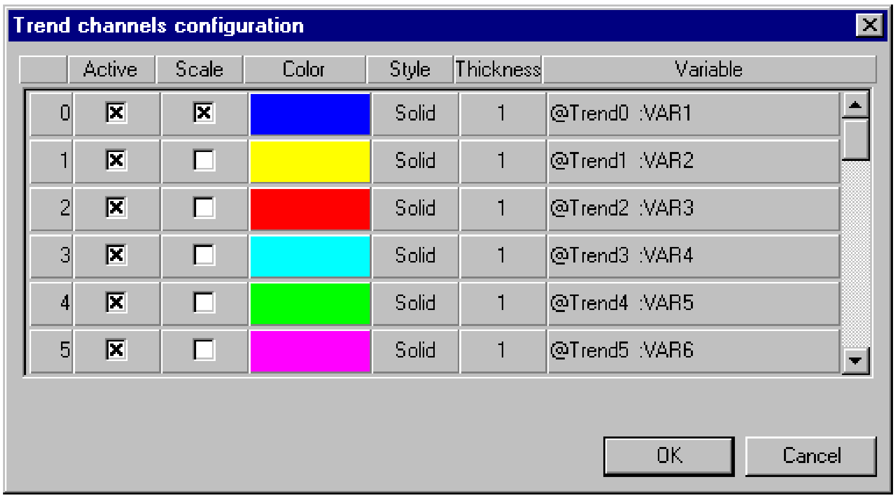
Units of measurement to be displayed for the Y axis



Trend Variables

	Variable		Lim. Inf.	Lim. Sup.	Grid	Units	Description
0	Real	Trend0	-1	1	1		VAR1
1	Real	Trend1	-2	2	1		VAR2
2	Real	Trend2	-3	3	1		VAR3
3	Real	Trend3	-4	4	1		VAR4
4	Real	Trend4	-5	5	1		VAR5
5	Real	Trend5	-6	6	1		VAR6
6	Real	Trend6	-7	7	1		VAR7

The window for channels configuration allows to specify the graphic characteristics of the 32 available channels, which channels are active, which one must be used as Y axis, and also which ones of the configured variables must be associated to each channel.

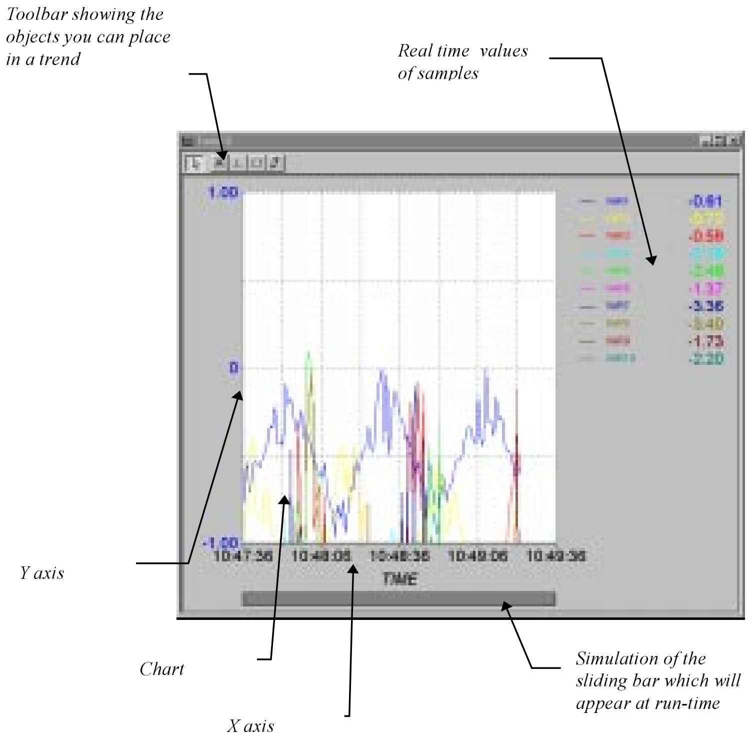


Trend channels configuration

	Active	Scale	Color	Style	Thickness	Variable
0	☒	☒	🟦	Solid	1	@Trend0 :VAR1
1	☒	☐	🟨	Solid	1	@Trend1 :VAR2
2	☒	☐	🟥	Solid	1	@Trend2 :VAR3
3	☒	☐	■	Solid	1	@Trend3 :VAR4
4	☒	☐	🟩	Solid	1	@Trend4 :VAR5
5	☒	☐	🟪	Solid	1	@Trend5 :VAR6

10.1.4 Line Trend window

By clicking on button ***Modify*** from the ***Line Trend configuration*** window you open another window which will display the simulation of the selected trend. In it it will be possible to modify the various trend characteristics with simple drag operations. It is also possible to insert graphic elements like lines, rectangles, texts and pictures directly inside the trend window.



Toolbar showing the objects you can place in a trend

Real time values of samples

Y axis

Chart

X axis

Simulation of the sliding bar which will appear at run-time

10.2 Historical Trends

The Historical line trends allow to display in the form of timed plot one or more variables stored in an historical archive or a Data Base.

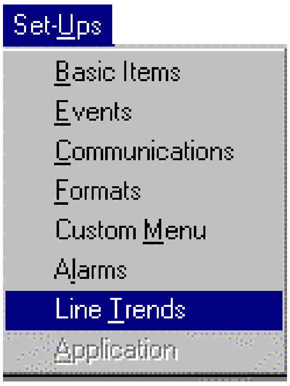
10.2.1 Basic Characteristics

Each line trend is displayed inside its own window, and can be configured through a number of parameters:

- Time interval to be displayed, or sampling period
- Archive or Data Base where the data are located
- Caption
- Location and characteristics of the Y scale
- Characteristics of the X scale (time)
- Grid characteristics for both axes

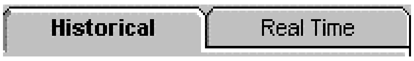
The list of variables the operator can choose from for that specific trend is read from the archive or Data Base configuration. You can display up to 32 variables ate the same time in a trend.
At runtime the operator can modify scaling and displayed variables. These modifications can be under password. It is also possible to display the same archived variables in a numeric table.

10.2.2 Creating Historical Line Trend



As for configuring real time trends, for historical trends you must select *Line Trend* from ***Set-Ups*** menu or push button 📈 to be found in the main toolbar.

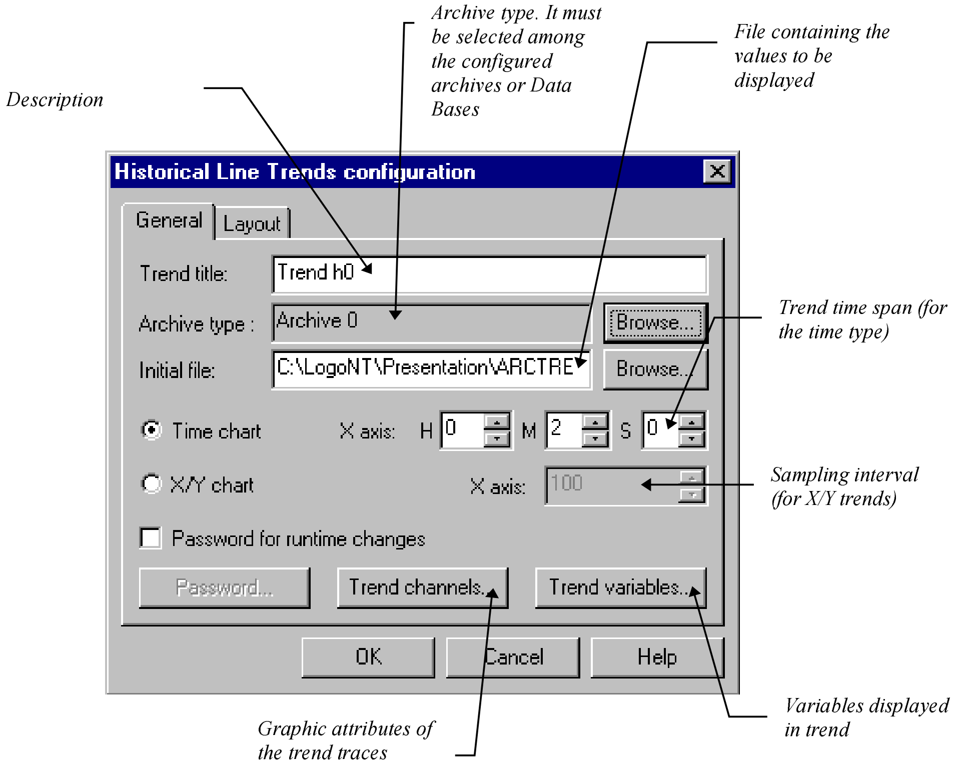
This action will display the window ***Line Trend configuration***, which allows to create and configure both real time and historical line trends.



Section ***Historical*** allows to create and configure historical trends. To create a new trend just click on button ***New***. This will open the configuration dialog.

10.2.3 Historical Line Trend configuration

To configure a line trend you need to specify a number of parameters, defining both the working of it and its graphic appearance. The majority of parameters can be set by selecting the desired trend and then click on button ***Characteristics***. This button will open the window ***Line Trend configuration***, where you can enter all required parameters.



Description

Archive type. It must be selected among the configured archives or Data Bases

File containing the values to be displayed

Trend time span (for the time type)

Sampling interval (for X/Y trends)

Variables displayed in trend

Graphic attributes of the trend traces

Historical Line Trends configuration

Trend title: Trend h0

Archive type : Archive 0

Initial file: C:\LogoNT\Presentation\ARCTRE

● Time chart X axis: H 0 M 2 S 0

○ X/Y chart X axis: 100

□ Password for runtime changes

Unlike real time trends, for historical trends you can not modify the variables, since they are read from an archive. To modify the variables you must change the archive configuration. In the configuration window you must nevertheless define both limits and caption.

The channel configuration is just like in the real time trends. Also the trend modifications and the addition of graphic elements is done as in the real time trends.

11. LOGOVIEW NT and the floor

The main duty of a supervisor application is the data collection from the floor. Obviously ***LOGOVIEW NT***, having been designed for this purpose, can offer a lot and allows many different possibilities.

The most important operations toward the floor are synchronous writing, synchronous reading and a-synchronous reading.

Synchronous writing fulfills the need to write variables into the floor devices. By synchronous reading instead variables can be read from external devices. Since these are synchronous operations, they make ***LOGOVIEW NT*** wait for the answer to the query, which slows down the whole communication. Therefore, this kind of operation must be performed only when required (recipes downloading, data read after operator command, etc.). The best way to read data from the field is instead the a-synchronous reading. This way, instead of periodic polling, the driver itself can communicate with ***LOGOVIEW NT*** only when a change has occurred, and store the updated values directly into their variables. Having only to update the changed variables, the operation becomes much quicker. This requires an intelligent driver, able to manage in an unsolicited manner all data inquiries.

11.1 Floor Devices

Before being able to communicate with the floor, ***LOGOVIEW NT*** needs to know the table of floor devices. In ***LOGOVIEW NT*** devices are identified by two names: the name of the device itself and the identifier of the station on which the device is located. The device name is symbolic, and is assigned to it depending on the kind of device; some devices can be named by the user. The station name is used when the device is connected to a remote station, linked to ***LOGOVIEW NT*** via network. In this case you must specify also the station name, to inform ***LOGOVIEW NT*** where the device is located. Devices can be of different kinds, such as another ***LOGOVIEW NT*** or LOGOVIEW 32 station.

The kind of link implemented in LOGOVIEW is very flexible, and allows a great number of different configurations, as depicted in the following Figures:

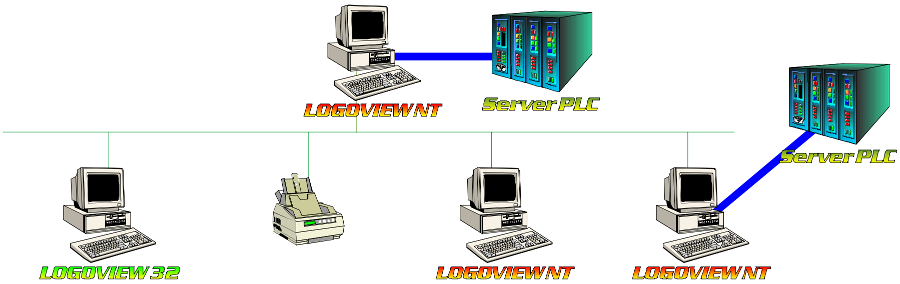


Figure 11-a: Traditional connection

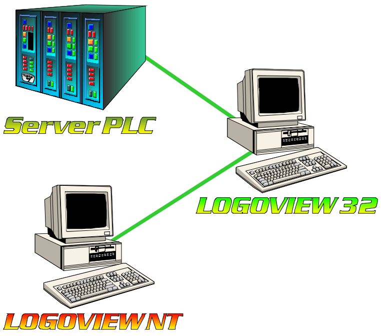


Figure 11-a: Bridge connection

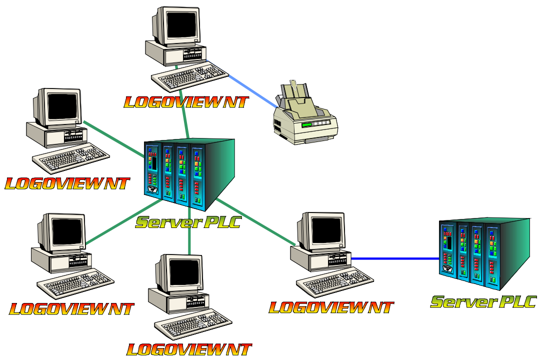


Figure 11-b: Mixed star connection

11.2 Communications types

With ***LOGOVIEW NT*** it is possible to establish two kinds of communications: SYNCHRONOUS, which are the traditional type, and A-SYNCHRONOUS.

11.2.1 Synchronous transactions

Synchronous transactions can occur from events only. The statements require to specify the device and all the parameters needed for reading or writing. The communication occurs as said above, with a wait for completion. Operations of this kind can be requested only toward devices defined in the drivers list. The operations possible are READ and WRITE.

```
EVENT 0:MAIN
VARIABLES NAME=V WORD=1 REAL=0
REM Initialization Event

.
.
.
.
DB_OPEN  "EQUIPES.DBF",1,:HEquipes,:RisDB,1
DB_GETREC  :HEquipes,:RisDB
DB_CLOSE  :HEquipes
WRITE_PLC  "LW_3964","",#Sync_1,5,1,2,:RisPlc
READ_PLC  "LW_3964","",#TURNO,1,9,2,:RisPlc
LET  #ActEquip = (#TURNO % 2) + 1
.
.
.
END
```

This example contains one READ and one WRITE operation on an external device.
The syntax of the two statements is similar:

WRITE_PLC ***"LW_3964","",#Sync_1,5,1,2, :RisPlc***

"LW_3964" :	*Device Name (must be present in the drivers list).*
"" :	*Remote Node Name. If local, input “”.*
#Sync_1 :	*First variable of the block to read or write (on LOGOVIEW NT).*
5 :	*Quantity of variables to read or write.*
1 :	*Variable Offset inside the device.*
2 :	*Flag. It has a double meaning: 1) Wait for reply or Do not Wait for reply, and also 2) Update always or Update variations only (valid for READ statement only)* *0 : Wait* *1 : Do not Wait*

:RisPlc : *Error Code (0 = OK).*

READ and WRITE statements access the device one at a time therefore, if more then one ***LOGOVIEW NT*** task requests communication, one waits for the other to be finished. In this case it is possible to release the second task by checking the Error Code: if 10 (ten) it means that the channel is busy, so the task could continue other operations until the channel is free. As said, it is also possible to specify for the READ statement if all data have to be read, or only the ones changed since the last READ.

This last option should be used with caution on network stations, since it may cause the loss of data on some stations.

11.2.2 A-synchronous transactions

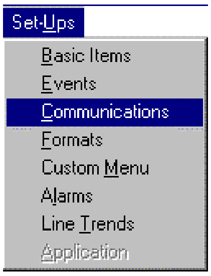
The only allowed a-synchronous operation is a READ.
It has been designed to perform cyclic reading, polling or timed.
These kinds of procedures normally lock the CPU, therefore slowing down the whole application. By using a-synchronous reading, ***LOGOVIEW NT*** will not leave any cyclic query to any device, but instead it will leave to the driver the duty to update any variable that may have changed.

11.3 Communications configuration environment

To set up communications a number of actions are necessary:

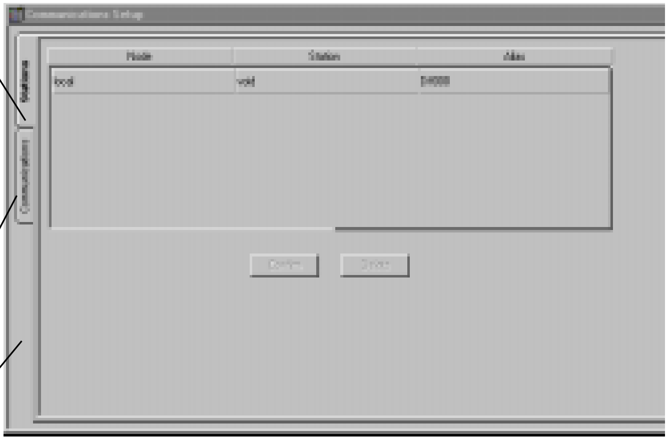
- Configuration analysis.
- Definition of the devices to communicate with.
- Definition and logical structuring of the data blocks to exchange with the field.

The system analysis depends on the kind of work to be done. Usually, in small plants, a simple connection is sufficient, including just one ***LOGOVIEW NT*** and one floor device. In large plants more complex architecture is needed. Check with more general documentation to get complete information for such cases. In this phase you also have to define the data map on the device and its name. It does not matter for now the device type, since during the whole development and testing phases the included PLC simulator will be used. This allows to simulate all operations normally performed by real devices. Only in the installation phase you will need to install the applicable device.



The other two steps will be performed directly using the ***LOGOVIEW NT*** environment.

To reach the communications editor, just press the button . This is available in the *Main Toolbar* . If you prefer use instead the applicable entry of the menu.



To setup the list of devices

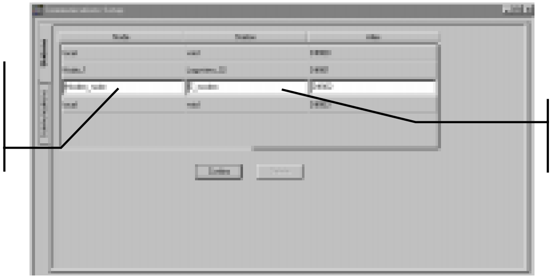
To configure the Operator's field commands

To configure the data blocks to be read from the field

This way the communications setup window will appear. Enter here all parameters defining the way ***LOGOVIEW NT*** will communicate with external devices and stations.
The window includes three configuration cards.

11.3.1 The Servers card

This card contains the list of items (servers) with which ***LOGOVIEW NT*** will have to connect itself. The list includes two columns. One shows the node names, the other the items registration names..



Name of the node on which the device is installed

Name of the field device

When the communication device resides in the same computer in which also ***LOGOVIEW NT*** resides, the *node* field will show the name *local.* This is the default name.

Important note. After any addition or change to list entries you must press the confirm button.

Once the list is completed, ***LOGOVIEW NT*** has the capability of establishing and maintaining connections with all listed active items, both the local and remote ones, detecting also when some device is closed or re-opened.

While entering names pay great care to enter them exactly in the same way as you did before, maybe elsewhere. Pay attention to capital and small letters, since they are considered different characters.

11.3.2 Data mapping

As said above, ***LOGOVIEW NT*** does not know which device type is connected. In frequent cases it might be another ***LOGOVIEW NT***. This is the reason why ***LOGOVIEW NT*** puts communication requests consisting of sequentially ordered variables, in the same order as the various variable types are handled by ***LOGOVIEW NT*** itself.

In practice the communication driver receives requests of this type:

- *send a block of 50 BYTE variables, starting from variable with offset 300, to be mapped starting from location with offset 1250.*

This must be understood by the driver as meaning:

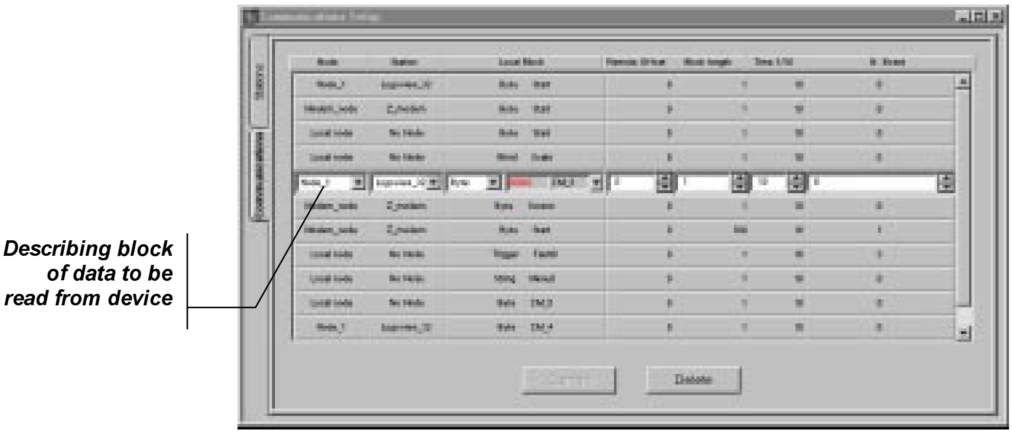
- *pick up from the internal map variables from offset 300 to offset 349 and send them to LOGOVIEW NT which, in turn, will store them starting from the BYTE variable with offset*

Evidently, if the driver to which ***LOGOVIEW NT*** is talking is another ***LOGOVIEW NT*** , this will simply retrieve the requested variables and send them to the requesting one. If instead the driver is interfaced to a PLC, it will be its task to perform requested data address conversion in order to match those of the connected device.

11.3.3 The communications card

Selecting the communications card you get access to the communication block planning table. These blocks, as said in the previous paragraph, are also used to define and program the connected device.

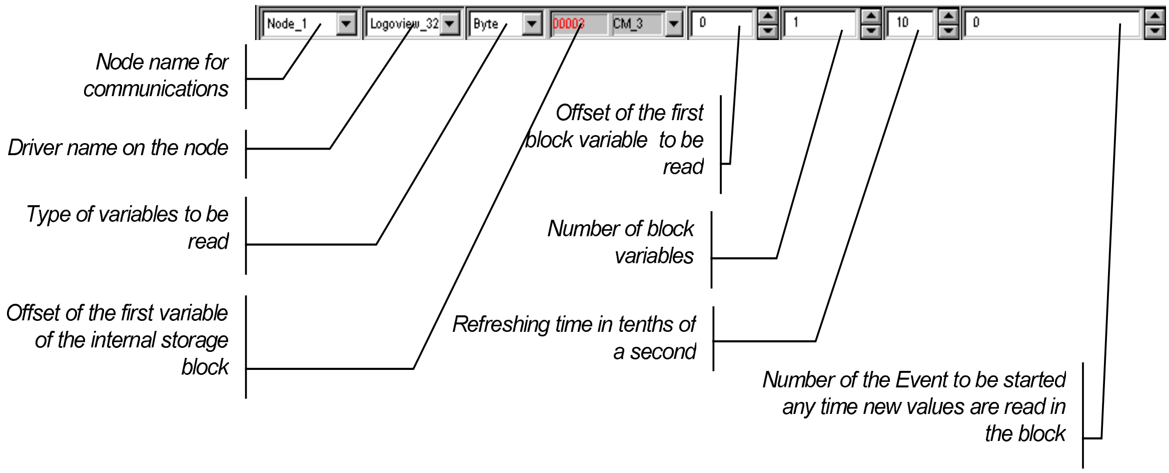
In order to do so, the driver must be programmed precisely according to the ***LOGOVIEW NT*** specifications. The input data table programming occurs by filling the following list:



Describing block of data to be read from device

During development you must configure all floor connection tables, by describing the various data blocks to be controlled by the driver. When connection is established, ***LOGOVIEW NT*** transfers all tables to the destination devices. Upon receiving tables, it is the driver’s task to arrange the reading cycle and, above all, prepare itself to send the supervisor only those variables which actually had a change of value since the last reading.

The description block contains the following fields:



Node_1 | Logoview_32 | Byte | 00003 | CM_3 | 0 | 1 | 10 | 0

Node name for communications
Driver name on the node
Type of variables to be read
Offset of the first variable of the internal storage block
Offset of the first block variable to be read
Number of block variables
Refreshing time in tenths of a second
Number of the Event to be started any time new values are read in the block

As it can by observed in the picture, the various blocks get created on the basis of all entered parameters. Only the event number is not transferred to the driver among all other pieces of data.

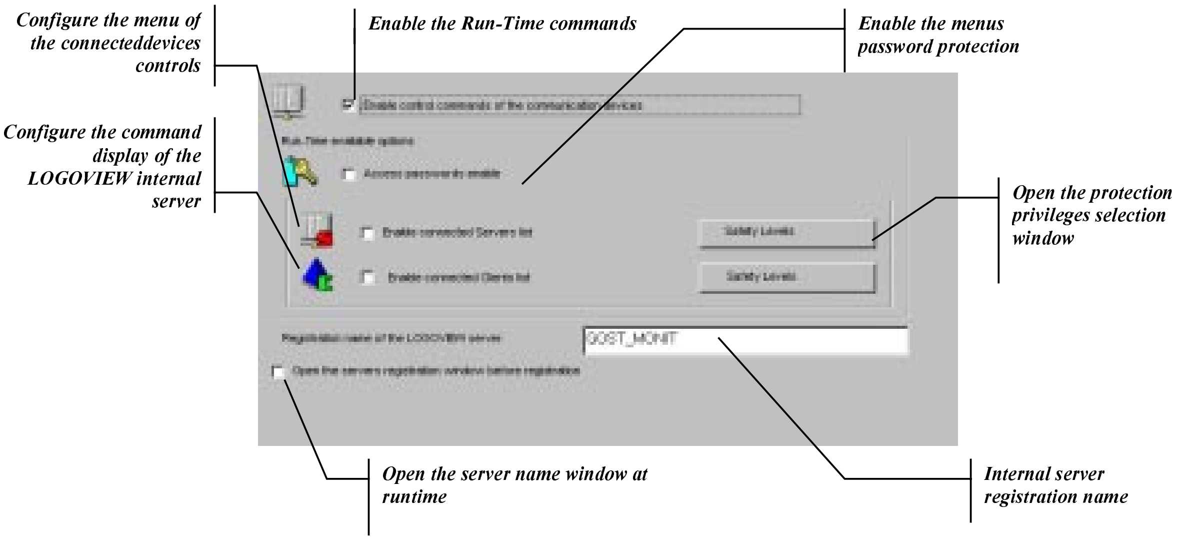
Starting from these, the driver shall create a table associating the various logic points controlled by the floor device, then check periodically (as programmed) their values and, if value changes occurred, send data to the supervisor.

11.3.4 The controls card

LOGOVIEW NT is also capable of remotely control the connected device. This is achieved by getting access to the command configuration card, which, in turn, allows configuring a pull down menu to appear at runtime, allowing the Operator to control the device from ***LOGOVIEW NT*** .

This capability is useful when working in a network distributed system, so that you can control a device residing on a certain node from another node.

The card shown below allows configuring the following controls:



☑ Enable control commands of the communication devices

Run-Time available options

□ Access passwords enable

□ Enable connected Servers list

□ Enable connected Clients list

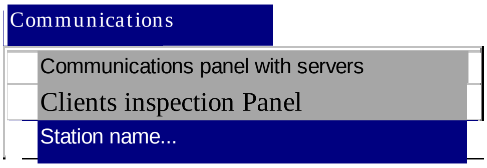
Registration name of the LOGOVIEW server: GOST_MONIT

□ Open the servers registration window before registration

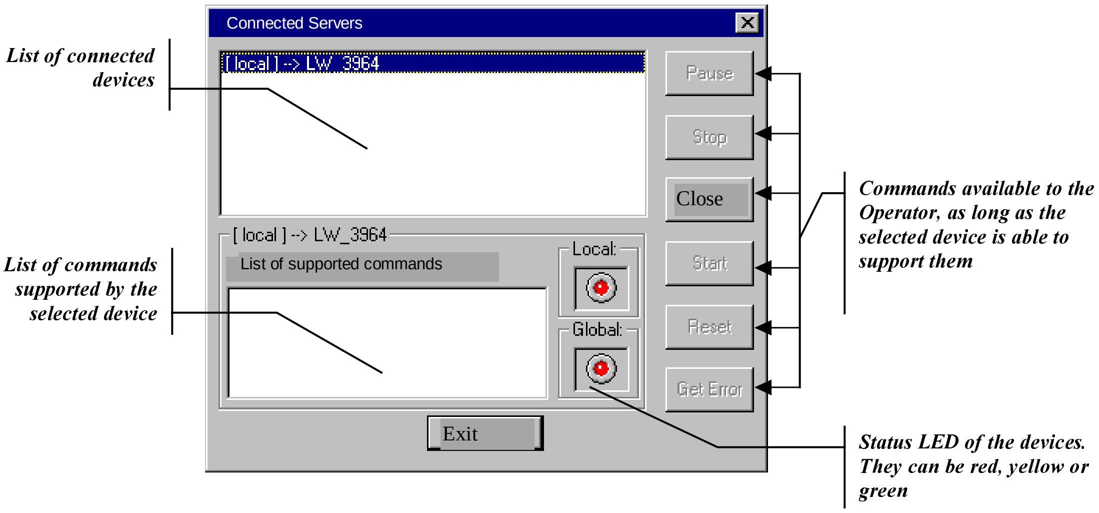
Safety Levels
Safety Levels
Configure the menu of the connecteddevices controls
Configure the command display of the LOGOVIEW internal server
Enable the Run-Time commands
Enable the menus password protection
Open the protection privileges selection window
Open the server name window at runtime
Internal server registration name

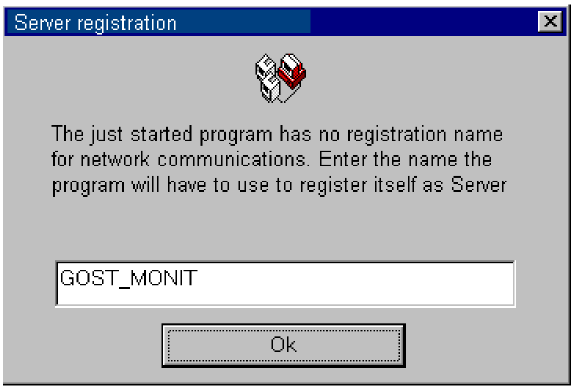
Once these controls have been configured as needed, upon starting the runtime a pulldown menu labeled *Communications* will appear in the main menu bar. The entries you can find in such a menu are the following:

11.3.4.1 *Communications panel with servers*



This menu entry opens the control panel of the devices connected to ***LOGOVIEW NT.*** By means of this panel you can read each device status, the list of controls supported by each device and, on this basis, provide some commands. The command panel is shown below. To get complete documentation on these controls and the panel itself you are invited to consult the full documentation available on CD-ROM.


List of connected devices
List of commands supported by the selected device
Connected Servers
[local] --> LW_3964
[local] --> LW_3964
List of supported commands
Local:
Global:
Exit
Pause
Stop
Close
Start
Reset
Get Error
Commands available to the Operator, as long as the selected device is able to support them
Status LED of the devices. They can be red, yellow or green



The just started program has no registration name for network communications. Enter the name the program will have to use to register itself as Server

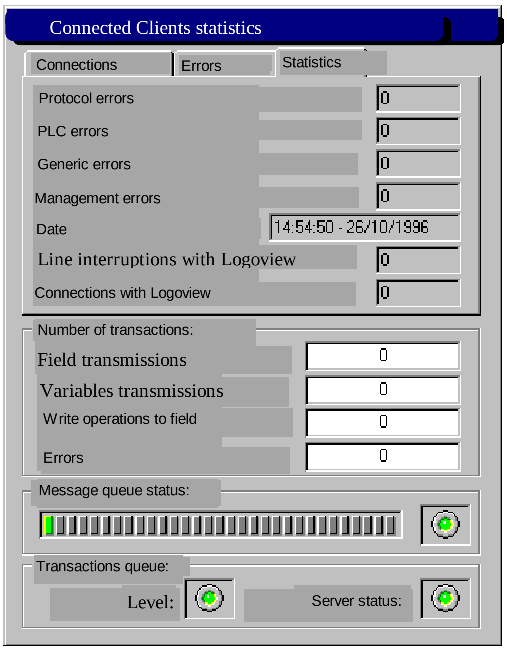
GOST_MONIT

11.3.4.2 *Station name...*

Last, you can read the registration name of the ***LOGOVIEW NT*** internal server. Selecting such entry, a window will appear containing this name. You can read it but you can not change it. This entry in fact is the only one not requesting a password.

11.3.4.3 *Clients inspection panel*

This entry allows opening the panel providing access to the ***LOGOVIEW NT*** internal server. In fact, LOGOVIEW can connect itself not only to external devices, but also can be seen by another LOGOVIEW as an external device. This way applications can be developed sharing the same data (as shown in the example configurations at the beginning of this chapter). Using this panel you can inspect possible server working errors as well as the load status of the communication queues. Also in this case, to get complete reference on the control and the panel itself, we suggest you consult the documentation available on CD-ROM. An instance of the panel is shown below.



Protocol errors 0
PLC errors 0
Generic errors 0
Management errors 0
Date 14:54:50 - 26/10/1996
Line interruptions with Logoview 0
Connections with Logoview 0

Number of transactions:
Field transmissions 0
Variables transmissions 0
Write operations to field 0
Errors 0

Message queue status:

Transactions queue:
Level:
Server status:

PART III

Runtime Environment

12. Runtime Environment

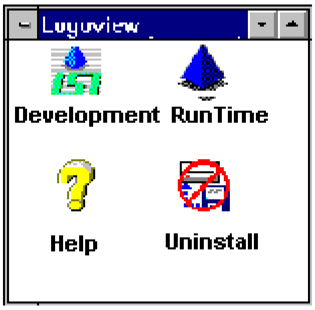
The runtime environment allows to execute the ***LOGOVIEW NT*** application. It contains all tools required by the Operator to interact with all plant supervision objects.

12.1 Application starting

You start the ***LOGOVIEW NT*** runtime environment as any other *WINDOWS* application: just double click on the application's icon.

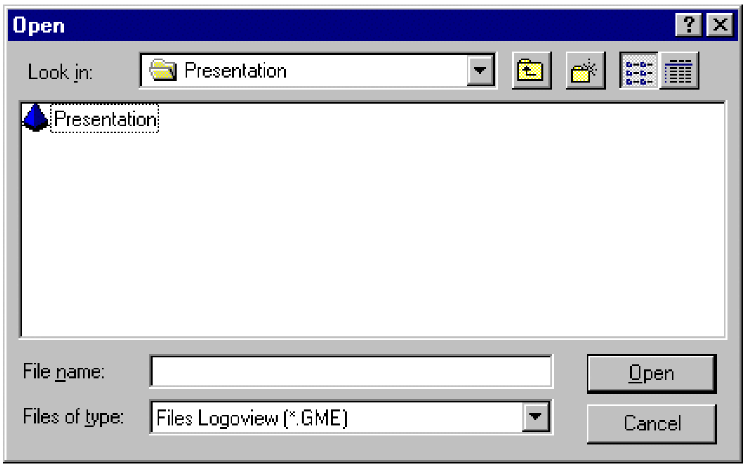
Icon for Execution
double click to start

The application icon is present in the ***LOGOVIEW NT*** group created by the installation procedure. The group can have an appearance different from the one shown below, depending on the operating system being used.



Users not possessing any knowledge of WINDOWS concepts and new to such operating system may look for a summary description of all its main features in the Microsoft WINDOWS User Manual, or in the online lessons.

Once started, the runtime environment will immediately ask for the application's name by displaying the file window.



Open

Look in: Presentation

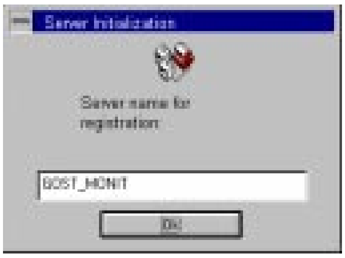
Presentation

File name:

Files of type: Files Logoview (*.GME)

In addition you can run an application directly from the development environment by pressing the dedicated button in the main toolbar.

12.2 Registering the application



Server Initialization

Server name for registration

GOST_MONIT

As previously said, ***LOGOVIEW NT*** operates in client/server systems by means of the *NET_DDE* protocol. Consequently it can be seen as user of services offered by communication drivers as well as supplier of such services to other ***LOGOVIEW NT*** applications. For this reason ***LOGOVIEW NT*** will immediately display just after starting the name of the station used for registering purposes, in the window shown aside.

LOGOVIEW NT **assumes** GOST_MONIT as default name. However, should the name be different, you can obviously change it in the dedicated white input box.

12.3 The Environment

The runtime environment will soon display the screen layout conceived as the first screen layout of the application sequence.

12.3.1 Main Toolbar

The runtime environment has a toolbar containing all main controls available to the Operator.

Button action: print out one of the *Print Formats* configured in the development phase.

Button action: open the password windows.

Button action: acknowledge the individual alarm presently selected in the alarms window.

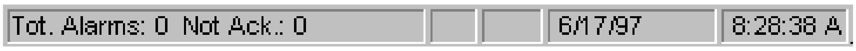
Button action: acknowledge all alarms displayed in the alarms window. This action applies only to the alarms actually in front of the operator. Alarms displayed in hidden parts of the window are not acknowledged.

Button action: provide general information regarding the package itself, version, authors and so on.

Button action: provide help on each individual control available in the runtime environment.

12.3.2 The Status Bar

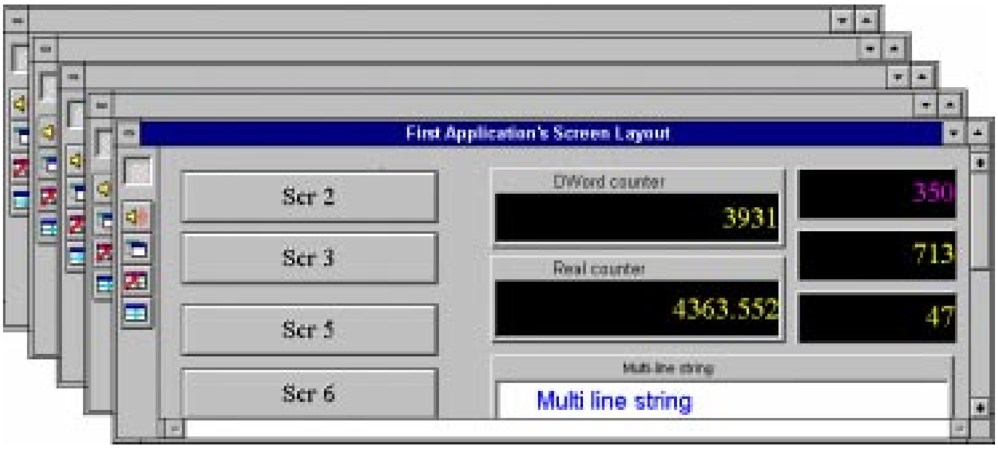
In the lower part of the application window a bar is displayed providing useful information such as total number of alarms, number of not acknowledged alarms, time, date and so on.



Tot. Alarms: 0 Not Ack.: 0 | | | 6/17/97 | 8:28:38 A

12.4 The environment windows

The runtime environment is mainly composed of the screen layout windows. These are the basic tools allowing the Operator to interact with the plant. ***LOGOVIEW NT*** is capable of concurrently displaying up to five screen layouts contained in as many windows.



To create a new environment window, just press the button available in the window left side. Upon this, a new window is created containing the same present screen layout, to begin with. Then the operator has an additional navigation point to reach all other screen layouts in the application's tree.

Keeping several windows open on screen will slow down the application. It is recommended to keep open at the same time only the windows strictly required to perform a given task..

12.4.1 Environment windows Toolbar

Each environment window provides a toolbar in the window left side. This bar contains a few controls for Operator commodity.

ENVIRONMENT TOOLBAR

Button action: display and hide the alarms window present in the lower part of the environment.

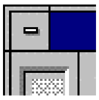
Button action: creates a new environment window, containing the same screen layout presently displayed. Then the operator has an additional navigation point to reach all other screen layouts in the application's tree.

Button action: split the environment window in four parts. This allows looking at the same time to window parts that otherwise could not be seen near each other.

Button action: cancel the window splitting, restore the window full view.

This flag can provide many features, according to the way it has been programmed during development. For more information see the application manual.

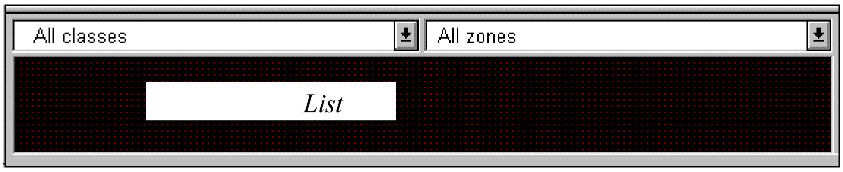
12.4.2 Closing environment windows



To close a window created by the button you need just to double click on the system menu shown to the left, available in the window upper part. The main window, (the one from which copies have been obtained) constitutes the running application. Therefore it can not be closed.

12.5 The alarms window

The most important window present in the runtime environment, besides the main window, is the one displaying the application's alarms, separately for each class and zone. This window is usually present in the lower part of the environment.



It contains all active alarms, their states and corresponding messages. Each alarm in this window can be individually acknowledged by the Operator by means of the button

In addition all alarms present on screen can be acknowledged by means of the button