Table of Contents

11. HIGH LEVEL PROGRAM INTERFACE

11.1 Overview of Programming Capabilities

- This chapter describes the various facilities available to you from a programming point of view, unlike the previous chapters, which cover fill-in-the-blank type configuration tasks.
- This chapter enables you to realize the full power and flexibility of the system by describing the available programming facilities.
- The table "Summary of Programming Capabilities" on the next page shows the full range of programming facilities available in ***MacroView***.
- Take a moment to study this table. The tools range from simple expressions in graphics and alarms to off-the-shelf interfaces such as dBase and Lotus to high level languages such as C and Fortran.
- In particular, please note the columns marked Capabilities, Limitations and Applicability.
- ***For most Process Control and SCADA applications, you will find that the meta script programming language is the best option. This is because the language was designed specifically for this purpose***.
- Also, please note that some of the facilities are optional.
- The optional facilities are only described briefly here but are described in more detail in their own documents.

Warning on Future Support Discontinuation

Because of the power and ease of use of the meta script language, it is probable that support for the following tools will not be continued indefinitely.

(i) exproc* The expression Processor,

(ii) quickrep* The quick Report Generator and

(iii) shell scripts. (Although these will still be supported for system administration applications.)

*These tools are not supported by NT.

In various test cases, we have found the meta script language far simpler and easier to use than the tools listed above. We therefore recommend that you avoid these tools for all future application implementations.

11.2 Summary of Programming Capabilities

Table 2: Programming Options

Programming Tools	Description	Capabilities	Limitations	Applicability	Example	Doc#: (Chapter)
metascripts	SQL like programming language suitable in process control, graphics and database applications.	Broad range of capabilities	Speed in some critical numeric applications.	Database, Control, Graphics etc.	`IF (L109.ALM>90)` `LET L109.MV=0;`	meta script chapter

Expression (in Alarms)	Expressions in the alarm database are evaluated to set or clear the alarm.	Logical
		Arithmetic
		Functions
			128 character expressions, (1 line)	Alarm Decisions	`($)LS="MAN"`
					`.AND.`
					`($).PV>($).PH`
						Alarms

Expression (in Graphics)	Expressions typed into the AutoCAD or AutoSketch file are evaluated when the graphic is called up.	Logical
		Arithmetic
		Functions
			128 character expressions, (1 line)	Graphic Calculations
				Color and shape change.
					`#C IF(PV`
					`(P72RUN)=1,11,10`
					`)`
						Graphics

Shell Commands	Shell commands that may be typed in interactively or used in script (Batch) files in combination with other shell facilities.	Get, Set values Alarm Messages Shell functions	Real-time data collection and setting.	Alarm Actions System control Advanced control	`Getval PV_LC110`	Programs
Spread-sheet Interface MVDDE Package	Collects real-time and historical data from process and places it in the spreadsheet. Also sends data from the spreadsheet to the process.	Full spreadsheet functionality Real-time Data Historical Data	Not simple to do on-line logs.	Quick generation of Reports/What ifs/ Analysis Off-line reports, Control	`=MVDDE|VECTOR!|PV(LC110)`	P-SSI

dBASE interface	Collects real time and historical data from process under dBASE program control and also writes data back to the process	Full dBASE functionality
		Graphic interface to dBASE entities,
		Real-time Data
		Historical Data
			User must know dBASE	On-line logs
				Reports
				Control
					`PV_FIC100 = 999`
					`Save to FILE`
					`run db4mem FILE`
					`restore from FILE`
					`@ 10,18 say PV_FIC100`
						P-DBI

Programming Tools	Description	Capabilities	Limitations	Applicability	Example	Doc#: (Chapter)
C Interface	Collects real time and historical data under C control and may write values back to the process.	Fast, simple programs Real-time Data Historical Data	User must know C	Simulations Advanced Control Reconciliation's Special	`fc_getdbl(n_values,dblarray)`	P-FORC
Fortran interface	Collects real time and historical data under Fortran control and may write values back to the process.	Fast, simple programs Real-time Data Historical Data	User must know Fortran	Simulations Advanced Control Reconciliation's Special Functions	`call fc_getdbl(n_values,dblarray)`	P-FORC
C Driver Kit	A library of C runtime routines, documentation and diagnostic routines that enable a user to write a compatible driver.	Professional, Compatible drivers	User must know C	Special purpose drivers.	`PLC_update()`	P-CDRV

Notes on choice of programming facilities

There are a large number of factors to consider when choosing a programming tool. E.g. for this application do I use Lotus or should I write the application in C? In addition to the factors listed in the table, you should also consider other factors such as:

(i) your own preference and experience in using the tool,

(ii) the amount of computer resources the tool will use,

(iii) the types of tool already used by your corporation and

(iv) the maintenance of the resulting software.

A **very** rough guide as to the suitability of the tools to the application is shown below.

Under no circumstances should this be used for anything other than a rough starting point for your choice.

For example, it is quite possible to create reports using shell scripts or to create on-line logs using Lotus.

Table 2: Rough guide for suitability

Application	Tool	Conditions
Graphics, Database applications, Reports, on-line logs, simulations	meta scripts	Providing there is no requirement for very fast applications.
Personal Computer applications	DDE Interface	Need to be linked via a ***MacroView*** DDE server.
Fast Applications	C/Fortran	For large, fast or complex applications
System functions	meta scripts using shell scripts	
Drivers	C Driver kit	

If you have doubts as to the best tool to use for your applications may we suggest you contact your nearest distributor who will happily provide you with assistance based on experience.

Format of this Chapter

As mentioned earlier, some of the programming tools in the summary page are optional and are described in detail in their own documentation. In particular, please see the chapter on meta scripts.

This document does however give a short description of these tools in the section on Optional Facilities.

The chapter gives a detailed description of the following subjects:

(i) Expressions and

(ii) Shell commands

11.3 Expressions

All about Expressions

MacroView expressions are used in numerous parts of the system as a general purpose tool to perform calculations, do comparisons, make logical decisions, control values, etc. As can be seen from the section Overview of Programming Capabilities in this chapter, expressions are used:

(i) In alarms (generally to make intelligent decisions as to whether an alarm condition exists);

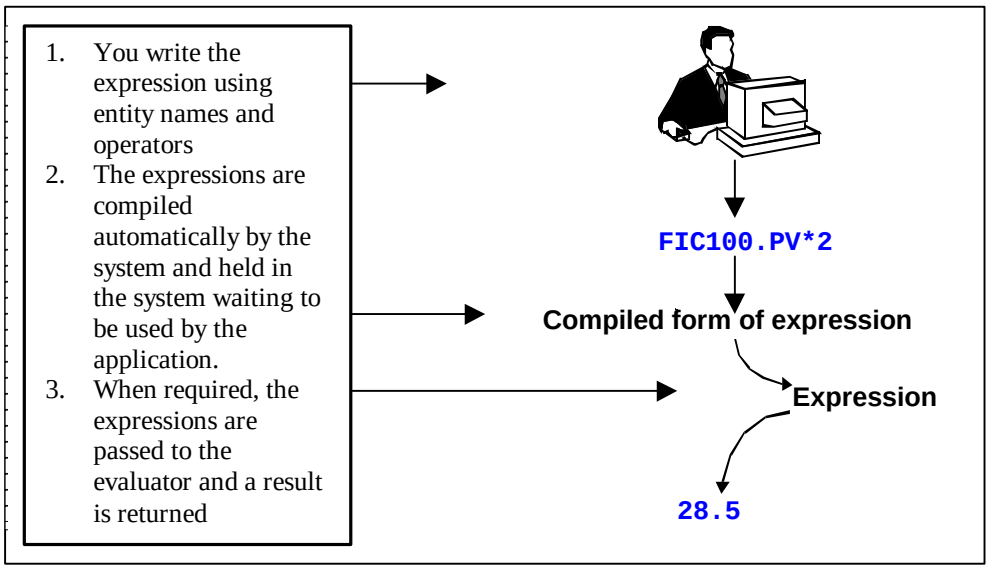
(ii) In graphics (used extensively to decide on colours, shapes, scales etc. and to perform calculations based on real-time values); and

(iii) In other areas such as the quality mask of the historical package.

Because of the extensive use of expressions in ***MacroView***, it is most important that you have a good practical working knowledge of expressions and their use.

How expressions are evaluated

The diagram shows how the expressions are used by the system to return results to the application.



Some characteristics of Expressions

As the following sections show, expressions are both simple and powerful and contain a large number of facilities. In all cases,

(i) The expression length must not exceed 128 characters (or 80 characters if running the DOS-based terminal package).

(ii) The result of an expression is returned to the application. For example, if the expression `FIC100.PV*2` is evaluated, it returns the value `28.4` (say) to the graphics presentation application where it is displayed on the screen.

(iii) You may use blank spaces within expressions (with AND, OR and NOT operators, you MUST have a space before and after the operators).

(iv) You may use brackets to show which part of the expressions should be evaluated first.

The following pages show other features of the expressions e.g. constants, operators, functions etc.

Constants

You may use constants in expressions. The table shows the various forms of constants and some examples.

Table 3: Types of Constants

Type of Constant	Examples
Numeric	`57.3` `11`
Character	`"MAN"` `"AUTO"`
Logical	`.TRUE. or .T.` `.FALSE. or .F.`

True and False

For the purposes of Boolean Logic and comparisons, any value in an expression that is not equal to zero is considered TRUE.

Any value equal to zero is considered FALSE.

When a comparison is carried out, if the comparison is true, a 1 is returned. If false, a 0 is returned.

E.g. `LIC100.PV > 14` returns 1 if true,
returns 0 if not true

As shown above, you may use `.TRUE.` or `.T.` and `.FALSE`. or `.F.` in expressions.

In the table below, we have used the following values in the examples:

```
LIC100.PV = 14.2      LIC100.LS = "AUT,"      FIC100.PV = 34.6
PMP43.PV = 1          PMP44.PV = 0
```

Table 4: Operators (See the example values below)

Type	Symbol	`Example`	Returns	Precedence	Comment
Exponentiation	** or ^	`LIC100.PV ^2`	201.64	7	
Divide	/	`LIC100.PV /2`	7.1	6	
Multiply	*	`LIC100.PV *2`	28.4	6	

Type	Symbol	Example	Returns	Precedence	Comment
Remainder	%	`LIC100.PV %2`	0.1	6	
Subtract	-	`FIC100.PV - LIC100.PV`	4.6	5	
Add	+	`FIC100.PV + LIC100.PV`	48.8	5	
Concatenate I	+	`"STATUS=" + LIC100.LS`	"STATUS = AUT"	5	Combines two character values
Concatenate II	-	`"ALM" - "LM"`	“A”	5	"Subtracts" two character values
Equal to	=	`PMP43.PV = 1`	1	4	A true expression returns a 1, a false expression return a 0
Not equal to	<> or #	`LIC100.PV #14`	1	4	
Less Than	<	`LIC100.PV <14`	0	4	
Greater Than	>	`LIC100.PV >14`	1	4	
Less Than or Equal To	<=	`LIC100.PV<= 12`	1	4	
Greater Than or Equal to	>=	`LIC100.PV>= 12`	0	4	
Contain	$	`"CD" $ "ABCD"`	1	4	Returns true if first string is contained in second string.
AND	.AND.	`PMP43.PV .AND. PMP44.PV`	0	3	You **must** have a space in front of and following the .AND.
OR	.OR.	`PMP43.PV .OR. PMP44.PV`	1	2	You **must** have a space in front of and following the .OR.
NOT	.NOT.	`.NOT. PMP43.PV`	0	1	You **must** have a space in front of and following the .NOT.

Table 4: Functions

Function Description	Example	Returns	Comment

`IF(condition)`
`{`
`RETURN True;`
`}`
`ELSE`
`{`
`RETURN False;`
`}`
	`IF(LIC100.PV>80)`
	`{`
	`  RETURN "HI"`
	`}`
	`ELSE`
	`{`
	`  RETURN "Norm";`
	`}`
		"Norm"	The IF statement is looked at more closely in its' own section below.

Function Description	Example	Returns	Comment

	IF(LIC100.LS="AUT")
	{
	RETURN 3;
	}
	ELSE
	{
	RETURN 4;
	}
		3	

STR(value, size, places) Returns string result of numeric value.	STR (LIC100.PV, 5, 2)	"14.20"	Useful to specify the number of decimal places.
SUBSTR(src, offset, num) Extracts a substring from another string.	SUBSTR(LIC100.LS, 1, 2)	"AU"	
VAL (string) Converts a string into a numeric value.	VAL("2.3")	2.3	
ABS(value) Returns the absolute value of the numeric value	ABS(LIC100.PV-FIC100.PV)	20.4	
INT(value) Returns the integer value of the numeric value	INT(LIC100.PV)	14	
RAND (range) Generates a random value between 0 and range.	RAND(99.9)	80.1	

Table 5: Date and Time Functions

Function Description	Example	Returns	Comment
DATE() Returns a string containing the current date	DATE() SUBSTR(DATE(), 6, 5)	"1992/07/09" “07/09”	Format is YYYY/MM/DD
TIME () Returns a string containing the current time.	TIME() SUBSTR(TIME(), 1, 5)	"09:34:44" "09:34"	Format is HH:MM:SS
HOUR() **MINUTE()** **SECOND()** Return a value with the current hour, minutes or second.	HOUR() MINUTE() SECOND()	9 34 44	

Table 6: Graphic specific functions.

Function Description	Example	Returns	Comment

FLASH(value)
Returns the colour value followed black cyclically.
	`IF(LIC100.PV>10)`
	`{`
	`  RETURN FLASH(Red);`
	`}`
	`ELSE`
	`{`
	`  RETURN Green;`
	`}`
		Returns
		Red
		Then
		Black
		Else
		Green
			Used mainly to flash graphics from black to a colour.

SQWAVE(x,y) Returns a 1 for x seconds and a 0 for y seconds repeatedly.	`2 = SQWAVE (3, 1.5)`	1 for 3 seconds 0 for 1.5 seconds 1 for 3 seconds	

Table 7: Trigonometric, Hyperbolic, and Logarithmic Functions

Function Description	Example	Returns	Comment
SINE	`SIN(30)`	.5	Angle in Degrees
ARCSINE	`ASIN(0.5)`	30	
COSINE	`COS(30)`	.87	
ARCCOSINE	`ACOS(0.87)`	30	
TANGENT	`TAN(30)`	.58	
ARC TANGENT	`ATAN(0.58)`	30	

Using Entities in Expressions

You may use entity names freely in expressions in the form of entity.attribute, for example:

```
FIC100.PV
```

In the case of dBase type entities, you may specify the offset from the current record number using the square brackets. I.e.

```
BATCH.PH[2]
```

This would return twice the PH field from the third last record of the dBase entity BATCH.

The IF Function

The IF function plays a major role in expressions so it is important for you to be comfortable with its' use.

The general form of the statement is:

```
IF (condition)
 {
   RETURN True;          Returns this value if the condition is TRUE
 }
ELSE
 {
   RETURN False;         Returns this value if the condition is FALSE
 }
```

As an example:

```
IF (PMP43.PV = 1)
 {
   RETURN "RUNNING";
 }
ELSE
 {
   RETURN "STOPPED";
 }
```

IF Expressions may be nested, i.e.:

```
IF (condition)
 {
   RETURN True1;
 }
ELSE IF(Condition2)
 {
   RETURN True2;
 }
ELSE
 {
   RETURN False2;
 }
```

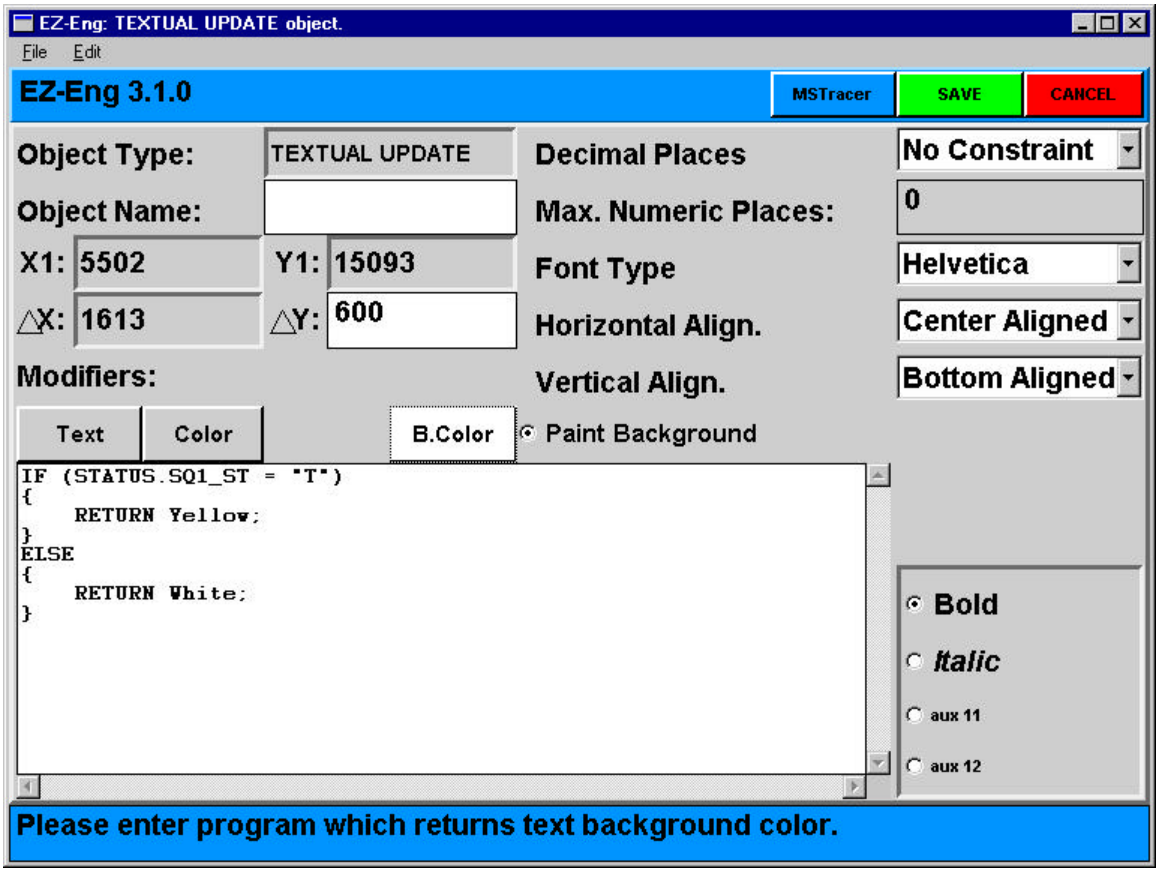
For example:

```
IF (PMP43.PV=1)
 {
   RETURN "RUNNING";
 }
```

```
ELSE IF(PMP43.PV=0)
 {
   RETURN "STOPPED";
 }
ELSE
 {
   RETURN "ERROR";
 }
```

Expressions in Graphics

Expressions within graphics are created via the EZ-Eng graphical interface. All graphical objects within MacroView use expressions in the Metascript Language format.



Expression in Alarms

Expressions may be used in the Alarm expression field.

For example, if the alarm type is `EXPRN` and the expression field is:

```
LS($) = "MAN" .AND. PV($) >PH($)
```

The alarm manager triggers an alarm if the status is MAN and the process variable is greater than the high limit.

For more information, see the Alarms Chapter in this manual.

11.4 Shell Commands

All about Shell Commands

Please see the warning on discontinuation on page 1 of this chapter.

Shell commands are commands that can be typed in directly at the UNIX prompt and that will receive an immediate response (Just like `cd` for change directory, you can type `getval` to get a value from the ***MacroView*** system). These commands may also be written into script or batch programs to create powerful reporting, advanced control, alarm handling etc. programs.

The ***MacroView*** system offers a number of shell commands that interact directly with the ***MacroView*** system. These include:

getval - This command enables you to get a value from the ***MacroView*** system.

e.g. `getval PV_LC110 83.7`

setval - This command enables you to set a value in the ***MacroView*** system providing you have the correct security access level.

e.g. `setval SV_LC110 83.6`

sendmsg - This command enables you to send a message to any of the alarm or message screens in the color of your choice.

e.g. `sendmsg 0 3 "Critical temperature on HTR 7"`

This section shows you how to use these simple commands and combine them with some of the rich UNIX facilities to create powerful applications.

Advantages of Shell commands

There are a number of reasons you should seriously consider using shells in the process control industry.

(i) Shell commands are simple. There is no large learning curve required to start writing simple and useful shell scripts.

(ii) There is a rich collection of shell facilities available that support such things as internal and global variables, if-then-else constructs, pipes, conditional evaluations

(iii) No compilations are required. The shell commands are executed immediately and you can type the commands directly at the UNIX/NT prompt and debug the commands interactively.

(iv) You can use all the facilities available in UNIX (not in NT), for example, in a script you may use the floating point calculator `bc` or the stream editor `sed` or a pattern scanning facility like `awk`.

(v) You can use shell commands and scripts to accomplish all the system type jobs such as performing backups, checking the number of users and disk usage etc.

(vi) Scripts are very portable from one UNIX system to another or NT to another NT System.

When you shouldn't use Shell commands

You shouldn't use shell commands when:

(i) the application needs to be extremely fast and extremely flexible (here a devoted C program should be used).

(ii) there is a requirement to handle a large amount of database work with possibly on-line logs and reports - (here a facility like dBase is preferable).

(iii) when the entities database is very large.

The `setval` and `getval` script utilities both create a memory resident entity cross reference table on start up.

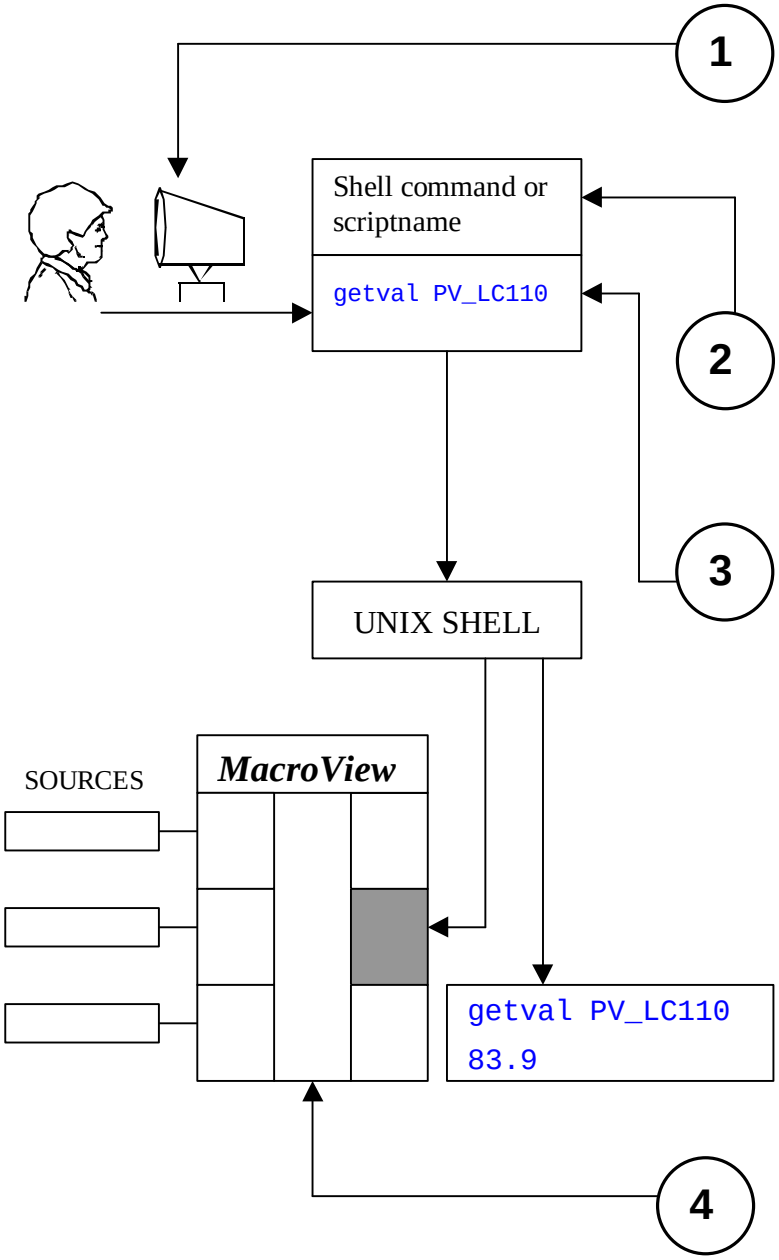
Thus if your system has a large number of entities, then these programs may need a lot of system resources to start up.

Whether this is restrictive depends upon individual situations and needs to be experimented with.

How Shell commands work

The diagram below shows how you can create applications commands (or scripts) and how the system uses them to perform the functions you specify.

How Shell Commands Work



You enter the shell command directly in at the UNIX prompt **OR** writes the shell commands into a script or batch program using an editor like vi, saves the program and makes it executable using

`"chmod a+x <scriptname>".`

The shell command is executed immediately by pressing enter **OR** the script is executed simply by typing the script name.

The UNIX/NT shell takes the command and interacts directly with the ***MacroView*** system to get data or set data in the sources or send messages to the alarm system. **Note:** You may use any of the popular shells available. i.e. Bourne, C shell or Korn shell (The examples used here are Bourne shell). (UNIX only)

Results from the shell commands are sent to the screen or piped to other shell commands.

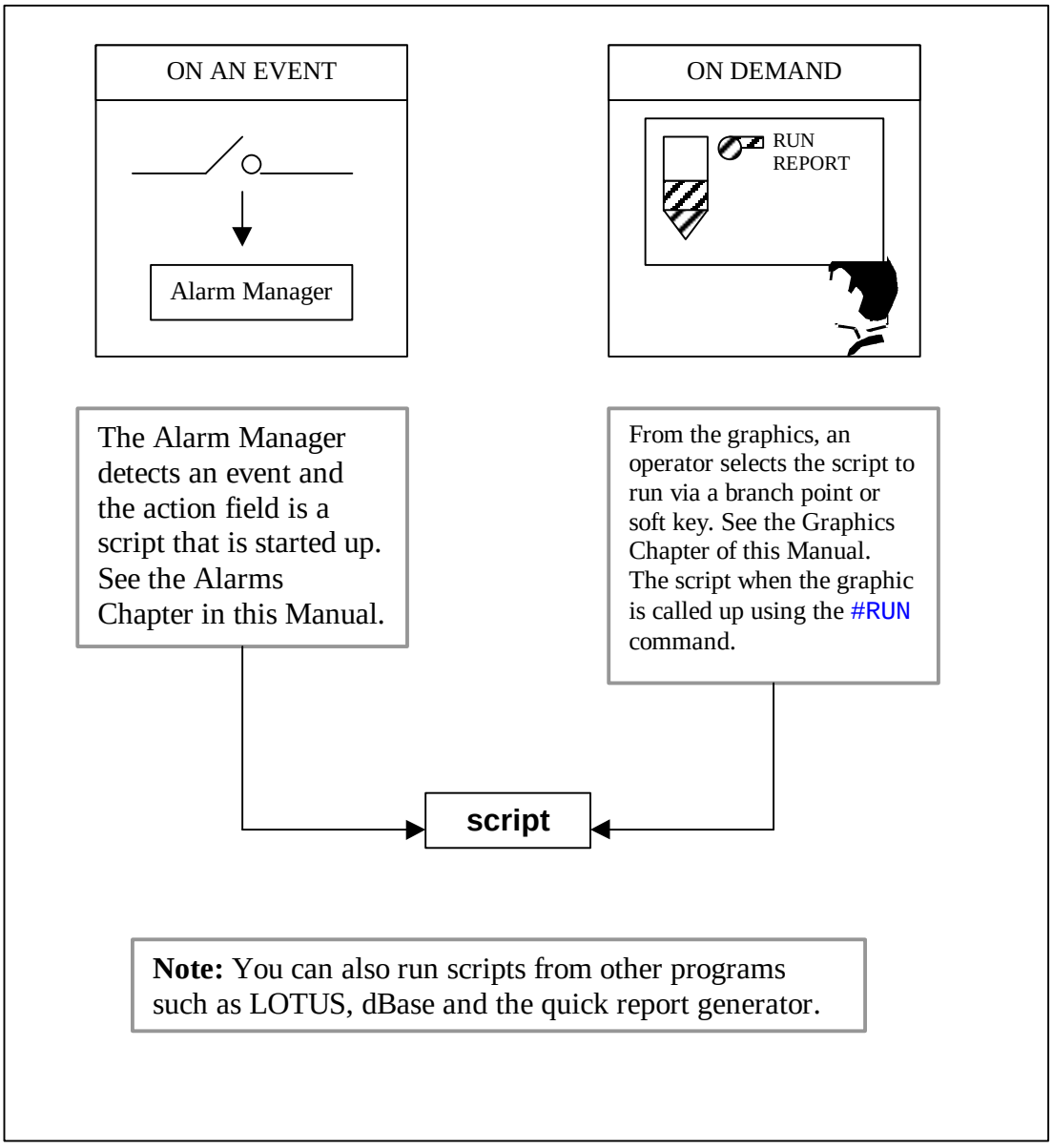
The shell commands work through the data dictionary so you may read data from any source and write data back to any source provided you have the necessary security access.

Executing scripts

There is a variety of ways that you can invoke the program once you have entered the shell commands into a file, given the file a name and made it executable with:

`"chmod a+x scriptname".` (UNIX only)

The diagram illustrates the possibilities.



Getval (Get the value of an attribute_entity)

The shell command `getval` returns the current value of an `attribute_entity` or a series of `attributes_entities`. You can use `getval` in a number of forms.

Simple Form

```
Getval PV_LC110

83.9
```

Enter the name of the attribute and Entity separated by an underscore

The current value of the attribute_entity is returned to the standard output. In this case, the standard output is the console screen.

Redirecting the output

```
Getval PV_LC110 > filename
```

Of course you can use any of the standard Shell facilities such as `|` (pipe), `>`(redirect), `` `(graves), `` `>>` (append) etc. Some of the examples at the end of the section show you how to do this. **Note**: `` `(graves), `` is UNIX only

Getting multiple values

```
getval PV_LC110 SV_LC110 MV_LC110 LS_LC110
83.9 84.0 38.4 AUT
```

You can string as many of these attribute_entity pairs in a command as you require up to a Unix system dependant number.

The values are printed out sequentially and in order.

Printing each value on a new line

```
getval -nl PV_LC110 SV_LC110 MV_LC110 LS_LC110
83.9
84.0
38.4
AUT
```

You use the -nl (new line) flag to specify that each value should be printed on a new line.

Of course, if you want to do some really complex formatting you can use `awk` (the UNIX pattern scanning facility) or cut, paste or `sed` (the stream editor). UNIX Only

If your entity names use the underscore character

```
getval -c PV:PW_33
49.6
```

The `-c` flag tells the system to look for a(:) as the separator between the attribute and the entity. In this case, the attribute is PV and the entity is `PW_33`.

Combinations of the -`nl` & -`c` flags

```
getval -nl -c -w PV:PW_33 SV:PW_33 MV:PW_33 LS:PW_33
49.6
50.0
89.3
MAN
```

You can use both the `-nl` and `-c` flags in combination.

Setval (Set the value of an attribute_entity)

The shell command `setval` writes a value down into the `attribute_entity` in the process or a series of values down to the process.

You must of course have the correct access levels to write a value to the process. If not, a system alarm message will be printed out.

There are a number of forms of the `setval` message

Simple Form.

```
setval SV_LC110 94.6
94.6
```

The value you wish to change to.

Enter the name of the attribute and entity separated by an underscore.

Once the system has written the value to the attribute_entity, it reads it back and returns the value to the standard output. (In this case the console screen).

Redirecting the output

```
setval SV_LC110 94.6 > filename
```

The returned value can be sent to a file as above or, you can use any of the standard FACILITIES such as `|` (pipe),`>>` (append), `` `(graves) `` (**Note:** `` `(graves) `` is UNIX only)

Setting multiple values

```
setval SV_LC110 94.6 LS_LC110 AUT
94.6 AUT
```

You can string as many of the (`attribute_entity value`) settings together in a command as you require up to the system dependant number that is particularly large.

The readback values are printout out sequentially and in order.

Printing each value on a new line

```
setval -nl SV_LC110 94.6 LS_LC110 AUT PV_PMP43 1
94.6
AUT
1
```

Use the `-nl` (new line) flag to specify that each value should be printed on a new line

Of course, if you want to do some really complex formatting, you can use `awk` (the UNIX pattern scanning facility) or cut, paste or `sed` (the stream editor).

If your entity names use the underscore character

```
setval -c SV:PW_33 49.6
49.6
```

The `-c` flag tells the system to look for a colon: as the separator between the attribute and the entity. In this case, the attribute is `PV` and the entity is `PW_33.`

Specifying how long to wait for the readback value

```
setval -w 2 SV_LC110 49.6
49.6
```

You can specify how long the program should wait before reading the readback value. In practice, the `setval` command places the value on a write queue. The actual time to complete the write process will depend on the speed of the source driver. The default wait time is 1 second. If you find the values returned by setval are not correct, first check you are not being blocked by security (i.e. check the system alarms) and if not, increase the wait time.

Combinations of the -nl, -c and -w flags

```
setval -nl -c -w 2 SV:PW_33 49.3 LS:PW_33 AUT

49.3
AUT
```

You can use any combinations of the various flags.

Note: to see how this command can be used in practical applications, take a look at some of the examples at the end of this section.

sendmsg (Send a message to the alarm or message system)

With the shell command `sendmsg` you can send a message to the alarm or operator guide message system from the script program.

In the command, you specify the **priority** of the message, the **colour** of the message, the **contents** of the message and if required, the **superfind** screen associated with the message.

Simple form

```
sendmsg 2 10 "Overload on Boiler seven"
```

The **priority** of the message - i.e. where it is to go and what action is to occur. This can have the following values.

0. System Alarm
1. System Alarm Summary Emergency
2. Alarm: Alarm Summary
3. Warning
4. Messages: Operator Guide Message Summary
5. Log only

The **colour** of the message is entered here. Colours are:

0. Black:
1. White
2. Half Red
3. Half Green
4. Half Blue
5. Half Yellow
6. Half Cyan
7. Half Magenta
8. Half White
9. Dark Grey
10. Red
11. Green
12. Blue
13. Yellow
14. Cyan
15. Magenta

The actual message that will appear in the alarm or message summary.

One of the examples show how you can make this a variable.

Messages with superfind

```
sendmsg 1 15 "Imminent shutdown of reactor"SCH004
```

This is the optional superfind screen. When the message appears in the alarm window, the operator can press the superfind key (Ctrl-F) and this screen will appear. Alternatively, operators can move the cursor to the message on the alarm or message summary and press enter to bring up the screen. To specify the screen you want, use the following form:

1. by screen name, e.g. `apl/screen_name`

   e.g.`SCH/boil`

2. by screen number e.g. `aplnnnn`

   e.g. `SCH0004`

Where `screen_name` is the name of the metafile you can only use 4 letter screen-names, `nnnn` is the screen number and `apl` is one of the following application codes

SCHEMATIC	SCH	ALARM	ALM
OVERVIEW	OVW	SYSTEM ALARM	SYS
GROUP	GRP	MESSAGE	MSG
HELP	HLP	TREND	TRD

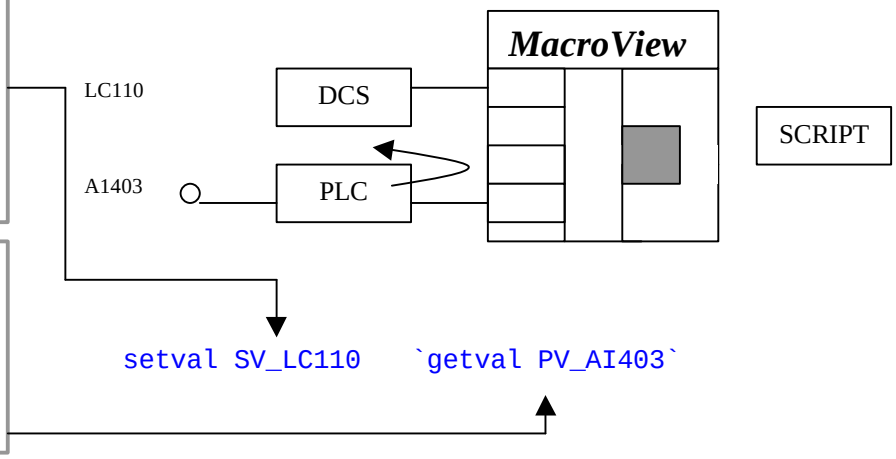
Examples

Simple data transfer

In this example, an Analog input in a PLC is transferred to the setpoint of a level controller in a DCS system.

Once the value of `PV_AI403` is returned (lets say the value is `43.7`) the new command becomes setval `SV_LC110 43.7`. This simply sets the setpoint value of `LC110` to `43.7`.

The expression inside the graves (`) is first evaluated and a number is returned in place of the `getval PV_AI403` command.



Note:
This one line of script is the entire program. a sleep command could be used to regulate the set period -- see the next example.

More complex data transfer

This example is the same as the last example except:

(i) the transfer takes place regularly once every 5 minutes and,

(ii) the transfer only takes place if the loop status of LC110 is AUTO

```
while [ 1 ]
do
      if expr `getval LS_LC110`= "AUTO"
      then
      echo "transferring data"
      setval SV_LC110 `getval PV_LC110'
      else
      echo "tag not in auto - No transfer"
      fi
      sleep 299
done
```

This statement is always true so the statements between this line and the done statement are executed forever, i.e. a continuous loop

If the status is AUTO, then:

1. Write a message "transferring data" &
2. Transfer the data

Otherwise, write a message saying there is no transfer and wait one second (equivalent to the time it would have taken in the setval command)

Wait a further 299 seconds to make up the 300 second loop time and redo the process.

Note:

The general form of the while statement is:

```
while       command_list_1
do          command_list_2
done
```

The general form of the if statement is:

```
if          command_list_1
then        command_list_2
      else        command_list_3
fi
```

Simple report

```
:
echo  "     Boiler values for `date' "
echo  " "
echo  " Current level is `getval PV_LC110'       percent "
echo  " Current flow is `getval PV_FIC110` GPM "
exit
```

In the report heading, the date is printed in the form:
Sat Aug 01 06:22:30 EDT 1992
The command in the accent `i.e. `getval PV_FIC110` is first evaluated and the value is placed in the string i.e. Current flow is 83.9 GPM is printed out.
Note: To get the report to print out, you must redirect the output to the line printer spooler. For example, if the script name is simple_report, you would type:

```
simple_report | lp
```

Alternatively, you can send the output to a file i.e. using `>>` or `>` and print the file directly to a device.
`simple_report>> |pt1:` (NT Version)

Script using the calculator bc and variables

```
level=`getval PV_LC110'
sv=`getval SV_LC110'
echo " ( $level + $sv) /2" | bc
avg=`echo "scale=4 ;( $level + $sv) /2 " | bc'
echo "Average is $avg "
```

UNIX
To assign a variable, just use the equal sign without spaces and a command. Later you refer to the variable by placing a $ sign in front of it (in this case $level).
You can send more than one instruction to `bc` at a time, just separate each command with a semicolon. For more information on `bc`, type `man bc` at the UNIX prompt. If the manual pages have been installed, you will see an explanation of `bc`.
In this command, the outer accent graves i.e. `` `echo ..bc ` `` means evaluate the whole command. This value is returned and set equal to avg.
To perform a calculation in the Bourne shell, we send a string to the calculator `bc`. This is done by printing (using echo) the string `($level + $sv)/2` and piping this result to `bc`.

Script using a loop

```
n=5
echo "looping $n times"
while [ $n -gt 0 ]
do
echo "Current level is `getval PV_LC110' "percent on `date'"
echo "setting value of SV to `setval SV_LC110 5.0' "
n= `expr $n - 1`
done
```

The end of the loop test -- if true, all the statements between here and the done command are executed.

This command decrements n.

Note that you can only use `expr` for simple integer calculations. Use `bc` for more complex calculations.

PROGRAM

Script that passes parameters

One of the powerful features of shell commands, is their ability to pass parameters so that you can create subroutines and build up a toolbox of useful routines. In this example called live, we want the report to write the live value of an entity **specified in the command line** and to continue writing this value until the break key is pressed.

```
if [ $# -lt 1 ]
then
echo "\n Usage: live attribute_entity e.g. live PV_LC110 \n"
exit 1
fi
echo "\n Values and items for $1 .. Terminate with the delete key. \ n"
while [ 1 ]
do
      echo " `getval $1`                          `date' "
      sleep 1
done
```

First check that you have entered an attribute_entity pair. In this case, `$#` refers to the number of parameters in the calling command.

Write a message as a heading that reflects the entity name and tells you how to get out of the loop.

Note: The `\n` just puts an extra line into the report.

Here we use `$1` as the name of the attribute_entity. To use this program, we type:

```
live PV_LC110
```

and we get:

Values and times for `PV_LC110 ..` Terminate with the delete key

50.1	Sat Aug 01	06:22:18 EDT 1992
51.3	Sat Aug 01	06:23:20 EDT 1992
52.0	Sat Aug 01	06:24:22 EDT 1992 etc.

Script that sends a message to the alarm screen

This script shows some of the techniques used in the previous scripts and how you can send values in a message to the alarm summary. The program flow is simply:

(i) read the values

(ii) calculate the deviation (and print it to the screen for interest)

(iii) create the sendmsg command with the value in the text part of the command

(iv) execute the command.

```
# simple program to calculate a deviation, print results and send message
# to the alarm summary.
#
# Calculate the deviation
level=`getval PV_LC110'
sv=`getval SV_LC110'
dev=`echo "scale=2;$level - $sv" | bc'
# Print results on the screen
echo "Current Level \t \t $level \t Percent"
echo "Current Set Value \t $sv \t Percent"
echo "Current Deviation \t $dev \t Percent"
#
# Send the message to the alarm summary.
sendmsg 0 3 "Current Deviation of LC110 is $dev Percent"
```

Reading Historical Data

Table 8: Reading Historical Data

Shell Script Statements	Comments
	There is no direct way to get historical data into a script. You must first get the data into an entity and from there, the script will read the values.
`dbfhist ../rep/hr.dbf -t 3600 -n 24` `FLOW0=`getval PV_FIC100(HR)'` `FLOW1=`getval PV_FIC100[1](HR)'` `FLOW2=`getval PV_FIC100[2](HR)'` `LEV0=`getval PV_LIC100(HR)'` `LEV1=`getval PV_LIC100[1](HR)'` `LEV2=`getval PV_LIC100[2](HR)'`	**Getting History with a dBase file:** 1. First create the database i.e. in dBase use `create hr`, add the required fields, e.g. `PV_FIC100, PV_LIC100` etc. (See the History Chapter for more information on setting up this file) 2. Create the entity HR and give it the address of the database, e.g. /u/macro/rep/hr.dbf. 3. Run the `dbfhist` utility, which places the historical values into the database? `dbfhist` can be run from the script or using `cron` (see A-CRON). In this example, 24 hourly values are placed in the dBase file. 4. Read the values from the database. In this case, the variables FLOWN are loaded with the hourly averages of FIC100 and the variables LEVN are loaded with the hourly averages of LIC100.

Learning to use scripts

The use of shell commands and scripts is obviously an enormous subject. For further information, we suggest you read the following material:

- "UNIX Shell Programming" by Lowell Jay Arthur Second Edition John Wiley & Sons Inc. ISBN 0-471-518221-21
- Sun Microsystems' description of the Bourne Shell - Appendix C
- For more information on the UNIX tools, we suggest you look at the UNIX Power Tools series by O'Reilly and Associates, Inc.

  E.g. for information on `sed` and `awk` read "Sed & Awk" by Dale Dougherty O'Reilly & Associates, Inc. ISBN 0-937175-59-5

The only real way to become proficient in scripts is to practice writing them. Because of the interactive nature of scripts, you will be creating useful and functional scripts in a relatively short period.

11.5 Optional Facilities

The remaining sections briefly describe the optional programming tools available to you.

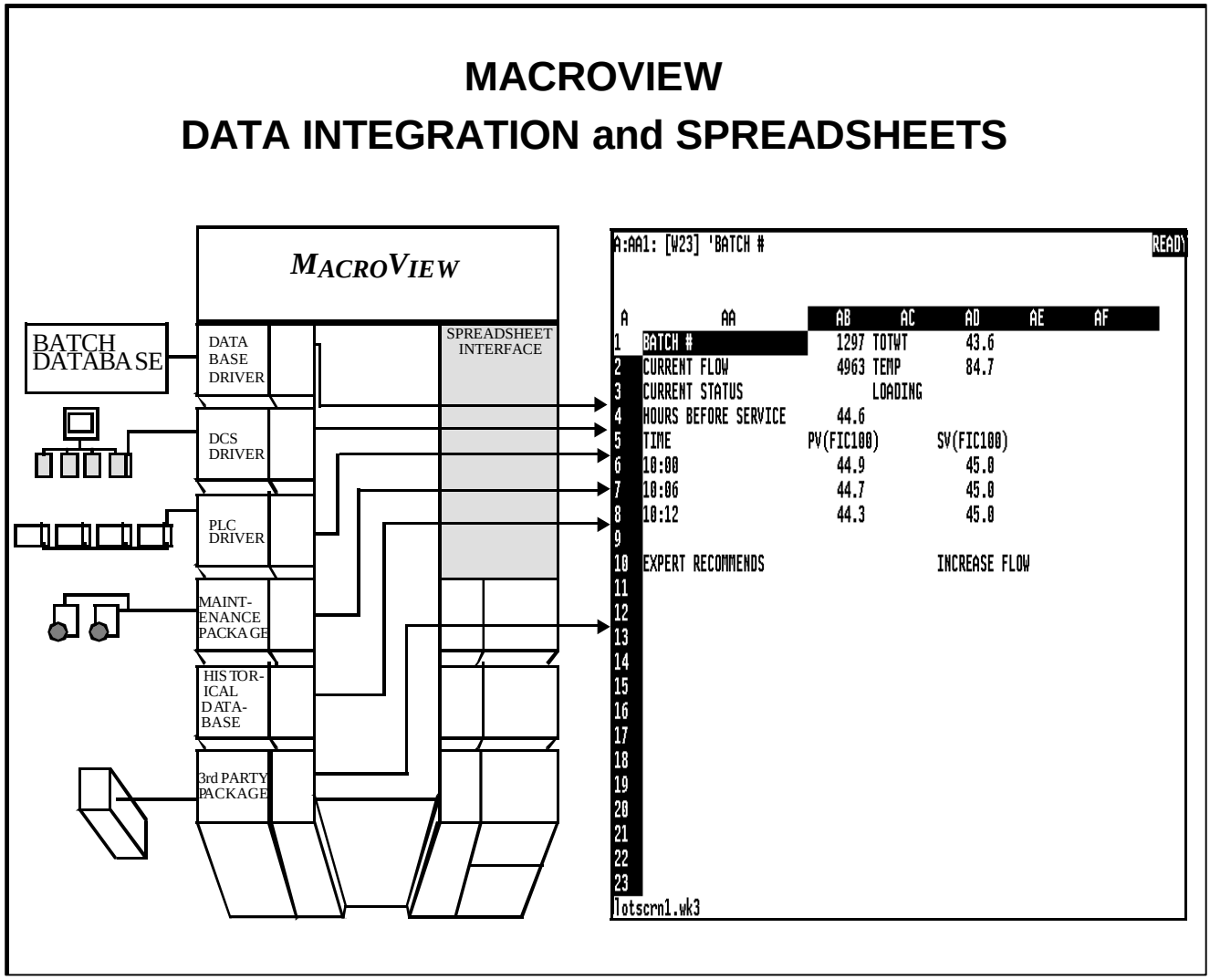
For more information on these tools, you can ask your distributor for the specific documentation referring to these tools.

These documents not only describe the tools in detail, they also include numerous examples both simple and complex.

Lotus Interface

In almost every process control or SCADA environment, the spreadsheet has become an essential tool to managers, planners, engineers, accountants and virtually everyone in the middle and upper managerial position.

The ***MacroView*** Spreadsheet Interface provides a simple but powerful means to connect the real world directly into these spreadsheets. Using ***MacroView***'s integration facilities, this data can come from such widespread sources as PLC's, DCS's, Laboratory analyses etc



The only requirements for you to start getting real data into spreadsheets are:

(i) You must know how to use a spreadsheet, and

(ii) You must know the entity name or tag name of the data you want brought into the system.

You may use the spreadsheet interface in a number of different ways:

- directly on the main workstation server,
- on a workstation anywhere in the network:
  - for background reports,
  - for high level control or
  - to generate 123 files for other users.

This section of the Programs Chapter only describes the basic functions of the spreadsheet interface.

The package document (P-SSI) shows in more detail how you use these functions.

It also shows how various applications such as background reporting can be implemented. In the Appendices, installation and other topics are covered.

A separate Appendix describes the example spreadsheets that come with the distribution diskettes.

Note on Suitability

You may use a Lotus Spreadsheet either in the Main Server or in a PC linked to the server using a DDE link. In most cases, the preferred option is to use the DDE interface. Please contact your ***MacroView*** distributor to get more information about the ***MacroView*** DDE Server.

Lotus Terminology

In this section, the standard key names have been used. (For example "Press APP1").

This means, "Press the key assigned to APP1 or application #1."

With the Lotus documentation, in the installation notes for the platform that you are using, there is a summary of which keys are mapped to which functions.

For example, APP1 is mapped to alt-F7 or function key 7 on the SUN platform.

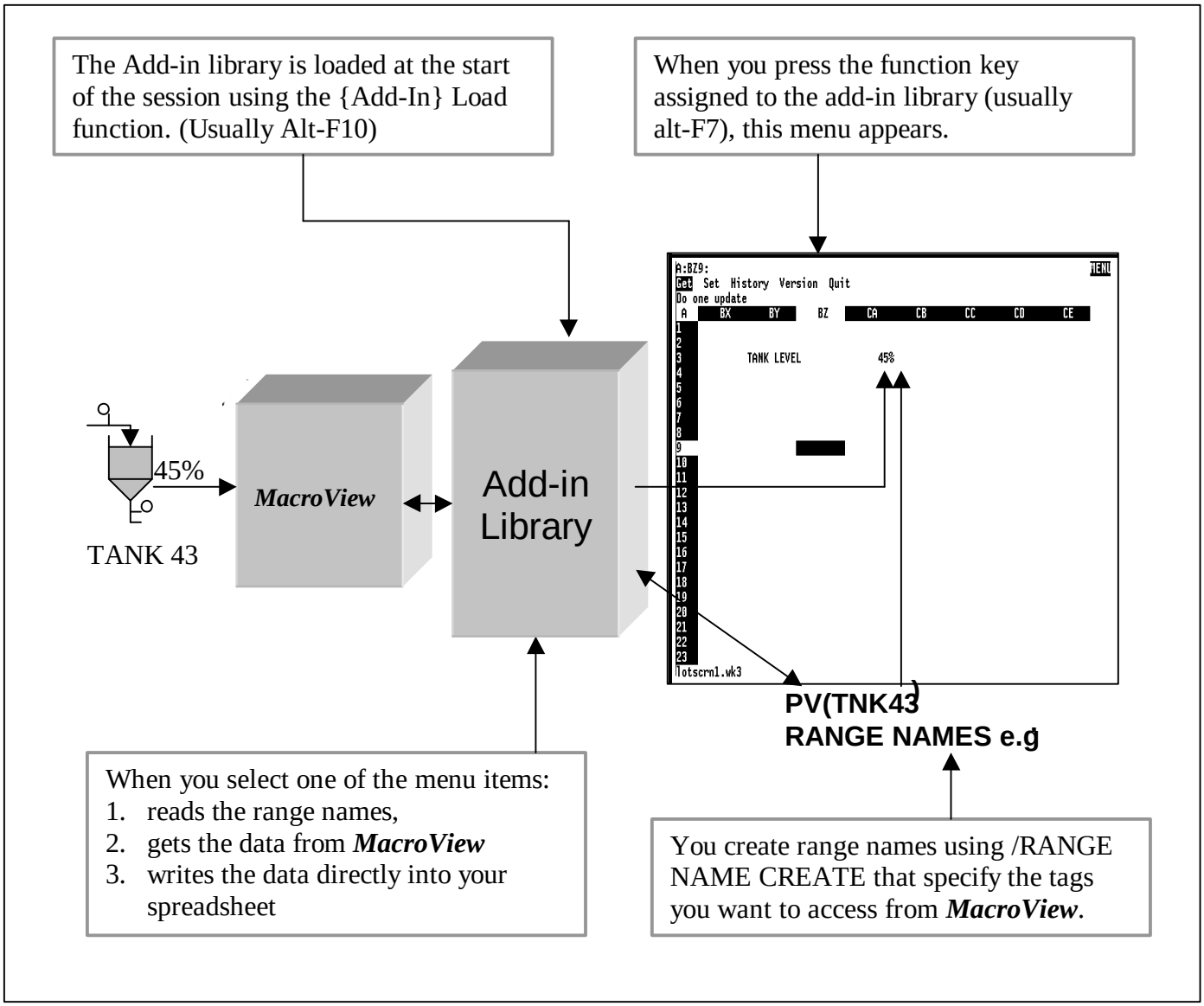
We have also used notes like / RANGE NAME CREATE - this means - "from the READY mode press /, the work sheet main menu appears, select RANGE the range menu appears", etc.

How the Spreadsheet Interface Works

The Spreadsheet "talks" (i.e. reads and writes to the ***MacroView*** live and historical data) via a set of add-in libraries.

These libraries are loaded into the system at the start of the session.

Once loaded in (this can be done automatically) you specify what information you need by simply writing the entity names into cells or naming ranges as the entity names.

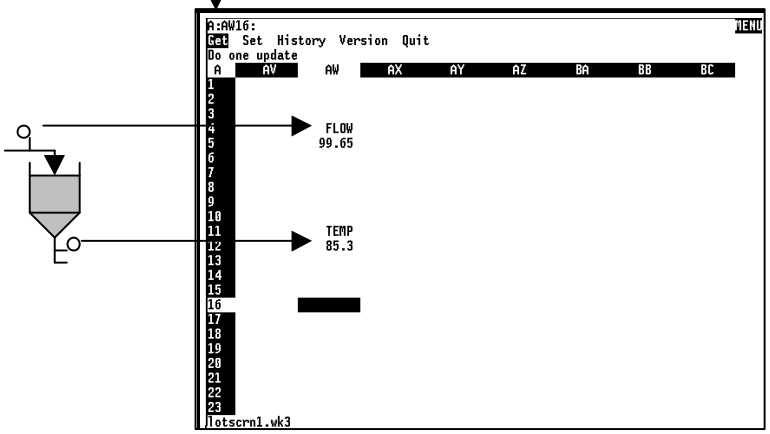


Interface Functions

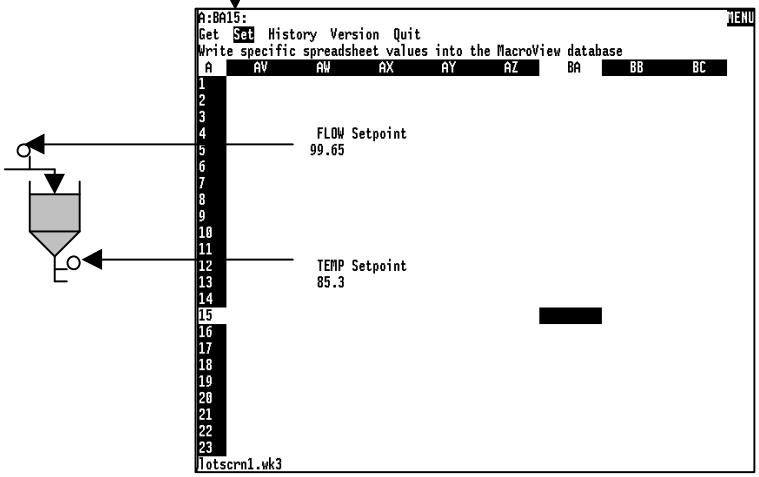
The main functions carried out by the spreadsheet interface are shown below.

1. **Getting single values**

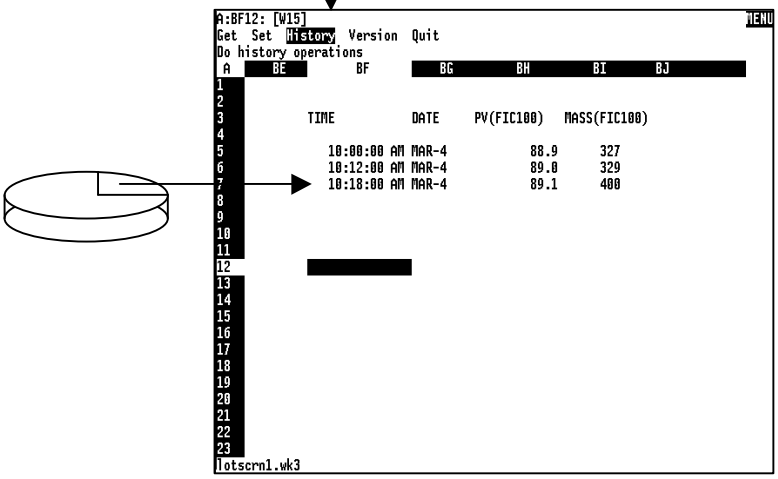
   Single cells are updated with live data from the process.



2. **Set values down** to the process Single cells are sent down to the process as set points or targets.

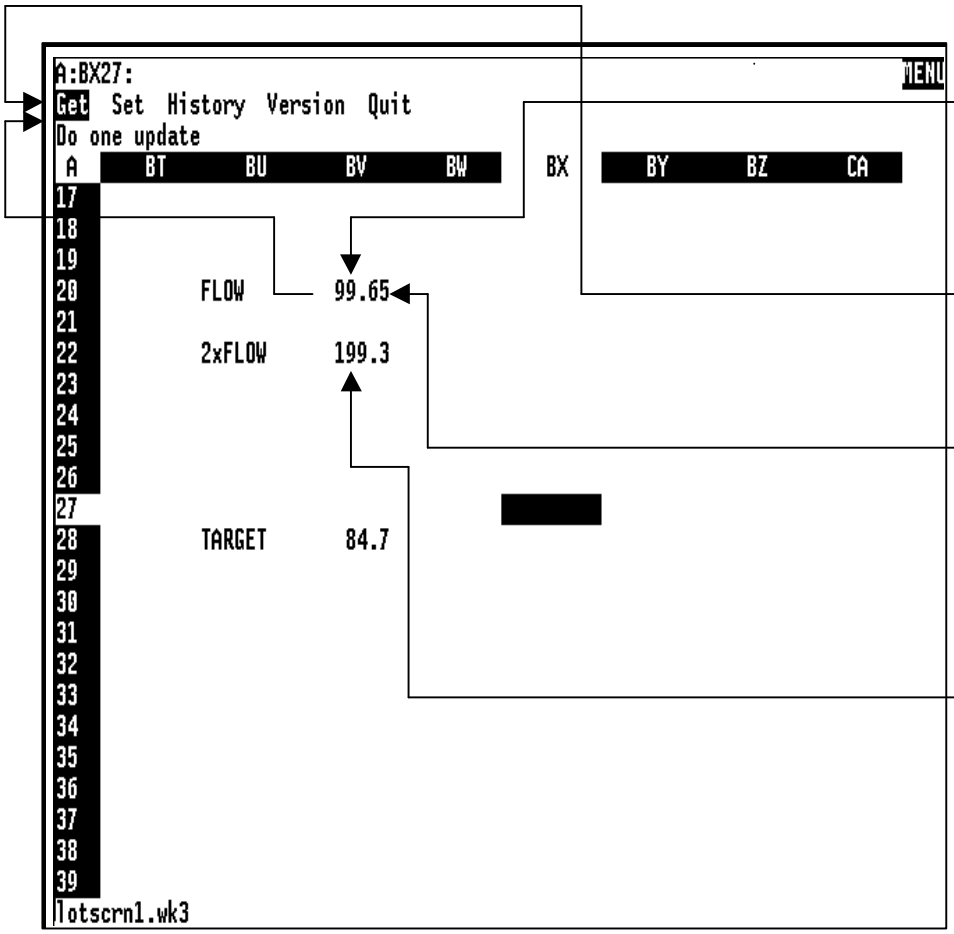


3. **Reading historical data** directly into the spreadsheet A block of historical data is read into the spreadsheet.



Getting live data values from *MacroView*

This diagram shows the steps required setting up the spreadsheet to read live data.



1. You name the cell with `/ RANGE NAME CREATE PV(FIC100).` You can have any number of cells being updated throughout the spreadsheet. **Hint:** For the range names, instead of using the form `PV(FIC100)` you may also use `PV_FIC100` or `PV[FIC100].`
2. This menu appears when you press [APP1] (usually alt-F7)
3. Press G for Get and in a moment the table appears into the spreadsheet

Once the spreadsheet has been updated, formulae in the spreadsheet are recalculated.

How It Works

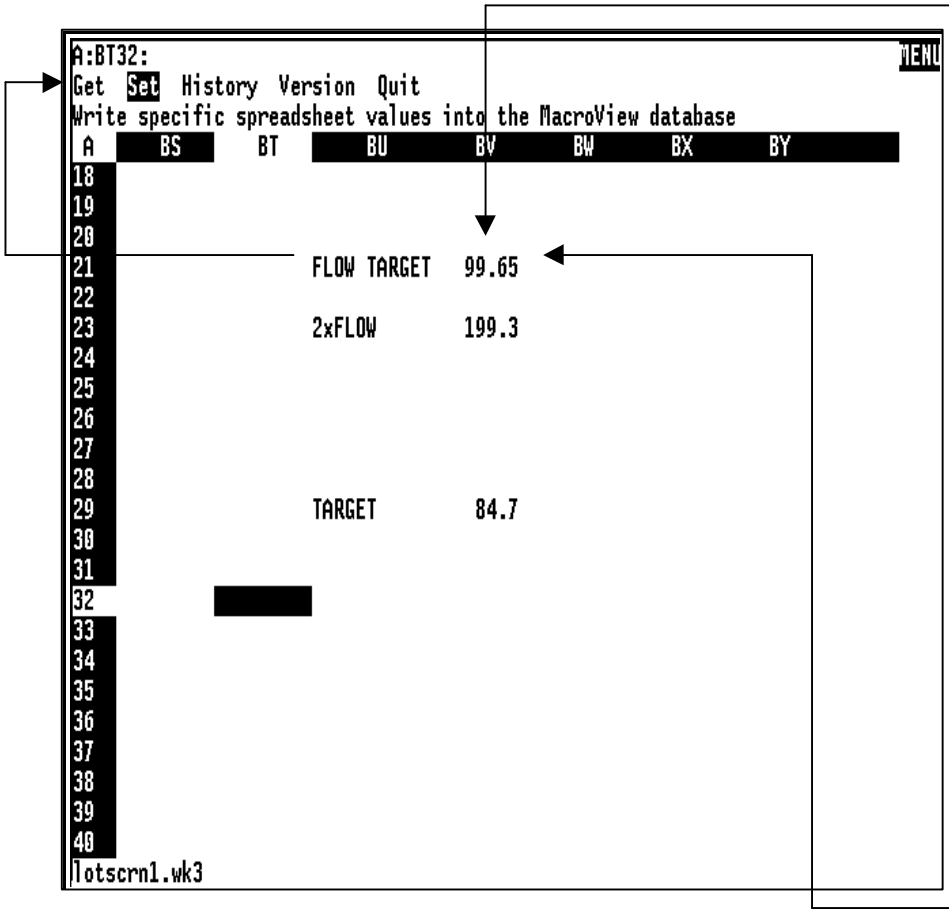
The add-in library:

1. Reads the range names of the spreadsheet looking for attribute(entity) combinations, and
2. It then gets the values from MACROVIEW and writes them into the spreadsheet.

Setting values down to the process

This diagram shows the steps required setting up the spreadsheet so that data can be written to the process.

PROGRAM



1. You name the cell with `/ RANGE NAME CREATE to be =SV(FIC100)`.

   The = sign indicates a set operation is required.

   You can have any number of cells as set values throughout the spreadsheet.

   Hint: For the range names, instead of using the form `=SV(FIC100)` you may also use `=SV_FIC100` or `=SV[FIC100]`

2. This menu appears when you press [APP1] (usually ALT F7)

How It Works

The add-in library:

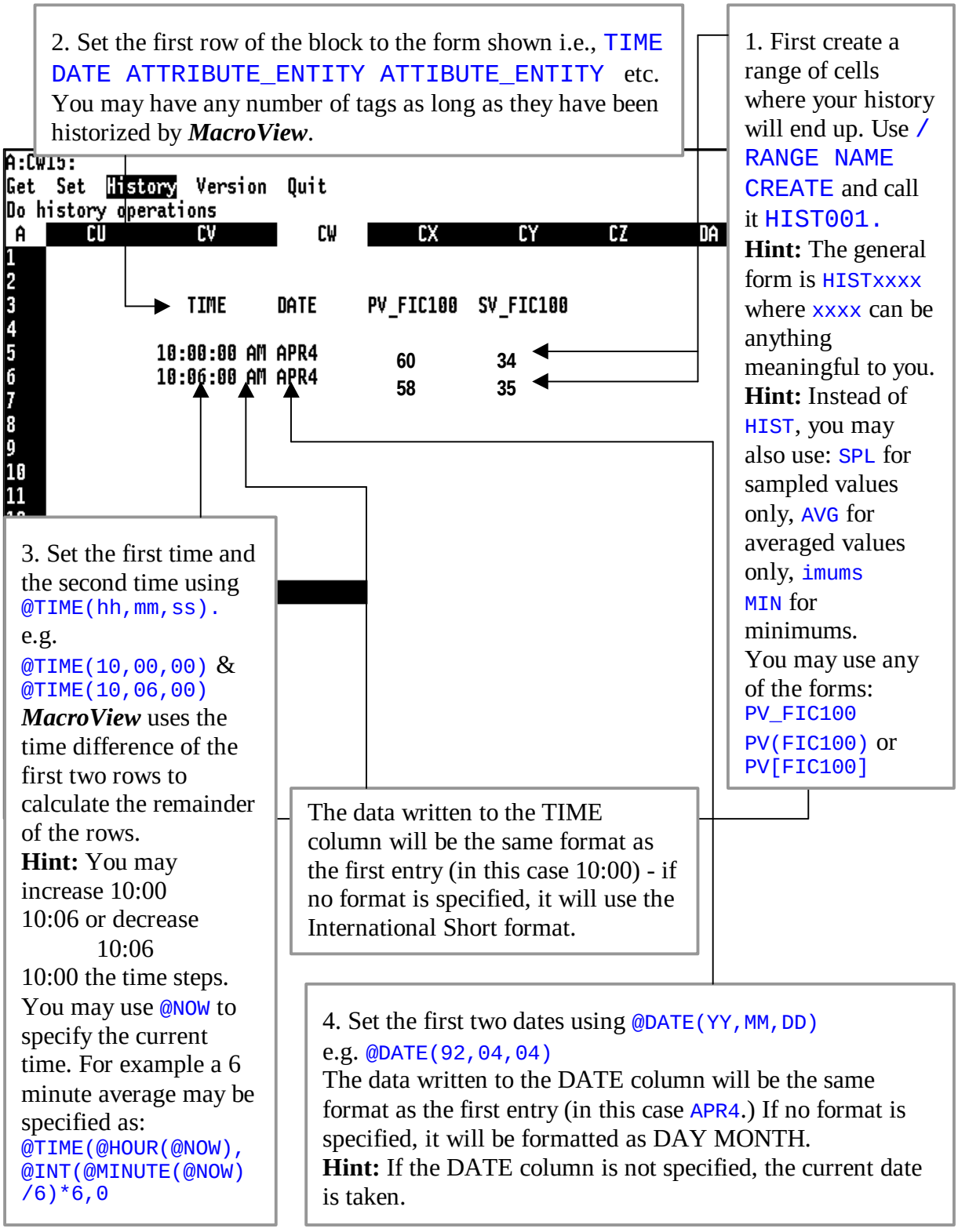
1. Reads the range names of the spreadsheet looking for `=attribute` (Entity name) combinations (e.g., `=SV(FIC100)`
2. It then sends these values to MACROVIEW to be written into the field.

Warning: Your user name must have access permission to write the information to the tag. If it doesn't an error message will be created.

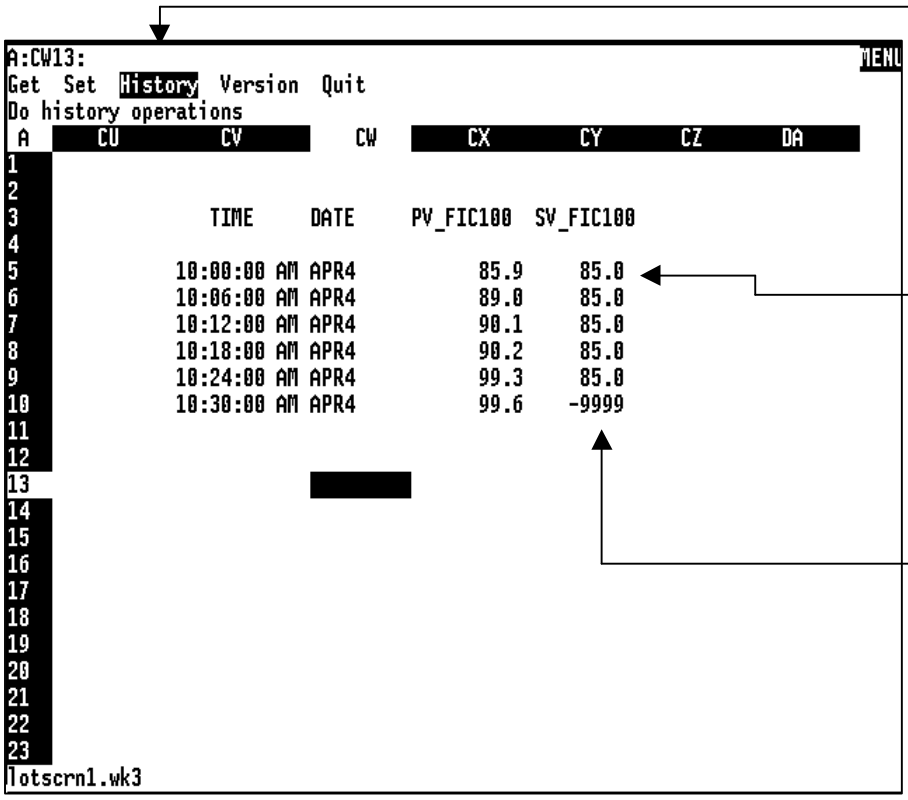
Reading a block of historical data

The next two diagrams show how you set up the spreadsheet to read historical data directly into the spreadsheet.

Setting up the Historical Block



Reading the Historical Block



Call up the ***MacroView*** menu using [APP1] (typically ALT F7) and press the H key for History. After a few moments, the data will be displayed in the range you specified.
invalid data is shown as
`-9999.I`
If there is no value for the time and date, the cell will be left empty.

How It Works

When the add-in is started, it reads the range names looking for the correct form, it then reads the first row for the entity names, it checks the start time and increment and then collects the data and writes it into the block you specified.

Other Issues not covered here

This section is designed only as an introduction to the spreadsheet interface.

In the document P-SSI, the following additional issues are covered:

(i) Other functions such as block read, block set and version read

(ii) Error handling

(iii) Networking

(iv) Applications such as:

- Writing reports
- Background reports
- On-line logs
- Using dBase entities
- Continuous updates
- Storing real time data

(v) Installation

11.6 dBase Interface

The dBase Interface is an optional package and is therefore only mentioned briefly here. For more information, please ask your distributor for the document P-DBI. Not required for NT.

Reasons for using dBase

There are a number of reasons for seriously considering dBase when writing applications in ***MacroView***

(i) **Data Management** - A large part of successfully managing process control sites is being able to capture, manipulate and display large amounts of data to the people making decisions. dBase is the ideal tool to handle both the data that is collected automatically and in particular, the data that is entered manually into on-line logs.

(ii) **System file compatibility** - All the ***MacroView*** system configuration files such as the entities database, sources database etc. are dBase compatible and can be accessed with dBase.

(iii) **dBase Integration** - dBase files may be treated like any other entity in the system - i.e. they are given entity names, may be assigned to trend pages, may be displayed in graphics etc. This means that, for example, data collected and entered manually may be presented on a trend display at the same time as a value that has been collected automatically in the process.

For more information on how to set up a dBase entity, please see the section dBase Source in the Sources Chapter.

(iv) **Data Export** - There are numerous third party packages that will accept data in a dBase file. e.g. Lotus, Project, etc. This is another way to get data into and out of ***MacroView***.

(v) **dBase Language** - dBase has an extremely easy to use, yet powerful 4GL language which is particularly useful for logs, reports, menus, queries etc. This language can be used interactively and in a program.

(vi) **dBase Tools** - There are a large number of tools available in dBase (such as ASSIST) which are extremely useful and finally,

(vii) **Industry Standard** - Being an industry standard, there is an abundant supply of software people and experience that can be used for development and support.

A Typical dBase setup

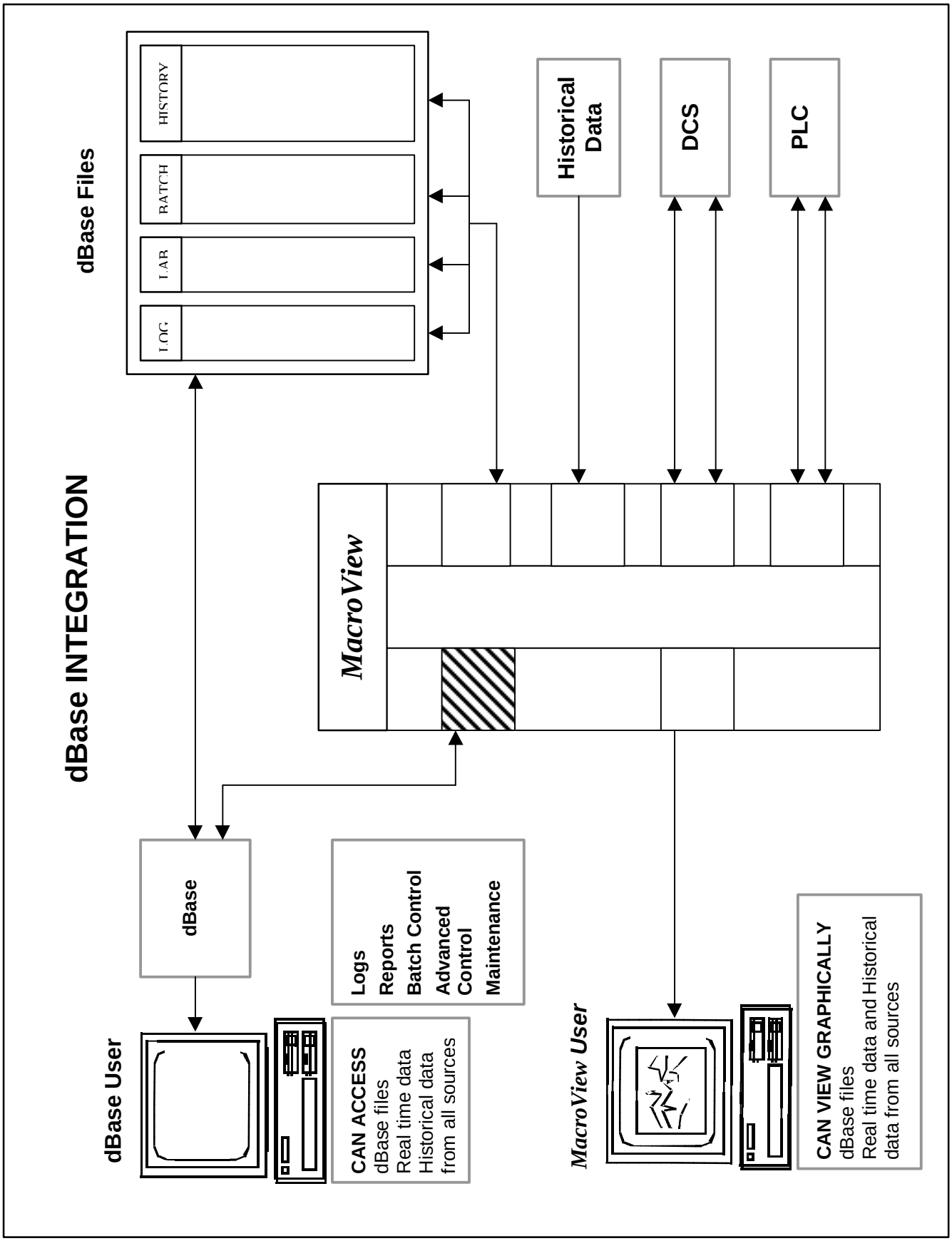
The next diagram shows the interaction between ***MacroView*** and dBase at a typical user site.

As can be seen, the power of the system is the ability to integrate dBase files and dBase itself into the ***MacroView*** system.

Please consider the following points:

- **Data** - There are a number of different kinds of data shown.

- **Data in dBase Files** - This is usually entered manually via dBase by operators, by a background dBase program which collects the data or by the historical utility `dbfhist` which collects history in dBase format.
- **Real-time Data** - This data is collected by the drivers and made available to users. In this case you is dBase itself.
- **Historical Data** - This data is collected regularly in the historical files and can be transformed across to the dBase files using the utility dbfhist.


dBase INTEGRATION
dBase Files
HISTORY
BATCH
LAB
LOG
Historical Data
DCS
PLC
MacroView
dBase
Logs
Reports
Batch Control
Advanced Control
Maintenance
dBase User
CAN ACCESS
dBase files
Real time data
Historical data
from all sources
MacroView User
CAN VIEW GRAPHICALLY
dBase files
Real time data and Historical data from all sources

Users

You can interact with dBase or dBase files **within the graphics environment**. When you are in the normal ***MacroView*** graphics environment, you can see dBase files in exactly the same way as normal entity. I.e. we give the dBase file an entity name, and the fields are treated as attributes.

So just as you would call up the PV of a DCS entity LC110, so you could call up the O2 field of an entity called LAB.

This feature is extremely useful because you can now freely mix dBase data and real time or historical data on trends, graphics, groups, overview etc.

For more information on how to set up a dBase entity, please see the section dBase Source in the Sources Chapter.

Applications

The diagram shows some typical applications that could be run in the dBase environment. For example:

- **Queries** - usually in interactive mode at the dBase prompt.
- **On-line Log** - dBase is ideal for setting up log screens that were formerly entered on paper.
- **Reports** - dBase forms a powerful reporting facility because it can be used to access and manipulate real-time, historical and dBase type data.
- **Other Applications** - Because of the programmability of dBase, it is well suited to a large number of applications including such applications as recipe management, batch handling, maintenance, inventory control, etc.
- **Menus** - You can use dBase to set up menu screens so that you can easily access and administer the various applications.

Learning dBase

This document does not cover the use of dBase itself - e.g. the dBase commands such as USE, SELECT, CREATE etc.

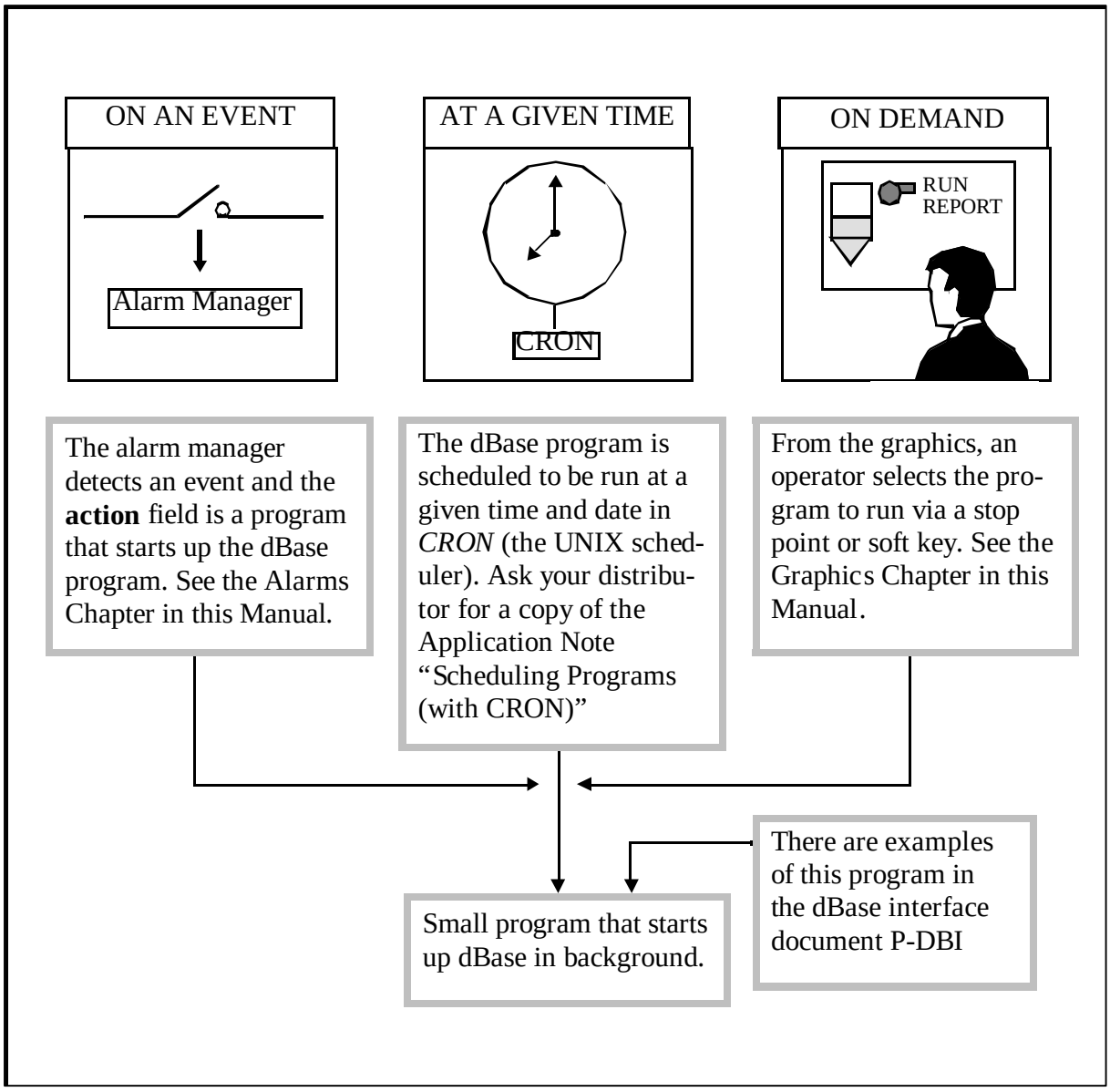
To learn dBase, we recommend you either attend a training course or purchase one of the numerous excellent books on dBase.

But above all, practice by writing dBase programs as much as possible.

Starting a dBase Program

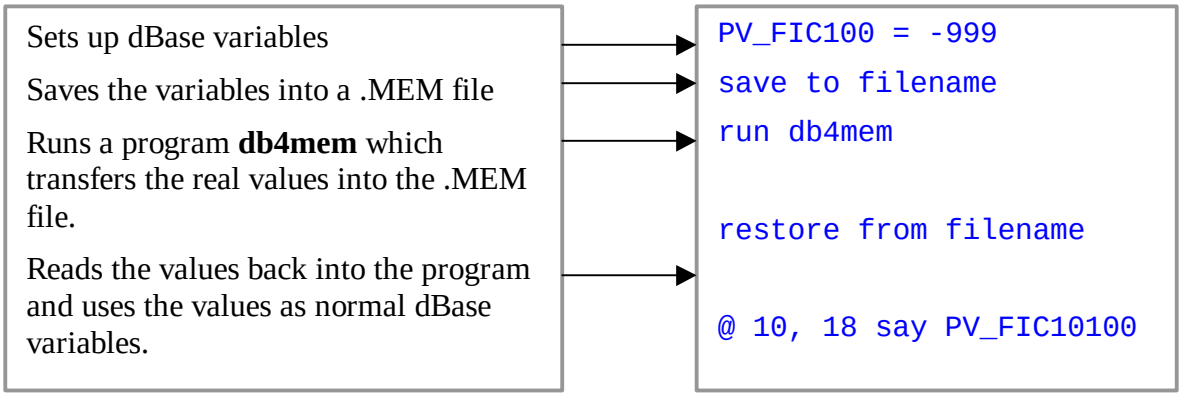
dBase programs may be started directly by you from dBase itself, or directly at the UNIX prompt by typing dBase `<program-name>`.

Alternately, as the diagram shows, the programs may be started in background on an event, at a given time or on demand from a graphic screen, i.e.



Getting real-time Values into dBase

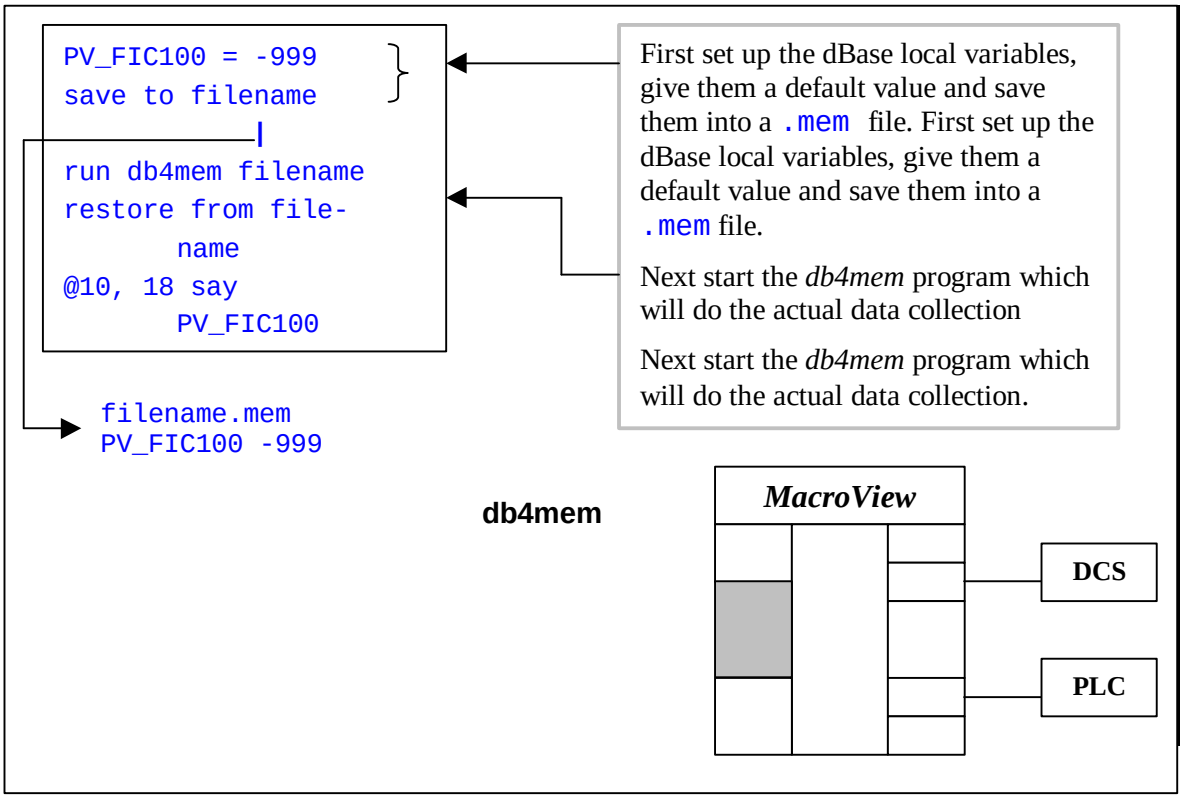
The two parts of the diagram on the next page show how you can get real-time data into a dBase program. In essence, the small program segment in the dBase program does the following:



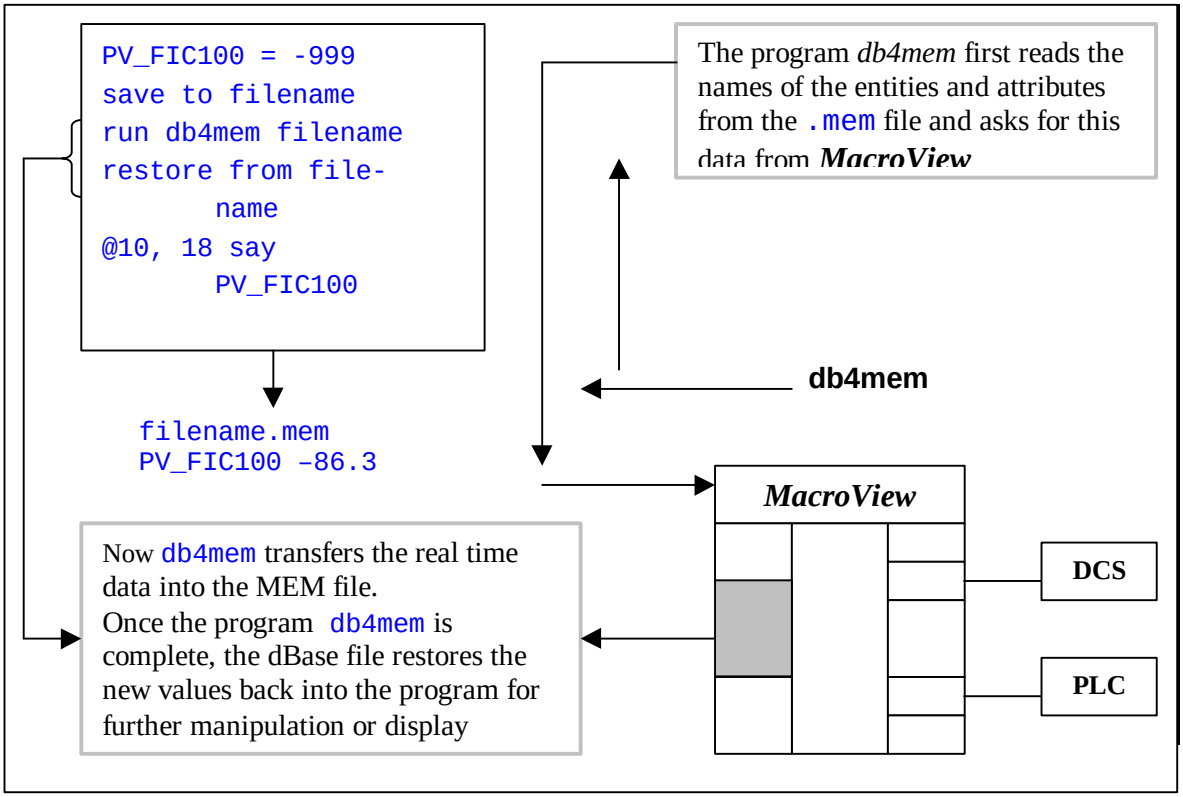
The full document for the dBase interface describes other features of the real-time data access such as:

- reading long entity names
- writing data back down to the process
- Additionally, the document shows how errors are handled and provides numerous examples.

Set up and start the transfer



Transfer the data



Getting historical values into dBase

The two parts of the diagram on the next page show how you can get historical data first of all into a dBase file and later into the dBase program itself.

To read the data, you need to:

(i) **Create a database** where the historical data will end up (the field names are actually used to specify which entity historical data is to be retrieved);

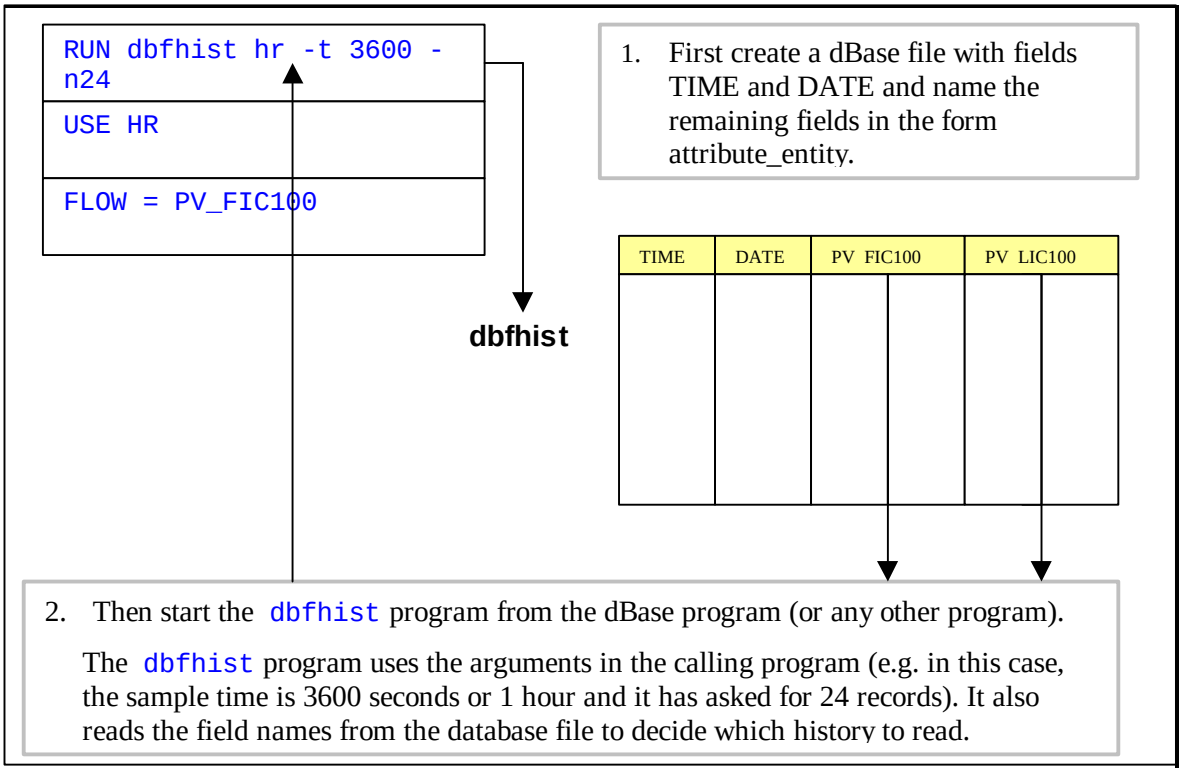
(ii) **Start the `dbfhist` program** with arguments that specify the database file, the start time, number of records to be retrieved etc. Once the `dbfhist` program has retrieved the data;

(iii) **Use the database** like any other database and bring the historical data into the dBase program for calculations and display.

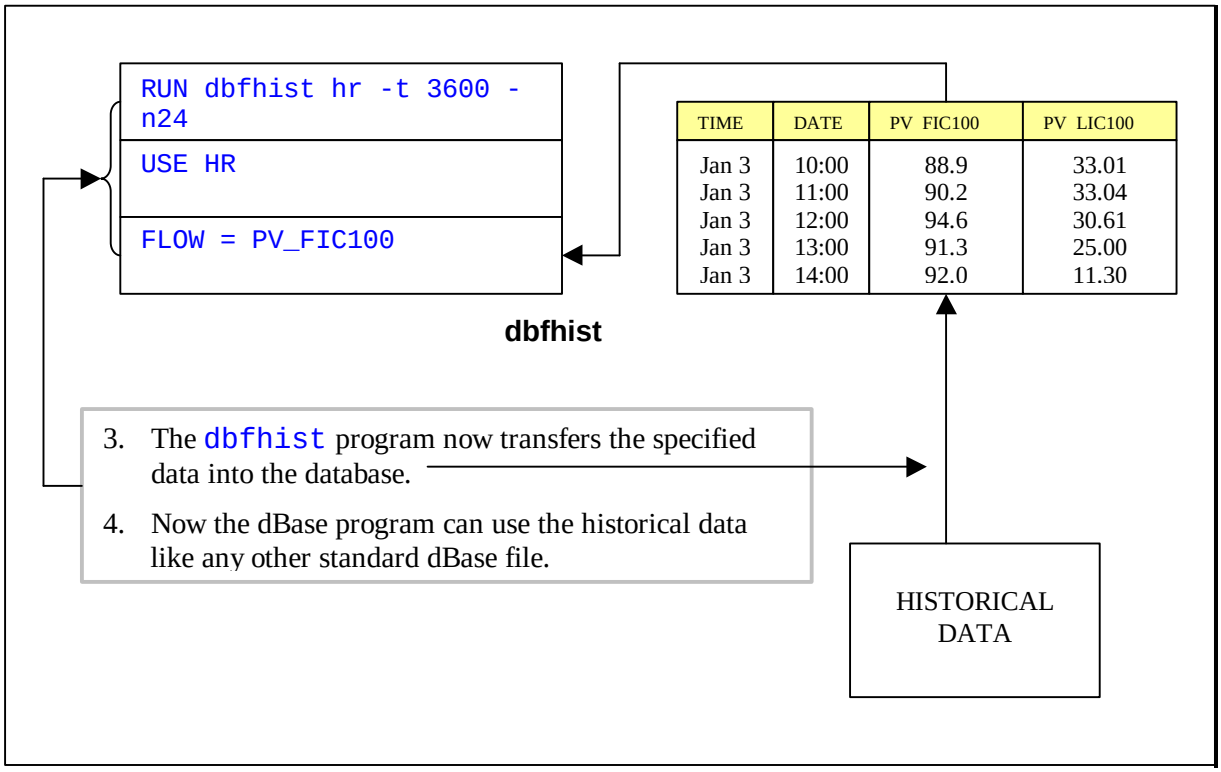
This section only gives an overview of the use of the historical data retrieval. The full document "dBase Interface" number P-DBI provides additional data such as:

- a full description of all the command line options:
- various other ways you can specify the attributes_entities and
- numerous examples.

Set up and start the transfer



Historical data transfer and Use



Issues not covered here

This section was designed only as an introduction to the dBase Interface. In the full document, "The dBase Interface" D-DBI, the following additional issues are addressed:

(i) Getting real time data (in more detail)

(ii) Getting historical data (in more detail)

(iii) Networking

(iv) Applications
- Simple report
- Report with long entity names
- Sending values to the process
- Putting historical data into a format for trending
- Getting Historical data
- Running reports in background
- Setting up dBase tags

(v) Appendices

(vi) Installation
- Error Messages

11.7 Fortran/C Interface

The Fortran/C interface is a set of software links that allow you to access real time and historical data via C and Fortran programs.

The Fortran interface is an optional package and is therefore only covered briefly here. This section is intended to give you a broad overview of the capabilities of the interface.

For more detailed information please ask your ***MacroView*** distributor for the document Fortran/C Interface, document number D-FORC.

Suitability

The Fortran/C interface package is ideally suited for the following types of applications:

(i) High speed and complex calculations, e.g. models, advanced control, optimisations, expert systems etc.

(ii) Interfacing to systems that are already written in C or Fortran.

(iii) Corporate standardisation. Where you or the corporation has a preference for C or Fortran and already has experience in these languages.

(iv) Creating flexible user interfaces. Where a complex user-interface is required or where the data structure is complex, the C language should be chosen because of its' flexibility.

Structure of the Interface

For you, the C calls are very similar to the Fortran calls. For example, the `sendmsg` function has an equivalent sendmsg routine in Fortran.

This is because the Interface routines have been written in C in a form that enables them to be directly called from Fortran.

This means that someone familiar in the C interface can easily use the Fortran interface and vice versa.

This also means that it will be easy to add PASCAL and other compiled languages in the future.

The Interface Functions

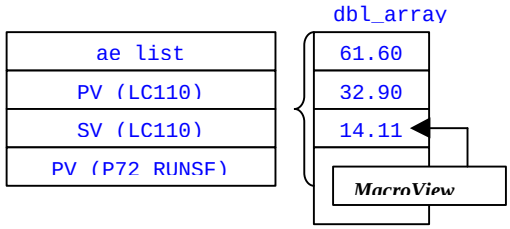
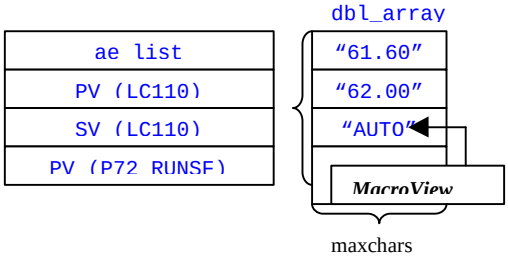
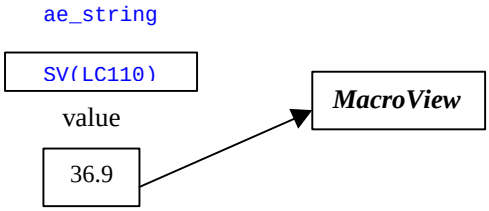
The Fortran/C interface consists of a set of software functions that can be divided into real-time access, historical access and miscellaneous functions.

The following is a summary of the functions available in the interface Real-time Access.

Table 8: Interface Functions.

Function Call	Result

`fc_list (n_value, ae_list)`
Sets up a list of entity and attribute names to be used as the specification for data transfer.
	`n_values` { `ae list`
	`PV (LC110)`
	`SV (LC110)`
	`PV (P72 RUNSF)`

Function Call	Result
`fc_getdbl (n_values, dbl_array)` Transfers data using the previous specified list from ***MacroView*** into a double precision floating point array.	
`fc_getstr (n_values, string_array, maxchars)` Transfers data using the previous specified list from ***MacroView*** into an array of strings each of length maxchars.	
`fc_setval (ae_string, value)` Writes the value down to the attribute(entity) specified in the `ae _string`	
`fc_setstr (ae_string, value_string)` Writes the string down to the attribute(entity) specified in the `ae_string`