

Table of Contents

10.	Meta Scripts	10-5
10.1	All About Meta Scripts.....	10-5
10.2	Meta Script Structure	10-5
	Components of a Meta Script.....	10-7
10.3	Most Common Meta Script Tools	10-8
	Most common meta script tools.....	10-9
10.4	Designing Meta Scripts.....	10-10
10.5	Entering and Executing the Meta Scripts	10-10
	Mshell (meta shell) and wmshell	10-11
	Error Handling	10-12
	Documentation	10-12
10.6	Variables.....	10-14
	Variable types summary	10-14
	Variable Definitions.....	10-15
10.7	System Variables.....	10-17
10.8	Global and Local Variables.....	10-18
	Global and Local Variables (How it works).....	10-19
	GLOBAL, LOCAL and not-specified variables (Suggested Rules)	10-20
10.9	Action Statement	10-21
	LET	10-22
	SET ATTRIBUTE Command.....	10-23
	EXECUTE	10-23
	DISPLAY	10-25
	SET PRINTER TO	10-28
	PRINT and PRINTLN	10-29
	EXIT WINDOW.....	10-32
	EXIT.....	10-32
	SEND <message> TO	10-32
	SET ...FROM ATTRIBUTE	10-39
	SET ...FORMAT.....	10-40

SET POINTER	10-42
SET MAXIMUM PROGRAM TIME	10-43
10.10 Control Statements.....	10-44
IF ELSE.....	10-45
REPEAT UNTIL	10-47
WHILE	10-48
BREAK	10-48
CONTINUE.....	10-49
RETURN.....	10-49
RUN	10-50
SLEEP.....	10-51
TERMINATE.....	10-52
LOOP THROUGH.....	10-52
10.11 Database Functions.....	10-53
INSERT	10-54
UPDATE	10-56
DELETE.....	10-57
FETCH.....	10-58
LOOP THROUGH.....	10-61
CREATE VIEW and DROP VIEW	10-63
CREATE VIEW FROM FILE <filename>	10-69
CREATE TABLE VARIABLE	10-71
CREATE HISTORICAL VIEW.....	10-73
DBERR and DBERRSTR	10-77
DB Error Codes and Strings	10-79
10.12 Operators and Functions	10-80
Operators	10-81
Special <i>MacroView</i> functions.....	10-83
Miscellaneous Functions	10-84
String Related Functions.....	10-86
Trigonometric, Hyperbolic and Logarithmic Functions	10-89
Binary Arithmetic Functions	10-90
10.13 Time and Date Functions	10-91

Time, Date and Duration Concepts10-92

Combining Different Formats10-93

DATE and TIME Functions10-94

Date and Time Operators.....10-101

Date and Time Related Set Actions10-103

Time Stamp and Duration Format String Options.....10-103

List of Tables

Table 1: Example Structural Components 10-5

Table 2: Methods to enter Meta Script 10-10

Table 3: Variable types summary..... 10-14

Table 4: Variable Definitions 10-15

Table 5: System Variables 10-17

Table 6: Global and Local Variables 10-18

Table 7: The LET Command 10-22

Table 8: The LET Command 10-23

Table 9: The EXECUTE Command 10-23

Table 10: The DISPLAY Command 10-25

Table 11: The DISPLAY Command 10-26

Table 12: The DISPLAY Command 10-27

Table 13: The SET PRINTER TO Command 10-28

Table 14: The SET PRINTER TO Command 10-29

Table 15: The SET PRINTER TO Command 10-29

Table 16: The SET PRINTER TO Command 10-31

Table 17: EXIT WINDOW Command 10-32

Table 18: The EXIT Command 10-32

Table 19: The SEND <message> TO Command 10-32

Table 20: The SEND <Message> TO Command 10-34

Table 21: The SEND <Message> TO Command 10-35

Table 22: The SEND <Message> TO Command 10-36

Table 23: The SEND <Message> TO Command 10-37

Table 24: The SEND <Message> TO Command 10-38

Table 25: The SET ...FROM ATTRIBUTE Command 10-39

Table 26: The SET FORMAT Command 10-40

Table 27: The SET FORMAT Command 10-41

Table 28: The SET POINTER Command 10-42

Table 29: The SET MAXIMUM PROGRAM TIME Command 10-43

Table 30: The IF ELSE Command 10-45

Table 31: The IF ELSE Command 10-46

Table 32: The REPEAT UNTIL Command 10-47

Table 33: The WHILE Command 10-48

Table 34: The BREAK Command 10-48

Table 35: The CONTINUE Command 10-49

Table 36: The RETURN Command 10-49

Table 37: The RUN Command 10-50

Table 38: The RUN Command 10-51

Table 39: The SLEEP Command 10-51

Table 40: The TERMINATE Command 10-52

Table 41: The LOOP THROUGH Command 10-52

Table 42: The INSERT Command 10-54

<i>Table 43: The INSERT Command</i>	10-55
<i>Table 44: The UPDATE Command</i>	10-56
<i>Table 45: The DELETE Command</i>	10-57
<i>Table 46: The FETCH Command</i>	10-58
<i>Table 47: The FETCH Command</i>	10-59
<i>Table 48: The FETCH Command</i>	10-60
<i>Table 49: The LOOP THROUGH Command</i>	10-61
<i>Table 50: The LOOP THROUGH Command</i>	10-62
<i>Table 51: The CREATE VIEW and DROP VIEW Command</i>	10-63
<i>Table 52: The CREATE VIEW and DROP VIEW Command</i>	10-64
<i>Table 53: The CREATE VIEW and DROP VIEW Command</i>	10-65
<i>Table 54: The CREATE VIEW and DROP VIEW Command</i>	10-66
<i>Table 55: The CREATE VIEW and DROP VIEW Command</i>	10-67
<i>Table 56: The CREATE VIEW and DROP VIEW Command</i>	10-68
<i>Table 57: The CREATE VIEW FROM FILE <filename> Command</i>	10-69
<i>Table 58: The CREATE TABLE VARIABLE Command</i>	10-71
<i>Table 59: The CREATE TABLE VARIABLE Command</i>	10-72
<i>Table 60: The CREATE HISTORICAL VIEW Command</i>	10-73
<i>Table 61: The CREATE HISTORICAL VIEW Command</i>	10-74
<i>Table 62: The CREATE HISTORICAL VIEW Command</i>	10-75
<i>Table 63: The CREATE HISTORICAL VIEW Command</i>	10-76
<i>Table 64: The DBERR and DBERRSTR Command</i>	10-77
<i>Table 65: The DBERR and DBERRSTR Command</i>	10-78
<i>Table 66: DB Error Codes and Strings</i>	10-79
<i>Table 67: Operators</i>	10-81
<i>Table 68: Operators</i>	10-82
<i>Table 69: Special MacroView Functions</i>	10-83
<i>Table 70: Miscellaneous Functions</i>	10-84
<i>Table 71: String Related Functions</i>	10-86
<i>Table 72: Trigonometric Functions</i>	10-89
<i>Table 73: Binary Arithmetic Functions</i>	10-90
<i>Table 74: Date and Time Functions</i>	10-94
<i>Table 75: Date and Time Operators</i>	10-101
<i>Table 76: Date and Time Related Set Actions</i>	10-103
<i>Table 77: Time Stamp and Duration Format String Options</i>	10-103

10. Meta Scripts

Meta scripts are programs that are written in the meta script language and are run from within the *MacroView* environment. This is the *preferred* programming method in *MacroView*. The next chapter "Programs" describes other programming tools that are now used less often.

10.1 All About Meta Scripts

The meta script language consists of commands that have been tailored to be most useful in the process control and SCADA environment.

The following meta script features are considered most important:

- **Integration into *MacroView***

The programs may be embedded in the *MacroView* graphics structure. You may perform useful tasks without having to leave the *MacroView* environment.

- **Real time and Historical Functions**

You may access, manipulate and control real time and historic data using the meta script commands.

- **Database Functions**

There is a set of powerful SQL-like database commands that enable such tasks as filtering, searching, appending records etc.

- **Graphical Functions**

You may embed the meta scripts in the CAD diagrams to perform custom presentation tasks such as setting the colour of objects based on real time data.

- **Engineering Language**

Above all, the meta scripts language is ideally suited for engineers.

It is possible to achieve extremely useful functions quickly without the need for extensive programming training.

Unlike programming languages like C and Fortran, a useful control function can be achieved with as few as 2 or 3 statements.

10.2 Meta Script Structure

The best way to learn about meta scripts is to look at an actual example. The example on the next page shows the major components of a typical meta script.

This meta script is executed when the opacity exceeds the regulation limit. It carries out the following functions:

Table 1: Example Structural Components

Function	Introduction	Type of Instruction
Checks to see if the boiler is on-line.	IF	Control Statement.

Function	Introduction	Type of Instruction
Sets a valve in manual and shuts it.	LET	Action.
Calculates the exceedance variable.	-	Operator.
Prints a message to Standard Output.	PRINT	Action.
INSERTS a record in a database and updates the record date and time.	INSERT DATE, TIME	Data base Command. Date & Time functions.

It is important to understand the different types of instructions because the documentation is organized according to these functions. I.e. There is a section on Control Statements; a section on Actions, etc.

Components of a Meta Script

```

// Reaction to an opacity exceedance
IF (BOILER1.ONLINE = 1)
{
  LET FW109.LS = 'MAN';
  LET FW109.MV = 0;
  LET excdnce = OPA1.PV - OPA1.PH;
  PRINTLN "Stack 1 Opacity: {OPA1.PV} at {NOW()}";
  INSERT INTO EXCEED1
    FIELDS DT, TM, EXCEED, COMMENT
    VALUES DATE(), TIME(), excdnce, 'Opacity
  Exceedance on Stack 1';
}

```

Comments

Precede the comment with a double forward slash.

Example

This meta script is started off from an alarm. It contains all of the components of a typical functional program.

Variables

May include real time variables like FW109.LS or program variables like excdnce.

Database Commands

SQL-like commands that can be used to store data, retrieve data and filter data either from databases or from historical data.

Time & Date Functions

A series of important time & date functions like DATE, TIME, NOW, DURATION etc.

Control Statements

These affect the flow of the program. In this case, the actions are only to be carried out if the Boiler is on-line. Examples of this type of statement include: IF..ELSE, REPEAT UNTIL & WHILE.

Actions

Statements that do something. These may set a value, print information, send a message, display a window etc.

Operator & Functions

These are used to manipulate information and include operators like +, -, *, /, .AND., .OR., .NOT., as well as functions like ABS(), STR(), VAL(), and RAND().

10.3 Most Common Meta Script Tools

The diagram "Most Common meta script tools" on page5 provides further insight into the meta script language.

Here we have grouped the most common and useful commands from the various statement types. In addition to those listed below, there are a large number of extra features and tools that can be employed to provide even more powerful meta scripts.

If you are just starting with meta scripts, we do however recommend that you become thoroughly familiar with a limited number of commands such as those listed below before progressing with the much larger set of commands.

Most common meta script tools

ACTIONS	<pre>LET varName = <exprn>; DISPLAY WINDOW <file> LET ent.attr=<exprn>; PRINTLN <exprn>;</pre>	
PROGRAM CONTROL	<pre>IF(condition)action IF (condition) ; RETURN; ELSE action; { actions; } ELSE { actions; }</pre>	
DATABASE FUNCTIONS	<pre>INSERT INTO <entity> UPDATE <entity> FIELDS <field1>, <field2> SET field1=expr1, field2=expr2; VALUES <value1>, CREATE VIEW <viewName> AS <value2>; SELECT <selectSpec> FROM <entity> WHERE "< filterSpec>" ORDER BY <orderSpec>; FETCH NEXT OF <entity> WHERE "<exprn>";</pre>	
OPERATORS	<pre>*/+- = # > >= < <= .AND. .OR. .NOT.</pre>	FUNCTIONS
		<pre>ABS() STR() FLASH() SUBSTR() INT() VAL() RAND() CHR()</pre>
HISTORICAL VIEWS	<pre>CREATE HISTORICAL VIEW <tableName> AS SELECT <fieldName1>=ent.attr1, <fieldName2>=ent.attr2 WITH <period> (<duration>) <hForm> WHERE TIME >= <startTstamp> AND TIME <= <endTstamp>;</pre>	TIME & DATE
		<pre>DATE DATESTR DURATION DURATIONSTR NOW TIME TIMESTR SET DATE FORMAT TO "MM:DD:YY" -, +, =, >, <, #</pre>

10.4 Designing Meta Scripts

When designing your meta scripts, you should follow the normal guidelines that you would use for any programming language. In particular:

- First decide on the structure of the meta scripts and how these meta scripts are to achieve the desired functionality.
- Make the meta script as clear and simple as possible. (Avoid "clever" programming.) The program should be readable and understandable if read over the telephone.
- Use variable names that are easily understood. (A well chosen variable name is often more valuable than a comment.)
- We strongly advise that you keep the size of the meta scripts to less than 10 lines. Longer programs are better done when the customer has more experience or has attended the advanced programming class.
- If you are having problems debugging the meta script, we recommend you use the meta script tracer described in the graphics chapter. This tool provides a clear indication of what is happening in the meta script.

10.5 Entering and Executing the Meta Scripts

The executable file which is used to execute meta script commands is called *mshell*, in the UNIX system, and *wmshell.exe*, in the Windows NT system. There are a number of ways that you can enter and execute the meta scripts as the table below indicates:

Table 2: Methods to enter Meta Script

Environment	Entering the Meta Script	Starting the Meta Script
The version III graphics environment.	Type the meta script into the Graphic Object Editor. Note: see the graphics section.	Typically, the meta script is started from a button or is executed continuously while the display is active.
From the Unix prompt. A UNIX script can be initiated from the UNIX prompt or from the alarm manager for example.	You can create the meta script with any text editor and save it in a file in the UNIX system.	You can start the execution of the meta script by using the <i>mshell</i> command. I.e. <code>mshell <scriptname>.</code>
From a DOS prompt in Windows NT A DOS batch file can be initiated from the DOS command line or from the Alarm manager, for example.	You can create the meta script with any text editor and save it in a file in the NT system	You can start the execution of the meta script by using the <i>wmshell</i> command. I.e. <code>wmshell <scriptname>.</code>

Mshell (meta shell) and wshell

The `mshell` and `wshell.exe` are programs, for UNIX and NT systems respectively, which can be used to run a meta script without starting the operation program `xops3`.

Use: It is generally used where no direct graphical output is required. E.g.

- i. Where the meta script is to run in the background. (E.g. for simulators.)
- ii. For text based reports.
- iii. For complex database manipulation scripts such as environmental programs etc.

NOTE: The `mshell` program, like all *MacroView* programs, must be started from the configuration directory. This is usually set up through the `MACRODIR` environment variable.

Starting the mshell or wshell: There are two ways you can use the `mshell` and `wshell`.

- i. **Interactive mode for UNIX:** Enter the meta script commands directly onto the screen. To execute the program, hit the ***CNTRL-D keys together*** when the program is fully entered. E.g.

`cd $MACRODIR` (This is normally set to the configuration directory.)

`mshell -l` (This will give you a flashing cursor, from where you can type in meta script commands)

`PRINTLN NOW();`

CNTRL-D

Tue 23rd Sept 8:45am

Hitting the ENTER key will return you back to the command line prompt.

Background mode for UNIX: In the `mshell` command, specify the filename of the meta script file to be executed. I.e.

`mshell <fileName> &` this will start the `mshell` program and execute it in the background.

- ii. **Interactive mode for NT:** Enter the meta script commands directly onto the screen. To execute the program, hit the ***ENTER Key*** when the program is fully entered. E.g.

`cd %MACRODIR%` (This is normally set to the configuration directory.)

`wshell -l` (This will give you a flashing cursor, from where you can type in meta script commands)

`PRINTLN NOW();`

ENTER

Tue 23rd Sept 8:45am

Hitting the ENTER key, again will return you back to the command line prompt.

mshell and wshell arguments: There are various arguments that can be used in the `mshell` command line (both interactive and background modes.) I.e.

- `-init var1=xxx var2=xxx ...`

This allows various variables to be set to initialized values.

E.g. `mshell -init lineStatus=Healthy`

- `-ptr LABEL1=ENTITY1 LABEL2=ENTITY2 ...`

This allows various pointers to be set to Entity names.

E.g. `mshell -ptr LABEL1=FIC100`

- `-v`

This displays all the available options.

Error Handling

All *MacroView* errors that occur are logged in a dedicated table variable called ERRLOG. The ERRLOG table variable has four attributes called TM, DT, MSG and CODE. All of the attributes are strings with the exception of the CODE, which is numeric. The table is limited to 1024 entries.

You can trap for an occurrence of an error using the following command:

```
ON ERROR RUN <metascript>;
```

Where `metascript` is a meta script that evaluates to an “ascii” string. The string defines a meta script filename which is executed when the error is trapped. Typically, the meta script program would:

- i. Examine the contents of the current record in the ERRLOG table variable and
- ii. Cause some graceful shutdown of the program.

Alternatively, if you are using the Navigator, you can call up the internal alarm summary, which will show these messages:

```
mshell (meta shell) and wshell.exe
```

Documentation

The next sections provide a more formal overview of the meta script language.

The sections include:

- **Variables**

The rules associated with choosing variable names.

- **Action statements**
Statements that do something.
E.g. `LET`, `PRINT`, `DISPLAY WINDOW` etc.
- **Program control**
Where you can select which parts of the script are executed.
E.g. `IF`, `ELSE` statements.
- **Database Functions**
These are SQL-like statements that enable you to easily work with databases.
E.g. `INSERT`, `FETCH`, `CREATE VIEW`
- **Historical Views**
Enable you to treat historical data as if it were in a database.
- **Operators & Functions**
These manipulate the values. Examples include:

Operators:	<code>+</code> <code>-</code> <code>*</code> <code>/</code>
	<code>=</code> <code><</code> <code>></code> <code>#</code>
	<code>.AND.</code> <code>.NOT.</code> <code>.OR.</code>
Functions:	<code>ABS()</code> , <code>INT()</code> , <code>RAND()</code> .
String Functions:	<code>VAL()</code> , <code>CHR()</code> , <code>SUBSTR()</code> .
Trigonometric:	<code>SIN()</code> , <code>COS()</code> , <code>TAN()</code> .
Binary:	<code>BIT()</code> , <code>BINAND()</code> , <code>BINOR()</code> .
- **Date & Time Functions**
Functions that relate to the manipulation of date and time information.
E.g. `DATE`; `TIME`; `NOW`; `DURATION` etc.

10.6 Variables

The table below is a summary of the various types of variables available for use in the *MacroView* system. These are discussed in more detail in Table3, "Variable Definitions," on page10.

Variable types summary

Table 3: Variable types summary

Variable Type	Example	Descriptions
User Variables	<code>temp,</code> <code>numDays</code>	These variables may be created and used freely in meta scripts. There are LOCAL and GLOBAL variables. See "Global and Local Variables" on page13.
System Variables	<code>HalfGreen,</code> <code>UserName</code> <code>SetRequest</code>	These variables are reserved by the system and have specific meanings as discussed in the section "System Variables" on page12.
<code>ENTITY.ATTRIBUTE</code> <code>ATTRIBUTE (ENTITY)</code> (See Note **)	<code>FIC100.PV</code> <code>PV(FIC100)</code>	These variables are linked to the field via the entity definitions. Their content varies in sympathy with the field variables. The <i>preferred</i> format is <code>ENTITY.ATTRIBUTE</code> . All entities and attributes should be upper case.
Table Variables	<code>LABDATA</code>	These are tables that exist in memory but not on the disk. They are generally created for convenience at the start of the program and discarded once the program is over.

**** NOTE: Support for the form `ATTRIBUTE (ENTITY)` will be discontinued sometime in the future. We strongly encourage users to adopt the `ENTITY.ATTRIBUTE` format for this reason.**

Variable Definitions

Table 4: Variable Definitions

Variable	Description
<p>USER VARIABLES</p> <p>Format</p> <p>Scope</p> <p>Restriction</p> <p>convention</p>	<p>These variables may be used freely in meta scripts.</p> <p>You may make the variables any length although we recommend a length of less than 20 characters.</p> <p>There are LOCAL and GLOBAL Variables. See the next sections on when to use LOCAL and GLOBAL variables. LOCAL variables are available for use by the window and it's children.</p> <p>GLOBAL variables are available in every application used by that operational program.</p> <p>You may not choose a variable name that is the same as a system variable name, e.g. Black.</p> <p>By convention, variables.</p> <p>Start with a lower case letter.</p> <p>Have an uppercase letter at the start of a new word within the variable (e.g. numBatch) and</p> <p>Where the word is too long, abbreviations are created by eliminating the vowels in the word.</p> <p>E.g for testbed control, tstBdCtrl.</p> <p>temp, tankAlarm, floMax, opctyLm.</p> <p>Note: That there is no maximum size of variables.</p>
<p>SYSTEM VARIABLES</p> <p>Convention</p> <p>Restriction</p> <p>Examples</p>	<p>These variables are reserved by the system and have specific meanings as discussed in the next table "System Variables."</p> <p>By convention, system variables start with a capital letter.</p> <p>E.g. Black, UserName.</p> <p>You may not write to system variables.</p> <p>You may not create your own variables with the same name as a system variable.</p> <p>HalfMagenta, ConsoleType.</p> <p>For a full list of user accessible system names, Refer to Table1, "System Variables" on page12.</p>
<p>ENTITY.ATTRIBUTE</p> <p>format</p>	<p>These variables are linked to the real-time data via their entity definitions. Their content varies in sympathy with the real-time values.</p> <p>E.g. <code>FIC100.PV</code>, <code>LAB1.DT</code></p> <p>This is the <i>preferred</i> format for referencing real-time and database type</p>

10.7 System Variables

The table below summarizes the System Variables that are automatically available when you start *MacroView*. The first letter of the system variables (and variables used by *MacroView* applications like the Navigator) are capitalized to distinguish them from the user variables. To avoid a clash with a system variable, ensure that your variables start with a lower case letter.

Table 5: System Variables

Name	Value	Description
Black	0	The color index representing the color black.
White	1	The color index representing the color white.
HalfRed	2	The color index representing the half-red color.
HalfGreen	3	The color index representing the half-green color.
HalfBlue	4	The color index representing the half-blue color.
HalfYellow	5	The color index representing the half-yellow color.
HalfCyan	6	The color index representing the half-cyan color.
HalfMagenta	7	The color index representing the half-magenta color.
Grey (Gray)	8	The color index representing the color grey.
DarkGrey (Gray)	9	The color index representing the color dark grey.
Red	10	The color index representing the color red.
Green	11	The color index representing the color green.
Blue	12	The color index representing the color blue.
Yellow	13	The color index representing the color yellow.
Cyan	14	The color index representing the color cyan.
Magenta	15	The color index representing the color magenta.
UserAccessLevel	0 - 99	Access level assigned to this user from the users configuration.
ConsoleAccessLevel	0 - 99	Access level assigned to this console from consoles configuration.
AccessLevel	0 - 99	The access level of this console and user combination.
ConsoleName		The name of this console.
ConsoleType		The type of console (e.g. UNIX).
SystemId		The host name of this <i>MacroView</i> system.
PortName		The name of the device that connects the user to the system.
UserName		The name of the user running this application.
SetRequest		Used internally in object to hold requested value until Validation

Name	Value	Description
		Program returns 1.

10.8 Global and Local Variables

You may define variables to be Local or Global. The table below summarizes the differences in functionality between these two types of variable.

Table 6: Global and Local Variables

Global or Local		Comments
GLOBAL	Definition	Global Variables are available to all applications in the currently running operational program. The GLOBAL statement defines them. E.g.
	Example	<code>GLOBAL myApplicationStatus;</code>
	Usage	Avoid using the global statement wherever possible. It is just possible that someone else could choose the same Global name that you have chosen resulting in confusion and interaction with their program. Use GLOBAL variables only for variables which must be accessed across all windows and meta script programs.
	Optional start-up	To be backward compatible with previous versions that did not support LOCAL and GLOBAL variables, you may start off the operational program using the <code>-defaultGlobal</code> argument. This will treat all variables as GLOBAL unless explicitly defined as LOCAL.
LOCAL (Using the LOCAL command)	Definition	Local variables are available to the current <i>window and any children of that window</i> . I.e. the variable will be available to DISPLAY AREAS, meta scripts called from a window etc.
	Example	<code>LOCAL phase1;</code>
	Usage	Wherever possible use local variables. Note that if the LOCAL command is encountered, the system will <i>not</i> look in the parent for the same variable. Variables are LOCAL if: <ul style="list-style-type: none"> i. They are defined LOCAL using the LOCAL declaration. ii. They have no declaration in the meta script but the search up the parent tree encounters a LOCAL declaration. iii. They encounter no GLOBAL or LOCAL declarations at all up

Global or Local		Comments
		the parent tree.

Global and Local Variables (How it works)

The system uses the following algorithm to determine whether to use a LOCAL or

GLOBAL variable:

- i. Whenever a variable is encountered in a meta script, a search is made back up the meta script and into the parent windows for a declaration.
- ii. The system uses the **first** declaration it finds. I.e. if a LOCAL declaration is found that will be used, if a Global declaration, then that variable will be used.
- iii. If there are no declarations in the meta script (or the parents of the meta scripts), then the variable will be deemed LOCAL to that meta script.
- iv. The **exception** is that if the `-defaultGlobal` argument is used in the start up command, in which case ALL variables are deemed GLOBAL.

Clearly, from the above, it is possible to have several LOCAL variables of the same name in different meta scripts and one GLOBAL variable.

Technical Note: If you specify LOCAL variables in the INIT EARLY or INIT LATE meta scripts, then since these meta scripts are the first meta scripts encountered in a graphic, the variables will be available to all the meta scripts and children windows by inheritance.

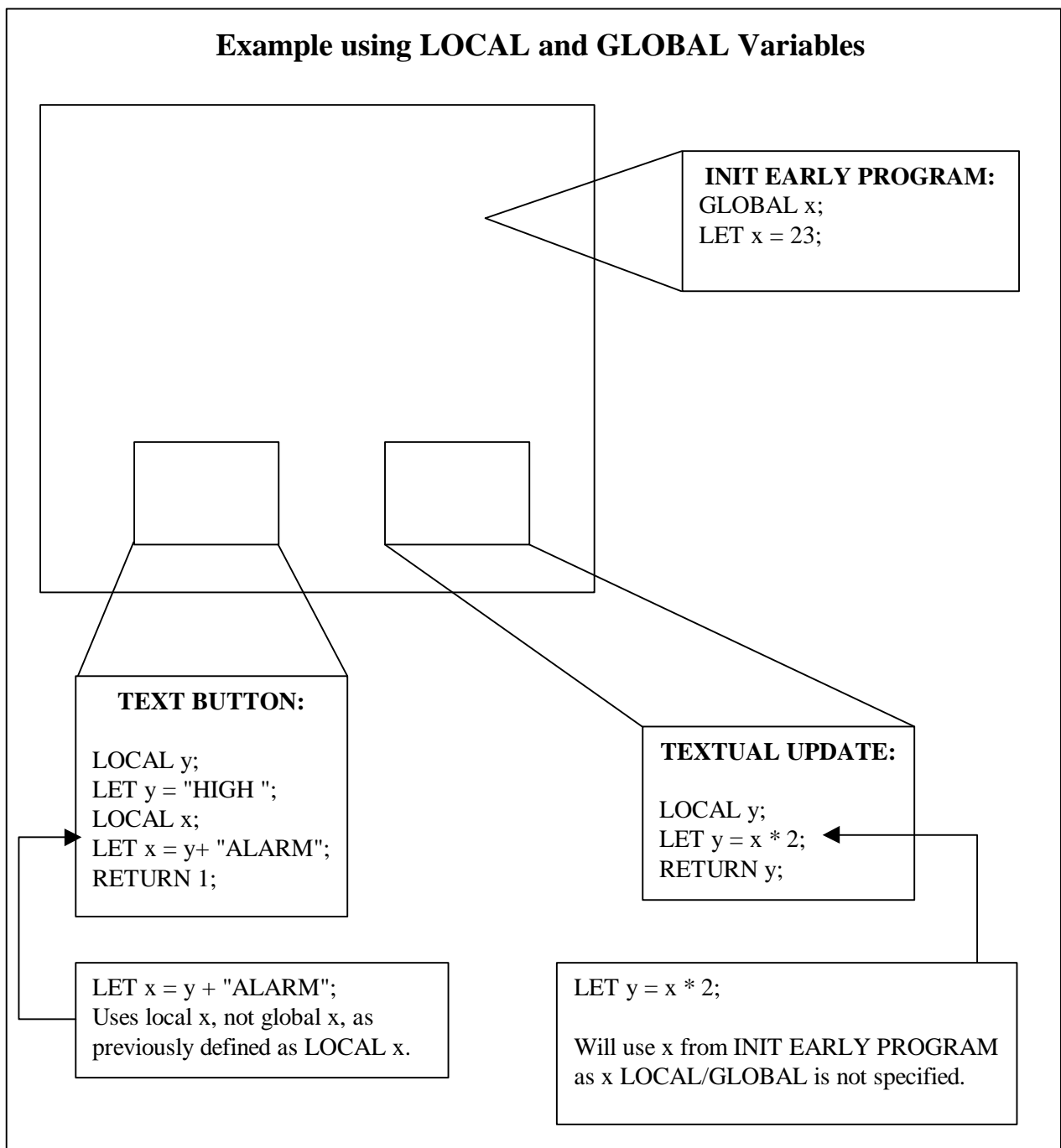
See the note on the next page about suggested rules for using GLOBAL and LOCAL variables.

GLOBAL, LOCAL and not-specified variables (Suggested Rules)

We advise using the following rules to choose whether to use GLOBAL or LOCAL variables.

- **Only** use GLOBAL if the variables are to be used by other applications.
- Use LOCAL if the variable has no use outside this meta script.
- Do not specify LOCAL or GLOBAL if a search through the parent is required.

(Refer to the diagram below.)



10.9 Action Statement

Action statements are statements that do something.

For example, LET, PRINT etc. The diagram below is a summary of the main action statements. These statements are discussed in more detail in the next pages of this section.



LET

Also see "SET ATTRIBUTE Command" on the next page.

Table 7: The LET Command

Statement	Description	
LET	The LET statement assigns a value to a variable or the attribute of an entity. The attribute of an entity.	
Format 1	<p>There are various forms of the LET statement as shown below:</p> <p><i>form</i></p> <pre>LET <ent>.<attrib>=<exprn>;</pre> <p>The LET statement simply calculates the expression and assigns it to the entity.attr pair.</p> <p><i>example</i></p> <pre>LET FW109.PV =350; LET FW109.LS="MAN"; IF (FW109.LS="CAS") LET FA109.SV= ratio * FW109.PV;</pre>	
Format 2	<p><i>form</i></p> <pre>LET <variable> =<exprn>;</pre> <p>The LET statement simply calculates the expression and assigns it to the variable.</p> <p><i>example</i></p> <pre>LET dateFlag = "T"; LET maxNumBatches = 200;</pre>	
Format 3	<p><i>form</i></p> <pre>LET attr(entity) =<exprn>;</pre> <p>This is just an alternative way that you can specify the attribute entity pair.</p> <p>Note: Support for this format will be discontinued some time in the future. We strongly recommend you use the first format.</p> <p><i>example</i></p> <pre>LET TM(BATCH9) = "6:45"; LET SV(FIC100) = 0;</pre>	
Write Delay Implications	<p>Note:</p> <p>The LET statement does not wait for the command to complete before continuing to the next statement. It is important to note that it may take some finite time for the write to field variables to take place. Where it is important to wait until the action has completed, just set up a loop that reads the value back until it sees that the change has been made or just use the SLEEP command.</p>	

SET ATTRIBUTE Command

Also see LET on page 10-22.

Table 8: The LET Command

Statement	Description
SET ATTRIBUTE	The SET ATTRIBUTE statement assigns a value to an attribute string.
format	<p><i>form</i></p> <pre>SET ATTRIBUTE "<ent>.<attrib>" TO <exprn>;</pre> <p>In the example above, the expression "<ent.attr>" must resolve to an ent.attr or variable name. The SET ATTRIBUTE statement allows you to indirectly set the value of an attribute to a variable.</p> <p><i>examples</i></p> <pre>SET ATTRIBUTE "{SPentity}.SV" TO calcSetPoint+userAllowance; SET ATTRIBUTE attrName TO 15.0;</pre>
Write Delay Implications	<p>Note:</p> <p>The LET statement does not wait for the command to complete before continuing to the next statement. It is important to note that it may take some finite time for the write to field variables to take place. Where it is important to wait until the action has completed, just set up a loop that reads the value back until it sees that the change has been made or just use the SLEEP command.</p>

EXECUTE

Table 9: The EXECUTE Command

Statement	Description		
EXECUTE	The EXECUTE statement takes a string expression, resolves it and executes it as a meta script command. In another variation of this statement, you can assign a value to a variable when a string is executed.		
Format 1	<p>There two forms of the EXECUTE statement as shown below:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; width: 50%;"> <p><i>form</i></p> <pre>EXECUTE <string expression>;</pre> <p>The execute command simply executes the string that follows the command</p> </td> <td style="vertical-align: top; width: 50%;"> <p><i>example</i></p> <pre>LET x = "LET a="; EXECUTE(x+ "3;");</pre> </td> </tr> </table>	<p><i>form</i></p> <pre>EXECUTE <string expression>;</pre> <p>The execute command simply executes the string that follows the command</p>	<p><i>example</i></p> <pre>LET x = "LET a="; EXECUTE(x+ "3;");</pre>
<p><i>form</i></p> <pre>EXECUTE <string expression>;</pre> <p>The execute command simply executes the string that follows the command</p>	<p><i>example</i></p> <pre>LET x = "LET a="; EXECUTE(x+ "3;");</pre>		

DISPLAY

Table 10: The DISPLAY Command

Statement	Description
<p>DISPLAY</p>	<p>The DISPLAY command causes the operations program (i.e. xops3) to display a window. There are two major variants of this command:</p> <pre>DISPLAY <exprn>;</pre> <p>Which replaces the window being shown.</p> <pre>DISPLAY WINDOW <exprn>;</pre> <p>Which causes a new window to appear on top of the "parent" window.</p> <p>As the examples show below, you have control of the position and size of the new window. Refer to the diagram below.</p> <div style="display: flex; justify-content: space-around;"> <div data-bbox="421 826 900 1543"> <p>DISPLAY "metafile 2";</p> <p>The new display replaces the old display.</p> </div> <div data-bbox="924 826 1426 1543"> <p>DISPLAY WINDOW "metafile 2";</p> <p>You have control of the size and position of the new window.</p> </div> </div>

Note:

It is important that you use the correct pathname for the metafile being used in these commands. If these commands are used within the *MacroView* Navigator, it should be remembered that the Navigator is started from the MACRODIR directory, this is normally set to the configuration directory. E.g.

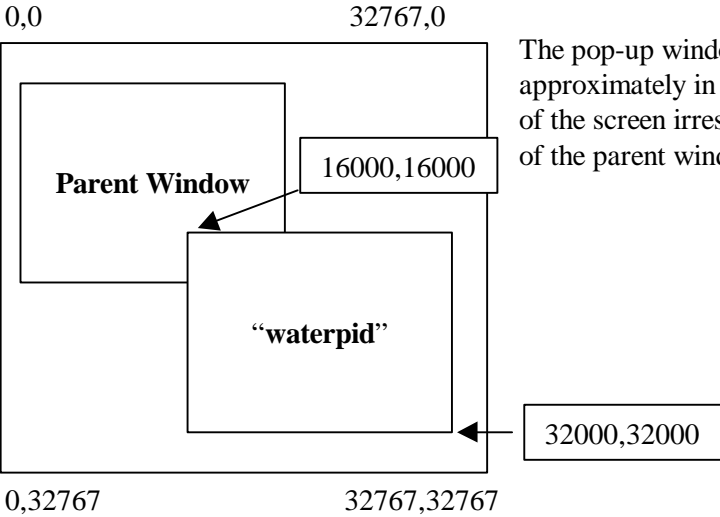
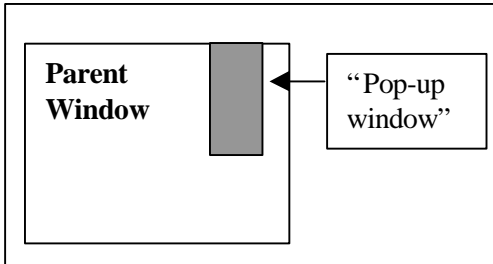
`/u/macro/config`– for SCO UNIX systems, and

`C:\users\macro\config`– for NT systems.

Table 11: The DISPLAY Command

Statement	Description	
arguments <code><file></code> FRAME	<code><file></code> <p>This is a metafile name. You may include an absolute path name or a path name relative to the directory of the parent window. You may use an expression that evaluates to a metafile name but it must evaluate to a string.</p> <p>FRAME</p> <code><x1>, <y1>, <x2>, <y2></code> <p>These are the top left co-ordinates (<code><x1></code>, <code><y1></code>) and the bottom right co-ordinates. (<code><x2></code>, <code><y2></code>.) They use VDC co-ordinates, i.e. the top left-hand corner is 0,0 and the bottom right hand corner is 32767, 32767.</p> <p>Note: If the new window is modal, it will be centered.</p> <p>Refer to header object in the Graphics Chapter.</p>	
Form 1	<i>form</i> <code>DISPLAY <file>;</code>	<i>example</i> <code>DISPLAY "tb_schem";</code> This example simply replaces the existing screen with the screen defined by the metafile <code>tb_schem</code> .
Form 2	<i>form</i> <code>DISPLAY WINDOW <file>;</code>	<i>example</i> <code>DISPLAY WINDOW "hlppmp43";</code> A pop-up window called "hlppmp43" appears over the existing window.
open file error	If the file specified cannot be found, xops3 will look for a file called " <code>openerr.dgt</code> " to display instead. This " <code>openerr.dgt</code> " can be used to display an error message indicating that a file could not be found.	

Table 12: The DISPLAY Command

Statement	Description	
<p>Form 3</p>	<p><i>form</i></p> <pre>DISPLAY WINDOW <file> FRAME <x1>, <y1>, <x2>, <y2> RELATIVE TO SCREEN;</pre> 	<p><i>example</i></p> <pre>DISPLAY WINDOW "waterpid" FRAME 16000, 16000, 32000, 32000 RELATIVE TO SCREEN;</pre> <p>The pop-up window will appear approximately in the lower right quarter of the screen irrespective of the position of the parent window.</p>
<p>Form 4</p>	<p><i>form</i></p> <pre>DISPLAY WINDOW <file> FRAME <x1>, <y1>, <x2>, <y2>;</pre> 	<p><i>example</i></p> <pre>DISPLAY WINDOW "iconbar" FRAME 20000, 0, 22000, 25000;</pre> <p>The pop-up window will be positioned relative to the origin of the <i>parent</i> window by the amounts given in the FRAME command.</p>

SET PRINTER TO

Table 13: The SET PRINTER TO Command

Statement	Description	
SET PRINTER TO	<p>The SET PRINTER TO command is used to set up the name of the device that the next PRINT or PRINTLN statements (see the next section "PRINT and PRINTLN" on page26) are to send their output to. Once the device is defined, the outputs of the subsequent print statements go to that device until the next SET PRINTER TO statement.</p> <p>There are five main kinds of SET PRINTER TO statements:</p>	
1. STDOUT (This is the default)	<p><i>Format</i></p> <pre>SET PRINTER TO STDOUT;</pre>	<p><i>Output goes to</i></p> <p>The standard output of xops3. This is typically a console or terminal X window.</p>
2. Spooling device	<p><i>format</i></p> <pre>SET PRINTER TO <exprn>;</pre> <p><i>example</i></p> <pre>SET PRINTER TO "lp";</pre>	<p><i>Output goes to</i></p> <p>The spooling device called <exprn>.</p> <p>The output is sent to the printer spooler called lp.</p>
3 File	<p><i>format</i></p> <pre>SET PRINTER TO FILE "<exprn>;"</pre> <pre>SET PRINTER TO APPEND TO FILE "<exprn>;"</pre> <p><i>examples</i></p> <pre>SET PRINTER TO FILE "../report/postMtm";</pre> <pre>SET PRINTER TO APPEND to FILE "../messages/eventLog";</pre>	<p><i>Output goes to</i></p> <p>A file called <exprn>.</p> <p>The end of the file called <exprn>.</p> <p>The output is written to a file called postMtm in the directory ../report.</p> <p>The output is added to the end of the file eventLog in the ../messages directory.</p>

Table 14: The SET PRINTER TO Command

Statement	Description	
4. Program input	<p><i>format</i></p> <pre>SET PRINTER TO INPUT OF "<exprn>";</pre> <p><i>examples</i></p> <pre>SET PRINTER TO INPUT OF "mail macro";</pre>	<p><i>Output goes to</i></p> <p>The input of a program called <code><exprn></code>.</p> <p>The outputs of the print statements are sent to the input of the program called mail. The program sends a mail message to the user macro.</p>
Note	<p>In <i>all</i> of the above statements, <code><exprn></code> must evaluate to a string.</p>	
Note	<p>In the case of the spooling device format, the program input format and the display window format, the printer output is only actually sent when the meta script is completed.</p>	
Note	<p>For examples of how the SET PRINTER TO statement is used with the PRINT and PRINTLN statement, refer to the next section on the PRINT statement.</p>	

PRINT and PRINTLN

Table 15: The SET PRINTER TO Command

Statement	Description	
PRINT and PRINTLN	<p>These statements print information to the device that has been specified by the SET PRINTER TO statement.</p>	
argument	<p><i>format</i></p> <pre>PRINT <exprn>;</pre> <p>Send the string <code><exprn></code> to the print device.</p> <pre>PRINTLN <exprn>;</pre> <p>Send the string <code><exprn></code> to the print device and append a new line character.</p>	
	<p>The expression must evaluate to a string. The string is then sent to the selected print device. There are a number of string-related functions described later in this chapter that can be used to create the required content and format of the string.</p>	

Statement	Description	
example	<p><i>example</i></p> <pre>PRINTLN;</pre>	<p>This simply causes a new-line character to be sent to the printer device.</p>
example	<p><i>example</i></p> <pre>PRINT "Level={LI109.PV3} %";</pre> <p>Note: See the section "Operators and Functions" on page76 on curly brackets.</p> <p>The evaluation of string variables can be simplified with the curly bracket notation. I.e. { and }. The curly brackets show parts of the expression that must be evaluated first. Use the curly brackets to evaluate the variables or entity.attributes in a string first. For example:</p> <pre>LET attrSet= "TYPE = '{TypeStr}' .AND. ATTRIBUTE = '{UPPER(SetRequest) '";</pre> <p>The { } have the highest precedence and are evaluated left to right prior to the remainder of the expression.</p> <p>You may only use the curly brackets inside the double quotes. They are unaffected by single quotes.</p>	<p>This results in a print out of a string like</p> <pre>Level=36.9432 %</pre> <p>Note: that there will not be a line feed after this statement has been printed. The next print string will start on the same line.</p>

Table 16: The SET PRINTER TO Command

Statement	Description																		
example	<p>example</p> <pre>PRINTLN "Level={LI109.PV3} %";</pre> <p>This results in a string like LEVEL=36.94 % being printed. PRINTLN means a new line will be started with the next print statement.</p>																		
example	<p>example</p> <pre>SET PRINTER TO "lp"; PRINTLN "BATCH" "\t"+ "Required"+ "\t"+ "Measured"; PRINTLN; LOOP THROUGH batView { PRNTLN BAVIEW.BAT+ "\t"+ BATVIEW.RTRQ+ "\t"+ BATVIEW.RTMS; }</pre> <p>The printer will send it output to the device called "lp".</p> <p>First a title is printed. The \t characters cause a tab of 5 characters to be made between the words. You can also use \n for the newline character.</p> <p>The program loops through the database and prints a line for each record. The values are once again separated by the tab marks.</p> <p>Note: The printout would look like this:</p> <table border="0" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>BATCH</th> <th>Required</th> <th>Measured</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>12.3</td> <td>12.4</td> </tr> <tr> <td>2</td> <td>12.5</td> <td>12.5</td> </tr> <tr> <td>3</td> <td>9.0</td> <td>8.9</td> </tr> <tr> <td>4</td> <td>21.3</td> <td>21.7</td> </tr> <tr> <td>etc.</td> <td></td> <td></td> </tr> </tbody> </table>	BATCH	Required	Measured	1	12.3	12.4	2	12.5	12.5	3	9.0	8.9	4	21.3	21.7	etc.		
BATCH	Required	Measured																	
1	12.3	12.4																	
2	12.5	12.5																	
3	9.0	8.9																	
4	21.3	21.7																	
etc.																			

EXIT WINDOW

Table 17: EXIT WINDOW Command

Statement	Description
EXIT WINDOW	<p>The exit window causes the currently active window to terminate. It is typically used with the pop-up windows on the OK or CANCEL buttons.</p> <p><i>Note that this will not work from within a DISPLAY AREA.</i></p>

EXIT

Table 18: The EXIT Command

Statement	Description
EXIT	<p>The EXIT <i>object</i> is run whenever a metafile window terminates. The forms editor can not be used to create this object. It must be created in the CAD diagram as follows:</p> <ul style="list-style-type: none"> • Type the modifier #EXIT in the graphic area. • Add the meta script commands terminated by the semicolon. • Group the #EXIT modifier and all the text that makes up the program. • Convert the CAD diagram using the Convert3 program.

SEND <message> TO

Table 19: The SEND <message> TO Command

Statement	Description
SEND <message> TO	<p>Messages are used as a means of communicating between different objects and different windows. From a meta script, you can send a message to:</p> <ul style="list-style-type: none"> • A window application. • Another object in the same window. • Another object in a different window. <p>To be successful, the receiving object or window must have a matching label or ID and must also be able to understand the message. This understanding may already be built into the object or window or it must be specified in a TRIGGER program.</p>

Statement	Description
	<pre> graph TD subgraph Metafile_Window [Metafile window] O1[Object] O2[Object] end O1 -- "SEND <message> TC" --> TC[TC] O1 -- "SEND <message> TO <metafile:object>" --> O2 O3[Object] -- "SEND <message> TO <metafile>;" --> MF[metafile] MF --> O4[Object] </pre> <p>Objects must be labelled with #ID or "Object Name" and must be able to understand the message.</p>

NOTE:

There are a number of Dynamic Graphic Objects (DGO's) used throughout the *MacroView* system, some of these are specific to the standard *MacroView* Navigator package, whilst others are embedded within the operations program (xops3) itself.

The *MacroView* Navigator contains a number of objects, such as the `MessageCenter` trigger program etc, which can be used to provide functionality. Please refer to the Navigator User Manual (U-NAV-3.1.0) for a complete list of these and a description of their use.

Table 20: The SEND <Message> TO Command

Statement	Description													
format	<p>There are four main formats that can be used with the <code>SEND <message> TO</code> command.</p>													
	<table border="0" style="width: 100%;"> <tr> <td style="width: 30%;"><i>format</i></td> <td style="width: 30%;"></td> <td style="width: 40%;"><i>Message goes to</i></td> </tr> <tr> <td><code>SEND <message> TO <object>;</code></td> <td></td> <td>An object in the same window.</td> </tr> <tr> <td><code>SEND <message> TO <metafile:object>;</code></td> <td></td> <td>An object in the window <code><metafile></code>.</td> </tr> <tr> <td><code>SEND <message> TO <metafile>;</code></td> <td></td> <td rowspan="2">A window called <code><metafile></code>.</td> </tr> <tr> <td><code>SEND <message> TO "MetaFile";</code></td> <td></td> </tr> </table> <p>Note: that you cannot send messages between windows that have been created from different base windows. I.e. you can only send messages between windows that have the same base parent. You cannot send messages from one UNIX program to another.</p>	<i>format</i>		<i>Message goes to</i>	<code>SEND <message> TO <object>;</code>		An object in the same window.	<code>SEND <message> TO <metafile:object>;</code>		An object in the window <code><metafile></code> .	<code>SEND <message> TO <metafile>;</code>		A window called <code><metafile></code> .	<code>SEND <message> TO "MetaFile";</code>
<i>format</i>		<i>Message goes to</i>												
<code>SEND <message> TO <object>;</code>		An object in the same window.												
<code>SEND <message> TO <metafile:object>;</code>		An object in the window <code><metafile></code> .												
<code>SEND <message> TO <metafile>;</code>		A window called <code><metafile></code> .												
<code>SEND <message> TO "MetaFile";</code>														
Arguments	<p><code><message></code></p> <p>This is the string that is sent to the object or window.</p> <p>It can be a simple string like "RESET" or it can contain arguments.</p> <p>If it contains arguments, the general form is:</p> <p><code>"<messageString>(<arg1,arg2,...argN>)"</code></p> <p>The receiving object must understand the message and any arguments that are delivered.</p> <p><code><object></code></p> <p>This is a string containing the object label or ID. The label must be specified using the #ID modifier or entered in the Object Name field in the Graphic Object Editor.</p> <p><code><metafile></code></p> <p>This is a string containing the metafile name of the receiving window.</p> <p>note <code><Message String></code> and <code><Object></code> may contain spaces.</p>													

Table 21: The SEND <Message> TO Command

Statement	Description	
example	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p style="text-align: center;">RESET</p> <p>In a text button enter: SEND "AUT" TO "TNKCTRL";</p> </div> <div style="width: 45%; border-left: 1px solid black; padding-left: 10px;"> <p style="text-align: center;"><i>example</i></p> <p>AUT Action:</p> <pre>LET FW109.LS = "AUT"; LET FA109.LS = "AUT"; LET FM109.LS = "AUT";</pre> </div> </div> <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p style="text-align: center;">RESET</p> <p>In a text button enter: SEND "EMPTY" TO "Tank Cont";</p> </div> <div style="width: 45%; border-left: 1px solid black; padding-left: 10px;"> <p style="text-align: center;">EMPTY Action:</p> <pre>LET FW109.MV=0; LET FA109.MV=0; LET FM109.MV=100;</pre> </div> </div> <p>In the above example, two separate buttons send messages to a Trigger object called "Tank Cont". There are actions AUT and EMPTY in "Tank Cont" that match the messages.</p>	
example using arguments	Refer to the second example in the Trigger widget of the graphics sections.	
example using the in-built functions	<p><i>example</i></p> <pre>SEND "PutText(SEARCH)" TO "myButton"; SEND "SetProgram\n"+ "DISPLAY WINDOW '../qdata/rqsrch' FRAME 8000, 10000, 18000, 19000 RELATIVE TO SCREEN;" TO "myButton";</pre>	<p>In the INIT program, the meta script sends a message PutText with the label of the button "Search" as an argument. This is sent to an object called "myButton."</p> <p>Now a meta script program is sent to the same key.</p> <p>Note: the use of \n to get a line feed.</p>

Table 22: The SEND <Message> TO Command

Statement	Description
<p>example using metafile:object</p>	<p>example</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <div style="border: 1px solid black; width: 80px; height: 20px; margin: 0 auto; text-align: center; line-height: 20px;">OK</div> <p>In a text button action program enter:</p> <pre>SEND "Display(..schemat/stat1)" TO "plant:StatusDisplay";</pre> </div> <p>The program associated with the button called OK in the metafile causes a file called stat1 to be displayed in the display area called StatusDisplay. This display area is in the window called plant, which must already be visible.</p>
<p>example using "Bell"</p>	<p><i>example</i></p> <p>This makes the computer beep.</p> <pre>SEND "Bell" TO "MetaFile";</pre> <p><i>example</i></p> <pre>SEND "Bell(50, 500, 1000)" TO "MetaFile";</pre> <p>The general format is:</p> <pre>SEND "Bell" TO "MetaFile";</pre> <pre>SEND "Bell(volume)" TO "MetaFile";</pre> <pre>SEND "Bell(volume, duration)" TO "MetaFile";</pre> <pre>SEND "Bell(volume, duration, pitch)" TO "MetaFile";</pre> <p>Where volume is in %, 100% is the loudest, duration is in milliseconds and pitch is in Hertz.</p>

Table 23: The SEND <Message> TO Command

Statement	Description
<p>example using BeginWait and EndWait</p>	<p>example</p> <pre>SEND "BeginWait" TO "MetaFile";</pre> <p>This program makes the cursor change into a clock to signify that the processor may take time to complete the operation.</p> <p>example</p> <pre>SEND "EndWait" TO "MetaFile";</pre> <p>This program turns the cursor back to the original cursor from the clock indicating that the delay is over.</p> <p>Note: If two BeginWait messages are sent, then it will take two EndWait messages to clear the clock cursor.</p> <p>Also note that "Metafile" is not a file name, but the string "MetaFile".</p>
<p>example using SetUpdateRate (rate)</p>	<p>example</p> <pre>SEND "SetUpdateRate(200)" TO "MetaFile";</pre> <p>This program changes the update rate of the dynamic values in the metafile. I.e., it overrides the default value set in the start up command. The rate is measured in one hundredths of a second. I.e. the example above sets the update rate to 2 seconds.</p> <p>Note: You can also set the update rate for each metafile using:</p> <ol style="list-style-type: none"> i. The forms editor and entering the update rate in the header form. (Refer to the graphics chapter for more details.) ii. The start up command with the <code>-updateRate <seconds></code> command line parameter. iii. The #UPDATE modifier grouped with itself in the CAD diagram. E.g. #UPDATE 2.5 grouped with itself will set the update rate of the metafile to 2.5 seconds. <p>Note: If you enter the update rate in the file header, it overrides the update rate in the command line for that meta file.</p>

Table 24: The SEND <Message> TO Command

Statement	Description	
<p>Example using Focus</p>	<p><i>example</i></p> <pre>SEND "Focus" TO "pageInput";</pre> <p>Note: This is an extremely useful command for "busy" input screens.</p> <p>Note: The user can always move the focus from object to object using the TAB (forward) and shift TAB (backward) keys.</p> <p>For more information on Focus, see the section "Focus Issues" in the Graphics chapter.</p>	<p>This program asks the Display manager to place the Focus on an object called pageInput. The user's keystrokes will be sent to the target object without having to manually position the cursor.</p>
<p>Example using DisableClose</p>	<p><i>example</i></p> <pre>SEND "DisableClose" TO "MetaFile";</pre> <p><i>example</i></p> <pre>SEND "EnableClose" TO "MetaFile";</pre>	<p>This program prevents the user from closing the window using the Window Manager pulldown menu located in the border of the window.</p> <p>EnableClose re-allows the pulldown close option.</p> <p>Note: This may not work with all window managers.</p>
<p>Example using Update</p>	<p><i>example</i></p> <pre>SEND "Update" TO "MetaFile";</pre>	<p>This program forces a dynamic update of all the variables on the screen regardless of the current window update rate.</p>

SET ...FROM ATTRIBUTE

Table 25: The SET ...FROM ATTRIBUTE Command

Statement	Description
SET ...FROM ATTRIBUTE	<p>The <code>SET...FROM ATTRIBUTE</code> command is used to access an Entity/attribute where the entity and/or attribute is specified in another variable.</p>
format	<p><i>format</i></p> <pre>SET <stringVar> FROM ATTRIBUTE "{entityVar}.{attrVar}";</pre> <p>Where <code>entVar</code> is a string variable containing a valid entity name and <code>attrVar</code> is a string variable containing a valid attribute name for the entity type.</p> <p><i>example</i></p> <pre>SET selectTag FROM ATTRIBUTE "LC110.PV";</pre> <p>This is used, typically, in Action programs associated with double clicking on a Browse Box.</p>

SET ...FORMAT

Table 26: The SET FORMAT Command

Statement	Description	
SET ...FORMAT	The <code>SET...FORMAT</code> command is used to set up the time, date and duration formats for subsequent print statement. Refer to the Time/Date section. There are three main commands.	
format	<p><i>format</i></p> <pre>SET DATE FORMAT TO <exprn>;</pre> <pre>SET DURATION FORMAT TO <exprn>;</pre> <pre>SET TIME FORMAT TO <exprn>;</pre>	<p>Sets the date format for subsequent print statements.</p> <p>Sets the time duration format for subsequent print statements.</p> <p>Sets the time stamp format for subsequent print statements.</p>
argument	<p><exprn></p> <p>The expression must evaluate to one of the string formats that are given in the table "Time, Date & Duration Formats" in the section Date and Time Handling.</p>	
example	<p><i>example</i></p> <pre>SET DATE FORMAT TO</pre> <pre>"Ddd MMMM, CCYY";</pre> <pre>PRINTLN "Today is" +</pre> <pre>DATESTR(NOW());</pre>	<p>This will have a print out that looks like this:</p> <p>Today is 2nd August, 1994.</p>
example	<p><i>example</i></p> <pre>SET TIME FORMAT TO</pre> <pre>"hh:mm AM";</pre> <pre>PRINTLN "It is now" +</pre> <pre>TIMESTR(NOW());</pre>	<p>This will result in the following print out.</p> <p>It is now 05:33 PM</p>

Table 27: The SET FORMAT Command

Statement	Description
example	<p>example</p> <pre>SET DURATION FORMAT TO "%H %m %s %i"; PRINTLN "Time to changeover is \n" + DURATIONSTR(TIME(8,0,0))-NOW());</pre> <p>This example will give a print out like:</p> <p>Time to change over is:</p> <p>2 hours 3 minutes 5 seconds 123 milliseconds.</p>
embedded formats	<p>You can also embed the format within some commands without using the <code>SET FORMAT</code> command E.g.</p> <pre>PRINTLN TIMESTR(NOW(), "MMM DD, YY");</pre> <p>will result in</p> <p>Oct 15, 94</p>
note	<p>For additional date, time and duration commands refer to the chapter on Date and Time handling.</p>

SET POINTER

Table 28: The SET POINTER Command

Statement	Description	
SET POINTER	The <code>SET POINTER</code> command creates an alias which is used to indirectly reference an entity. All subsequent uses of the name will result in indirect access to the specified entity.	
format	<p><i>format</i></p> <pre>SET POINTER <pointerName> TO <entityName>;</pre> <p>Creates an alias that can be substituted in a generic program at a later stage.</p>	
arguments	<p><code><pointerName></code></p> <p>The alias to be used in a later program.</p> <p><code><entityName></code></p> <p>This is a string expression that resolves to an entity name.</p>	
example	<p><i>example</i></p> <pre>LET X = 110; SET POINTER Y TO "FIC" + X; prt_rpt.ms PRINTLN Y.PV;</pre>	<p>First create the aliases that are to be used later in a generic meta script.</p> <p>Now use the aliases in the meta script sub-routine prt_rep.ms</p>
example (using a string expression)	<pre>SET POINTER LVLCTRL TO "LC" + controllerNumber;</pre> <p>Meta script commands can now be used referring to LVLCTRL as a variable name.</p>	

SET MAXIMUM PROGRAM TIME

Table 29: The SET MAXIMUM PROGRAM TIME Command

Statement	Description	
SET MAXIMUM PROGRAM TIME	This command sets the maximum time the meta script program should take to complete. If the program takes longer than the time you specify, then it is aborted. Because of efficiency reasons, the check is only carried out on commands that could result in the program getting into an infinite loop. I.e. <code>SLEEP</code> , <code>FOR</code> , <code>DO...WHILE</code> , and <code>LOOP THROUGH</code> statements.	
format	<p><i>format</i></p> <pre>SET MAXIMUM PROGRAM TIME TO <exprn>;</pre> <p>SET MAXIMUM PROGRAM TIME TO INFINITE;</p>	<p>The timeout period is set to the number of seconds specified by <code><exprn></code>.</p> <p>The timeout interrupting mechanism is disabled.</p>
Default	This will depend on the environment. For the operations program, it is 10 seconds.	
example	<p><i>example</i></p> <pre>SET MAXIMUM PROGRAM TIME TO 20;</pre>	If the program time exceeds 20 seconds, the program will be aborted.

10.10 Control Statements

These statements affect the flow of the program. I.e. they determine whether a particular block of code will be executed based on the prevailing conditions. The diagram below is a summary of the main control statements available in the meta script language. The tables on the next page discuss these statements in more detail.

PROGRAM CONTROL

IF ELSE	<pre>IF(<exprn><action>; IF (<exprn>) { <actions>; }</pre>	<pre>IF(<exprn><action>; ELSE <action>;</pre>	<pre>IF(<exprn>) { <actions>; } ELSE<action>;</pre>	<pre>IF(<exprn>) { <actions>; } ELSE { <actions>; }</pre>		
REPEAT UNTIL	<pre>REPEAT { <actions>; } UNTIL(<exprn>) ;</pre>	<table border="1"> <tr> <td style="text-align: center; vertical-align: middle;">WHILE</td> <td> <pre>WHILE(<exprn>); { <actions>; }</pre> </td> </tr> </table>			WHILE	<pre>WHILE(<exprn>); { <actions>; }</pre>
WHILE	<pre>WHILE(<exprn>); { <actions>; }</pre>					
BREAK	<pre>WHILE (<exprn>) { <actions>; IF (<exprn>)BREAK; <actions>; } ←</pre>	CONTINUE	<pre>WHILE (<exprn>) { <actions>; IF (<exprn>)CONTINUE ; <actions>; }</pre>	RETURN	<pre>RETURN; RETURN <exprn>;</pre>	
RUN	<pre>RUN <exprn>; LET <tagName>=RUN(<exprn>); RUN <exprn> BEGIN SCRIPT Bourne shell script END SCRIPT;</pre>	SLEEP	<pre>SLEEP <exprn>;</pre>	TERMINATE	<pre>TERMINATE; Terminates the program that is running the meta script.</pre>	

IF ELSE

Table 30: The IF ELSE Command

Statement	Description
IF ELSE	The IF ELSE control statement provides the ability to choose a set of program statements to execute depending upon the result of an expression evaluation. If the expression evaluates to non-zero (i.e. TRUE) then the set of program statements associated with the IF are executed, otherwise the set of program statements associated with the ELSE are executed. (The ELSE is optional.)
format	The expression to be evaluated must be in round brackets. It must evaluate to T (i.e. non-zero) or F (i.e. zero). The IF part of the statement is executed when the expression is true and the ELSE part (if present) is executed when the expression is false. The curly brackets or braces { } must enclose groups of statements in a block to be executed.
Form 1	There are various forms of the IF ELSE statement as shown below: <i>form</i> <code>IF(<exprn><action>;</code> <i>examples</i> <code>IF(L109.PV >80)LET FW109.SV=0;</code>
Form 2	<code>IF(<exprn>) { <actions>; }</code> <code>IF(L109.PV >80) { LET FW109.LS='MAN'; LET FW109.MV =0; }</code>
Form 3	<code>IF(<exprn><action>; ELSE <action>;</code> <code>IF(FW109.MV <20)LET FW109.MV=100; ELSE LET FW109.MV=0;</code>
Form 4	<code>IF(<exprn>) { <actions>; } ELSE { <actions>; }</code> <code>IF(AccessLevel>80) { DELETE FROM OPLOG WHERE CURRENT; FETCH LAST OF OPLOG; } ELSE { LET errMsg='No authority'; DISPLAY WINDOW `errpop'; }</code>

REPEAT UNTIL

Table 32: The REPEAT UNTIL Command

Statement	Description		
<p>REPEAT ...UNTIL</p>	<p>This is a looping construct. I.e. it gives you the opportunity to loop through a set of commands until a specified condition is met.</p> <p>The expression must be in round brackets. I.e. () and must evaluate to true (T, non-zero) or false (F zero). You may use several lines for clarity. You must use curly braces { } to block multiple statements.</p>		
<p>Form 1</p>	<p>There are two forms of this statement.</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;"> <p>Form</p> <pre>REPEAT <action>; UNTIL(<expr>;</pre> </td> <td style="width: 50%; vertical-align: top;"> <p>example</p> <pre>REPEAT LET FW109.SV=FW109.SV+1; UNTIL (FW109.PV>=120);</pre> </td> </tr> </table>	<p>Form</p> <pre>REPEAT <action>; UNTIL(<expr>;</pre>	<p>example</p> <pre>REPEAT LET FW109.SV=FW109.SV+1; UNTIL (FW109.PV>=120);</pre>
<p>Form</p> <pre>REPEAT <action>; UNTIL(<expr>;</pre>	<p>example</p> <pre>REPEAT LET FW109.SV=FW109.SV+1; UNTIL (FW109.PV>=120);</pre>		
<p>Form 2</p>	<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;"> <p>Form</p> <pre>REPEAT { <actions>; } UNTIL (<exprn>;</pre> </td> <td style="width: 50%; vertical-align: top;"> <p>example</p> <pre>// Exercise valve LET tm=0; REPEAT { IF(FC109.MV<20)LET FC109.MV=100; ELSE LET FC109.MV=0; SLEEP 120; LET tm=tm+2; } UNTIL(tm>=10);</pre> </td> </tr> </table>	<p>Form</p> <pre>REPEAT { <actions>; } UNTIL (<exprn>;</pre>	<p>example</p> <pre>// Exercise valve LET tm=0; REPEAT { IF(FC109.MV<20)LET FC109.MV=100; ELSE LET FC109.MV=0; SLEEP 120; LET tm=tm+2; } UNTIL(tm>=10);</pre>
<p>Form</p> <pre>REPEAT { <actions>; } UNTIL (<exprn>;</pre>	<p>example</p> <pre>// Exercise valve LET tm=0; REPEAT { IF(FC109.MV<20)LET FC109.MV=100; ELSE LET FC109.MV=0; SLEEP 120; LET tm=tm+2; } UNTIL(tm>=10);</pre>		

WHILE

Table 33: The WHILE Command

Statement	Description				
WHILE	This is a looping construct similar to the <code>REPEAT . . . UNTIL</code> with the exception that the condition is evaluated at the start of the loop.				
format	The expression must be in round brackets. I.e. (<code><exprn></code>) and must evaluate to true (T non-zero) or false (F zero). You may use several lines for clarity. You must use curly braces <code>{ }</code> to block multiple statements.				
form	<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; border: none;">Format</td> <td style="width: 50%; border: none;">example</td> </tr> <tr> <td style="border: none;"> <pre>WHILE (<exprn>) { <multiple commands>; }</pre> </td> <td style="border: none;"> <pre>WHILE (FM109.PV<120) { LET FM109.SV=FM109.SV)+1; SLEEP 1; }</pre> </td> </tr> </table>	Format	example	<pre>WHILE (<exprn>) { <multiple commands>; }</pre>	<pre>WHILE (FM109.PV<120) { LET FM109.SV=FM109.SV)+1; SLEEP 1; }</pre>
Format	example				
<pre>WHILE (<exprn>) { <multiple commands>; }</pre>	<pre>WHILE (FM109.PV<120) { LET FM109.SV=FM109.SV)+1; SLEEP 1; }</pre>				

BREAK

Table 34: The BREAK Command

Statement	Description				
BREAK	The <code>BREAK</code> statement is used to abandon the current looping construct and go to the next sequential statement. The <code>BREAK</code> command can be used with both the <code>REPEAT . . . UNTIL</code> and also the <code>WHILE</code> construct.				
format	<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; border: none;"><i>Format</i></td> <td style="width: 50%; border: none;"><i>example</i></td> </tr> <tr> <td style="border: none;"> <pre>WHILE (exprn>) { <program statements>; IF(<exprn>) BREAK; <program statements>; }</pre> </td> <td style="border: none;"> <pre>WHILE (FC109.LS= 'CAS') { LET FC109.SV=FC109.MV*2; IF(L109.PV>90)BREAK; SLEEP 1; } LET FC109.LS= 'MAN' ; LET FC109.MV=0;</pre> </td> </tr> </table>	<i>Format</i>	<i>example</i>	<pre>WHILE (exprn>) { <program statements>; IF(<exprn>) BREAK; <program statements>; }</pre>	<pre>WHILE (FC109.LS= 'CAS') { LET FC109.SV=FC109.MV*2; IF(L109.PV>90)BREAK; SLEEP 1; } LET FC109.LS= 'MAN' ; LET FC109.MV=0;</pre>
<i>Format</i>	<i>example</i>				
<pre>WHILE (exprn>) { <program statements>; IF(<exprn>) BREAK; <program statements>; }</pre>	<pre>WHILE (FC109.LS= 'CAS') { LET FC109.SV=FC109.MV*2; IF(L109.PV>90)BREAK; SLEEP 1; } LET FC109.LS= 'MAN' ; LET FC109.MV=0;</pre>				

CONTINUE

Table 35: The CONTINUE Command

Statement	Description
<p>CONTINUE</p> <p>format</p>	<p>The CONTINUE statement is used to skip the remaining statements in the block but to still continue the looping "from the top". Like the BREAK command, the CONTINUE command can be used in both the REPEAT and WHILE constructs.</p> <p><i>Format</i></p> <pre> WHILE (<exprn>) { <actions>; IF(<exprn>) CONTINUE; <actions>; } </pre> <p><i>example</i></p> <pre> WHILE (BATVU.RECNO < 400) { FETCH NEXT OF BATVU; PRINTLN BATVU.CUSTOMER; IF(BATVU.DEV < limit) CONTINUE; PRINTLN "Bad Batch"; } </pre>

RETURN

Table 36: The RETURN Command

Statement	Description
<p>RETURN</p> <p>format</p>	<p>This statement is used to stop the execution of the program so as to return control to the calling program possibly with an optional return value. The return statement is often used in returning results such as colors in graphic meta scripts.</p> <p><i>Format</i></p> <pre> RETURN; Or RETURN <exprn>; </pre> <p><i>example</i></p> <pre> RETURN; RETURN HalfCyan; </pre>

Table 38: The RUN Command

Statement	Description
Format 3	<pre>RUN <exprn> BEGIN SCRIPT <script contents> END SCRIPT;</pre> <p>The third format enables a script to be embedded in the meta script. This avoids any administration overheads in maintaining separate script files. Note that the script must be a Bourne shell script. Note to that <code><exprn></code> is an optional expression of string format that may include arguments. This format is not supported under <i>MacroView</i> for Windows NT.</p>
Example 3	<pre>RUN " " BEGIN SCRIPT sendmsg 4 13 \'Report complete\' END SCRIPT; RUN " "</pre>
Note	
Example	
No arguments	<pre>RUN varName1 + "+" varName2</pre>
Arguments	(Arguments that can be referenced as \$1 and \$2 in the script that follows.)

SLEEP

Table 39: The SLEEP Command

Statement	Description
SLEEP	The sleep command suspends execution of the meta script for a number of seconds before continuing.
Format	<pre>SLEEP <exprn>;</pre>
example	<pre>//Wait 3 seconds SLEEP 3;</pre>

TERMINATE

Table 40: The TERMINATE Command

Statement	Description
TERMINATE	The terminate command terminates: <ul style="list-style-type: none"> • The meta script itself and • The program that is running the meta script.
Restrictions	The program that is running the meta script must have the ability to recognize the terminate signal for this to be effective.
Format	<code>TERMINATE ;</code>
example	<code>TERMINATE ;</code>

LOOP THROUGH

Table 41: The LOOP THROUGH Command

Statement	Description
LOOP THROUGH	The <code>LOOP THROUGH</code> command is a flow control command that is specific to database and is therefore discussed in more detail in the database section.

10.11 Database Functions

These statements are SQL-like statements that enable you to work with databases. The diagram below is a summary of the database statements available in the meta script language. The tables on the next pages discuss these commands in more detail.

INSERT	<pre>INSERT INTO <entity>; INSERT INTO <entity> FIELDS attr1, attr2,.. VALUES expr1, expr2,..;</pre>	UPDATE	<pre>UPDATE <entity> SET attr1=expr1, attr2=expr2,..;</pre>	DELETE	<pre>DELETE FROM <entity> WHERE CURRENT; DELETE ALL FROM <entity>;</pre>
FETCH	<pre>FETCH FIRST OF <entity>; FETCH LAST OF <entity>; FETCH PREVIOUS OF <entity>; FETCH PREVIOUS OF <entity> WHERE "<searchExprn>"; FETCH NEXT OF <entity>; FETCH NEXT OF <entity> WHERE "<searchExprn>";</pre>	LOOP THROUGH	<pre>LOOP THROUGH <entity> { <meta script Statements> }</pre> <p>Note <entity> can be an entity or a view.</p>	CREATE TABLE VARIABLE	<pre>CREATE TABLE VARIABLE <tableName> FIELDS <fieldName1> <fldFormat1>, <fieldName2> <fldFormat2>, <fieldNameN> <fldFormatN>;</pre>
CREATE VIEW	<pre>CREATE VIEW <viewName> AS SELECT <selectSpec> FROM <entity> WHERE "<filterExprn>" ORDER by <orderSpec>; Or alternatively FROM FILE "<fileName>"</pre>	CREATE HISTORICAL VIEW	<pre>CREATE HISTORICAL VIEW <tableName> AS SELECT <fieldName1> = <entity1.attr1>, <fieldName2> = <entity1.attr1>, . . . <fieldNameN> = <attrN(entityN)> WITH <duration> <period> <hForm> WHERE TIME >= <startTstamp> AND TIME <= <endTstamp>;</pre>	ERROR PROCESSING VARIABLE	<pre>BERR(<entity>) returns the error number. (0 if success) DBERRSTR(<entity>) returns an error message (empty if success)</pre>

INSERT

Table 42: The INSERT Command

Statement	Description
<p>INSERT</p> <p>Format</p> <p>Arguments</p> <p>Note on Dbase INSERT</p>	<p>The INSERT command enables you to append a record to a database and optionally set the values of the fields.</p> <p>There are two forms of this command:</p> <p>You can add a blank record (form 1).</p> <p>You can add a record and set the values of the specified fields. (form 2).</p> <p><i>entity</i></p> <p>An entity configured as a database entity.</p> <p><i>attr?</i></p> <p>The field names in the database. (I.e. <i>attr1</i>, <i>attr2</i> etc.)</p> <p><i>expr?</i></p> <p>Valid expressions that evaluate to the required values to go into the specified fields.</p> <p>The INSERT command is styled around the SQL version (which dBase users regard as an <i>append</i>) and not the dBase version. There is no equivalent to the dBase INSERT, which does actually insert a record rather than appending it.</p>
<p>Form 1</p>	<p><i>form</i> <i>example</i></p> <pre data-bbox="456 1227 1251 1249">INSERT INTO <entity>; INSERT INTO OPLOG;</pre> <p>This example simply appends a blank record at the end of the database entity OPLOG.</p>

Table 43: The INSERT Command

Statement	Description			
Form 2	<p><i>form</i></p> <pre>INSERT INTO <entity> FIELDS attr1, attr2,...attrn VALUES expr1, expr2,...exprn;</pre>	<p><i>example</i></p> <pre>INSERT INTO LABLOG FIELDS DT, TM, LEVEL, FLOW VALUES DATE(), TIME(), L109.PV, FM109.PV;</pre> <p>This example appends a record at the end of the database entity LABLOG and sets the values of the fields DT and TM to the current date and time. It also sets the fields LEVEL and FLOW to the values of the PV of L109 and FM109 respectively.</p>		
Formatting options	<p>You may freely organize the command across multiple lines so as to improve the readability of the command i.e. the following two statements are equally valid.</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;"> <p><i>statement 1</i></p> <pre>INSERT INTO OPLOG FIELDS TM, COMMENT VALUES NOW(), "Problems in Area 6";</pre> </td> <td style="width: 50%; vertical-align: top;"> <p><i>statement 2</i></p> <pre>INSERT INTO OPLOG FIELDS TM, COMMENT VALUES NOW(), "Problems in Area 6";</pre> </td> </tr> </table>		<p><i>statement 1</i></p> <pre>INSERT INTO OPLOG FIELDS TM, COMMENT VALUES NOW(), "Problems in Area 6";</pre>	<p><i>statement 2</i></p> <pre>INSERT INTO OPLOG FIELDS TM, COMMENT VALUES NOW(), "Problems in Area 6";</pre>
<p><i>statement 1</i></p> <pre>INSERT INTO OPLOG FIELDS TM, COMMENT VALUES NOW(), "Problems in Area 6";</pre>	<p><i>statement 2</i></p> <pre>INSERT INTO OPLOG FIELDS TM, COMMENT VALUES NOW(), "Problems in Area 6";</pre>			

UPDATE

Table 44: The UPDATE Command

Statement	Description						
<p>UPDATE</p> <p>Format</p> <p>Arguments</p>	<p>This command enables you to write into multiple fields of a database entity (or a view of the database entity.)</p> <p>You can write into the fields of the current record (form 1).</p> <p>entity</p> <p>The data base entity.</p> <p>attr?</p> <p>The attributes (i.e. field names) that are to be modified. (I.e. attr1, attr2 etc.)</p> <p>expr?</p> <p>Expressions that evaluate to the values to be placed in the fields.</p>						
<p>Form 1</p>	<table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; width: 50%;"><i>form</i></td> <td style="vertical-align: top; width: 50%;"><i>example</i></td> </tr> <tr> <td style="vertical-align: top;"> <pre>UPDATE <entity> SET attr1 = expr1, attr2 = expr2, attrN = exprnN;</pre> </td> <td style="vertical-align: top;"> <pre>LET ratio = FC109.SUM/FM109.SUM; UPDATE BATCH9 SET END_TM = NOW(), WT_TOT = FW109.SUM, CS_TOT = FC109.SUM, RAT_MS = ratio;</pre> </td> </tr> <tr> <td colspan="2" style="vertical-align: top;"> <p>This example first calculates the ratio of caustic to total addition to a mixing tank. It then writes the mixing end time, total water, total caustic and the measured ratio to the selected record of a database entity called BATCH9.</p> </td> </tr> </table>	<i>form</i>	<i>example</i>	<pre>UPDATE <entity> SET attr1 = expr1, attr2 = expr2, attrN = exprnN;</pre>	<pre>LET ratio = FC109.SUM/FM109.SUM; UPDATE BATCH9 SET END_TM = NOW(), WT_TOT = FW109.SUM, CS_TOT = FC109.SUM, RAT_MS = ratio;</pre>	<p>This example first calculates the ratio of caustic to total addition to a mixing tank. It then writes the mixing end time, total water, total caustic and the measured ratio to the selected record of a database entity called BATCH9.</p>	
<i>form</i>	<i>example</i>						
<pre>UPDATE <entity> SET attr1 = expr1, attr2 = expr2, attrN = exprnN;</pre>	<pre>LET ratio = FC109.SUM/FM109.SUM; UPDATE BATCH9 SET END_TM = NOW(), WT_TOT = FW109.SUM, CS_TOT = FC109.SUM, RAT_MS = ratio;</pre>						
<p>This example first calculates the ratio of caustic to total addition to a mixing tank. It then writes the mixing end time, total water, total caustic and the measured ratio to the selected record of a database entity called BATCH9.</p>							

DELETE

Table 45: The DELETE Command

Statement	Description						
<p>DELETE</p> <p>Format</p> <p>Arguments</p>	<p>This command enables you to delete selected records from a database entity or a view of a database entity.</p> <ul style="list-style-type: none"> • There are two forms of this command. • You can delete the current record (form 1). • You can delete all records (form 2). <p><i>entity</i>:</p> <p>The database entity.</p>						
<p>Form 1</p>	<table border="0"> <tr> <td data-bbox="450 772 922 817"><i>form</i></td> <td data-bbox="954 772 1439 817"><i>example</i></td> </tr> <tr> <td data-bbox="450 875 922 958"> <pre>DELETE FROM <entity> WHERE CURRENT;</pre> </td> <td data-bbox="954 875 1439 958"> <pre>DELETE FROM OPLOG WHERE CURRENT;</pre> </td> </tr> </table>	<i>form</i>	<i>example</i>	<pre>DELETE FROM <entity> WHERE CURRENT;</pre>	<pre>DELETE FROM OPLOG WHERE CURRENT;</pre>		
<i>form</i>	<i>example</i>						
<pre>DELETE FROM <entity> WHERE CURRENT;</pre>	<pre>DELETE FROM OPLOG WHERE CURRENT;</pre>						
<p>Form 2</p>	<table border="0"> <tr> <td data-bbox="450 981 922 1025"><i>form</i></td> <td data-bbox="954 981 1439 1025"><i>example</i></td> </tr> <tr> <td data-bbox="450 1084 922 1128"> <pre>DELETE ALL FROM <entity>;</pre> </td> <td data-bbox="954 1084 1439 1128"> <pre>DELETE ALL FROM BATCH9;</pre> </td> </tr> <tr> <td colspan="2" data-bbox="450 1187 1439 1261"> <p>This example deletes all the records (but not the database itself) from the database entity BATCH9.</p> </td> </tr> </table>	<i>form</i>	<i>example</i>	<pre>DELETE ALL FROM <entity>;</pre>	<pre>DELETE ALL FROM BATCH9;</pre>	<p>This example deletes all the records (but not the database itself) from the database entity BATCH9.</p>	
<i>form</i>	<i>example</i>						
<pre>DELETE ALL FROM <entity>;</pre>	<pre>DELETE ALL FROM BATCH9;</pre>						
<p>This example deletes all the records (but not the database itself) from the database entity BATCH9.</p>							

FETCH

Table 46: The FETCH Command

Statement	Description												
<p>FETCH</p> <p>Format</p> <p>Arguments</p>	<p>This command fetches the selected record from the database entity. In reality, it moves to the selected record and makes it the "current" record. (I.e. It sets the RECNO attribute of the database to the selected record number.)</p> <p>As shown below, there are various forms of this command to enable you to easily find the record of interest.</p> <p><i>entity</i></p> <p>The database entity.</p> <p><i>searchExprne</i></p> <p>The expression that is used to search through the database to find the record to be selected. This must evaluate to a string that is passed to the RDBMS, which is used as the expression to get the record of interest.</p>												
<p>Form 1</p>	<table border="0"> <tr> <td style="vertical-align: top;"><i>form</i></td> <td style="vertical-align: top;"><i>example</i></td> </tr> <tr> <td><code>FETCH FIRST OF <entity>;</code></td> <td><code>FETCH FIRST OF OPLOG;</code></td> </tr> <tr> <td><code>FETCH LAST OF <entity>;</code></td> <td><code>FETCH LAST OF OPLOG;</code></td> </tr> </table> <p>The examples above move to the first record and last record of the database entity OPLOG respectively.</p>	<i>form</i>	<i>example</i>	<code>FETCH FIRST OF <entity>;</code>	<code>FETCH FIRST OF OPLOG;</code>	<code>FETCH LAST OF <entity>;</code>	<code>FETCH LAST OF OPLOG;</code>						
<i>form</i>	<i>example</i>												
<code>FETCH FIRST OF <entity>;</code>	<code>FETCH FIRST OF OPLOG;</code>												
<code>FETCH LAST OF <entity>;</code>	<code>FETCH LAST OF OPLOG;</code>												
<p>Next form</p>	<table border="0"> <tr> <td style="vertical-align: top;"><i>form</i></td> <td style="vertical-align: top;"><i>example</i></td> </tr> <tr> <td><code>FETCH NEXT OF <entity>;</code></td> <td><code>FETCH NEXT OF OPLOG;</code></td> </tr> </table> <p>This moves you to the next record in the database.</p> <table border="0"> <tr> <td style="vertical-align: top;"><i>form</i></td> <td style="vertical-align: top;"><i>example</i></td> </tr> <tr> <td><code>FETCH NEXT OF <entity></code></td> <td><code>FETCH NEXT OF OPLOG;</code></td> </tr> <tr> <td><code>WHERE</code></td> <td><code>WHERE</code></td> </tr> <tr> <td><code>"<searchExprn>" ;</code></td> <td><code>"AREA = 'POWER ' " ;</code></td> </tr> </table> <p>This moves to the next record in the database where the field AREA has been set to POWER.</p>	<i>form</i>	<i>example</i>	<code>FETCH NEXT OF <entity>;</code>	<code>FETCH NEXT OF OPLOG;</code>	<i>form</i>	<i>example</i>	<code>FETCH NEXT OF <entity></code>	<code>FETCH NEXT OF OPLOG;</code>	<code>WHERE</code>	<code>WHERE</code>	<code>"<searchExprn>" ;</code>	<code>"AREA = 'POWER ' " ;</code>
<i>form</i>	<i>example</i>												
<code>FETCH NEXT OF <entity>;</code>	<code>FETCH NEXT OF OPLOG;</code>												
<i>form</i>	<i>example</i>												
<code>FETCH NEXT OF <entity></code>	<code>FETCH NEXT OF OPLOG;</code>												
<code>WHERE</code>	<code>WHERE</code>												
<code>"<searchExprn>" ;</code>	<code>"AREA = 'POWER ' " ;</code>												

Table 47: The FETCH Command

Statement	Description	
FETCH PREVIOUS	<i>form</i>	<i>example</i>
	<pre> FETCH PREVIOUS OF <entity>; </pre> <p>This moves you to the previous record in the database.</p>	
	<i>form</i>	<i>example</i>
	<pre> FETCH PREVIOUS OF <entity> WHERE "<searchExprn>"; </pre> <pre> FETCH PREVIOUS OF BATCH9; WHERE "CUSTOMER = `AMALG `"; </pre> <p>This moves back in the database until it reads a record where the field CUSTOMER has been set to AMALG.</p>	
FETCH FIRST OF	<i>form</i>	<i>example</i>
	<pre> FETCH FIRST OF <entity>; </pre> <p>This moves you to the first record in the database.</p>	
	<i>form</i>	<i>example</i>
	<pre> FETCH FIRST OF <entity> WHERE "<searchExprn>"; </pre> <pre> FETCH FIRST OF BATCH9; WHERE "CUSTOMER = `AMALG `"; </pre> <p>This moves to the first record in the database where the field CUSTOMER has been set to AMALG.</p>	
FETCH LAST OF	<i>form</i>	<i>example</i>
	<pre> FETCH LAST OF <entity>; </pre> <p>This moves you to the last record in the database.</p>	
	<i>form</i>	<i>example</i>
	<pre> FETCH LAST OF <entity> WHERE "<searchExprn>"; </pre> <pre> FETCH LAST OF BATCH9; WHERE "CUSTOMER = `AMALG `"; </pre> <p>This moves to the last record in the database where the field CUSTOMER has been set to AMALG.</p>	

Table 48: The FETCH Command

Statement	Description
<p>FETCH ABSOLUTE</p>	<p><i>form</i> <i>example</i></p> <pre> FETCH ABSOLUTE <record number> OF <DBASE4 entity>; </pre> <p> <pre> FETCH ABSOLUTE STATE.PV OF STATEDB; </pre> </p> <p>This positions the current record point for a DBASE data source to the absolute record number.</p> <p>A typical application of the FETCH ABSOLUTE command is described in the following scenario:</p> <p>A PLC contains a logical state machine program for controlling the startup and shutdown process for a complex piece of machinery. An entity (STATE) is configured in MacroView which reflects the current state number of the startup/shutdown sequence within the PLC (STATE.PV). In MacroView, a DBASE4 file is used to store the text descriptions of the various possible states and referenced in MacroView as a DBASE4 entity (STATEDB). This database has a text field (TEXT) which describes the current state. The record number of the database corresponds to the state number within the PLC.</p> <p>On a MacroView schematic, the following Textual Update MAT Text modifier could be used to display a textual description of the current state of the PLC startup/shutdown sequence:</p> <pre> FETCH ABSOLUTE STATE.PV OF STATEDB; RETURN "PLC State: " + STATEDB.TEXT; </pre> <p><u>Important Note:</u></p> <p>The above example uses a DBASE4 entity to access the required database. In this case, when the value of STATE.PV = 1 the record pointer is set to 1 in the data base, which is the first record in the database. When STATE.PV = 2 the pointer is set to record 2 etc.</p> <p>If you are referring to a VIEW, rather than a DBASE4 entity, the first record in the VIEW is actually record 0 (zero). So when the value of STATE.PV = 1, the pointer will be set to record number 2 in the database.</p>
<p>FETCH RELATIVE</p>	<p><i>form</i> <i>example</i></p> <pre> FETCH RELATIVE <offset> OF <DBASE4 entity>; </pre> <p> <pre> FETCH RELATIVE STATE.PV OF STATEDB; </pre> </p>
	<p>This will move the current record number relative to its current position and will accept a positive or a negative number.</p>

LOOP THROUGH

Table 49: The LOOP THROUGH Command

Statement	Description				
<p>LOOP THROUGH</p> <p>Arguments</p>	<p>This command enables you to work through a database or a view of a database and process the records. Generally, you would create a view of the database (and filter it if necessary) before running this command so that only the selected records are processed.</p> <p>entity</p> <p>The database entity or view of the database <code><entity></code>.</p> <p>meta script statements</p> <p>The statements in between the curly brackets ({ }) that are executed for all the records in the view.</p>				
<p>Form</p>	<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;"> <p><i>form</i></p> <pre> LOOP THROUGH <entity> { <meta script statements>; } </pre> </td> <td style="width: 50%; vertical-align: top;"> <p><i>example</i></p> <pre> LOOP THROUGH BATCH9 { IF(ABS(BATCH9.MSRT -BATCH9.RQRT) < 3.5) LET BATCH9.OFFSPEC = T; ELSE LET BATCH9.OFFSPEC = F; } </pre> </td> </tr> <tr> <td colspan="2" style="padding-top: 10px;"> <p>This example goes through a database and sets the field OFFSPEC true if the difference between the required ratio RQRT and the measured ratio MSRT is greater than 3.5</p> </td> </tr> </table>	<p><i>form</i></p> <pre> LOOP THROUGH <entity> { <meta script statements>; } </pre>	<p><i>example</i></p> <pre> LOOP THROUGH BATCH9 { IF(ABS(BATCH9.MSRT -BATCH9.RQRT) < 3.5) LET BATCH9.OFFSPEC = T; ELSE LET BATCH9.OFFSPEC = F; } </pre>	<p>This example goes through a database and sets the field OFFSPEC true if the difference between the required ratio RQRT and the measured ratio MSRT is greater than 3.5</p>	
<p><i>form</i></p> <pre> LOOP THROUGH <entity> { <meta script statements>; } </pre>	<p><i>example</i></p> <pre> LOOP THROUGH BATCH9 { IF(ABS(BATCH9.MSRT -BATCH9.RQRT) < 3.5) LET BATCH9.OFFSPEC = T; ELSE LET BATCH9.OFFSPEC = F; } </pre>				
<p>This example goes through a database and sets the field OFFSPEC true if the difference between the required ratio RQRT and the measured ratio MSRT is greater than 3.5</p>					

Table 50: The LOOP THROUGH Command

Statement	Description
Example 2	<pre> <i>example2</i> CREATE VIEW BATCHSELECT AS SELECT * FROM BATCH9 WHERE "DT >= {fromDate} .AND. DT <= {toDate}" ORDER BY DT; PRINTLN "Offspec batches for selected period"; LOOP THROUGH BATCHSELECT { IF (BATCHSELECT.OFFSPEC) PRINTLN "{BatchSelect.DT} {(BATCHSELECT.RTMS - BATCHSELECT.RTRQ)} + {BATCHSELECT.BATCH}"; } </pre> <p>This second example first creates a view of only the data that corresponds to the period of interest. I.e. between the dates <code>fromDate</code> and <code>toDate</code>. It then loops through the database and prints out a very simple report of the <code>offspec</code> records that include the date, deviation from spec and the Batch number.</p>

CREATE VIEW and DROP VIEW

Table 51: The CREATE VIEW and DROP VIEW Command

Statement	Description
<p>CREATE VIEW and DROP VIEW</p>	<p>The CREATE VIEW command creates a view of a database entity. The view is a subset of the database that you may have applied filtering and sorting to. The created view can be used in every respect like the database entity itself.</p> <p>The DROP VIEW command simply drops the view and frees up the memory. This is a command that allows for more efficient usage of the memory. The DROP VIEW command only works on view entities and dbf entities.</p>
<p>NOTE:</p>	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>Original Database Entity</p> <div style="border: 1px solid black; padding: 5px; width: 100px; margin: 0 auto;"> <p style="text-align: center;">Example BATCH9</p> </div> </div> <div style="text-align: center;"> <p>CREATE VIEW (Selects fields, filters, orders)</p> <div style="font-size: 2em;">→</div> </div> <div style="text-align: center;"> <p>View of the Entity</p> <div style="border: 1px solid black; padding: 5px; width: 100px; margin: 0 auto;"> <p style="text-align: center;">Example BATVU</p> </div> </div> </div> <p>This view is probably the most important of the SQL-like instructions and it is worth while getting to know it well.</p>
<p>format</p>	<pre>CREATE VIEW <viewName> AS SELECT <selectSpec> FROM <entity> WHERE "<filterExprn>" ORDER BY <orderSpec>; DROP VIEW <viewName>;</pre> <p style="margin-left: 200px;">Choose the fields you wish to view.</p> <p style="margin-left: 200px;">Filter out the records of interest.</p> <p style="margin-left: 200px;">Order the records.</p> <p style="margin-left: 200px;">This line is optional and drops the view.</p>
<p>example</p>	<pre>DROP VIEW TEMP_SEARCH_VIEW;</pre>

Table 52: The CREATE VIEW and DROP VIEW Command

Statement	Description		
arguments	<p>viewName</p> <p>This is the name of the new view. It can be used just like a database entity or a variable once the view has been created. For example, you can use it to display a browse widget or to print out only certain fields of a database.</p> <pre>SELECT <selectSpec></pre> <p>Use this to choose which fields or columns you want in the view. This is usually a list of the fields separated by commas. (For all fields use <code>SELECT *</code>.)</p> <pre>WHERE "<filterExprn>"</pre> <p>This is used to choose the records or rows that are to be included in the view.</p> <p>The system will evaluate the expression for each record and if the result is true, the record will be included in the view.</p> <p>Note: that the <code>filterExprn</code> must be a string that is understood by the RDBMS.</p> <p>Note: For dBase <code>WHERE</code> statements, the matching of strings will be satisfied if there is a partial match. E.g. Fetch first of <code>ENTITIES</code></p> <pre>WHERE "ENTITY = 'LC110'";</pre> <p>will match with the entity name <code>LC110B</code>. To avoid this, you must pad out the WHERE clause with a space I.e.</p> <pre>WHERE "ENTITY = 'LC110 '";</pre> <pre>(ORDER BY) <orderSpec>;</pre> <p>This specifies the order in which the records or rows are sorted in the database.</p> <p>Once again, it must be a string that the RDBMS understands. The default is ascending order.</p>		
Simple example	<table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; width: 50%;"><i>form</i></td> <td style="vertical-align: top; width: 50%;"><i>example</i></td> </tr> </table> <pre>CREATE VIEW <viewName> AS SELECT <SelectSpec> FROM <entity> WHERE "<filter Exprn>" ORDER BY <orderSpec>;</pre> <pre>CREATE VIEW NUBATCH AS SELECT DT, MSG FROM BATCH9;</pre> <p>This example creates a simple view that only contains the fields (columns) DT and MSG. It does not use the optional WHERE and ORDER BY clauses.</p>	<i>form</i>	<i>example</i>
<i>form</i>	<i>example</i>		

Table 53: The CREATE VIEW and DROP VIEW Command

Statement	Description
<p>Example 2</p>	<p><i>example</i></p> <pre>CREATE VIEW OPLOG AS SELECT * FROM OPLOG WHERE "DTOS(DT) > {DATESTR(NOW()-DAYS(30), `CCYYMMDD`)}" ;</pre> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Select all columns in the table.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Create a string from the dBase field DT Note: DTOS is a dBase function.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Create a date string corresponding to the date 30 days previous to the current date.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Add the string <code>CCYYMMDD</code> to complete the required string that is sent to the RDBMS.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Refer to the DATESTR (tstamp,format) expression description in the TIME & DATE expressions section.</p> </div> <p>The above <code>WHERE</code> clause, once executed, would create the <code>string WHERE "DTOS(DT)>19940623"</code>.</p>
<p>Example 3</p>	<p><i>example</i></p> <pre>CREATE VIEW ALARMS AS SELECT * FROM ALARMDB WHERE "ACKN = 'F' .OR. NORMAL = 'F'" ORDER BY DT, PRIORITY;</pre> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Create a view called <code>ALARMS</code> from the entity <code>ALARMDB</code>.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>View only the alarms that are either unacknowledged <i>or</i> have not returned to normal.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Order the view by <code>DT</code>, (primary order) & <code>PRIORITY</code> (secondary order.) I.e. The view is broken up into days and within each day, the alarm priorities are high priority first.</p> </div>

Table 54: The CREATE VIEW and DROP VIEW Command

Statement	Description
<p>Example 4 (more complex 3)</p>	<p>In this example, a filter pop-up window within a graphic is used to interactively create a filtered view. The window could look like this below:</p> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: fit-content;"> <p style="text-align: center;">FILTER</p> <p>filterDateFlag → <input type="checkbox"/> DATE 19980728 filterDate</p> <p>filterSectionFlag → <input type="checkbox"/> SELECTION POWER filterSection</p> <p style="text-align: center;"> <input type="button" value="CANCEL"/> <input type="button" value="OK"/> </p> </div> <p style="text-align: right;">This triggers the metascript</p> <p>The following meta script would be executed once the OK button is pressed.</p> <p><i>example</i></p> <pre> IF (filterDateFlag) LET oplogFilter = "OPDATE = {CTOD({filterDate})"; ELSE oplogFilter = "T"; IF (filterSectionFlag) LET oplogFilter = oplogFilter + " .AND. SECTION = " + filterSection; CREATE VIEW OPVIEW AS SELECT * FROM OPLOG WHERE oplogFilter ORDER BY OPDATE, OPTIME; </pre> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>If the filterDateFlag check box is depressed, create a DATE component to the filter string.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Otherwise, just set the date component of the string to true. This will show all dates.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>If the SECTION checkbox is depressed, then add a section (component) to the filter string. Note that the .AND. means both criteria must be satisfied.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>You could add further components to the filter string as desired.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Now that the filter string has been created by the preceding code, it can be used easily within the WHERE clause.</p> </div>

Table 55: The CREATE VIEW and DROP VIEW Command

Statement	Description
<p>Example 5 (One to Many View)</p>	<p>This view displays only those records that are associated with a given value</p> <p><i>example</i></p> <pre>CREATE VIEW CUSTVIEW AS SELECT * FROM BATCH9 WHERE "CUSTOMER = `{CUSTOMER.NAME} `";</pre> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>An entity called BATCH9 has been created which looks at the batch9.dbf database.</p> </div> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>The WHERE clause filters out only those batches from the BATCH9 database that are associated with the customer currently being pointed at in the customer database.</p> </div> <div style="text-align: center; margin: 10px 0;"> <p>The diagram illustrates the data flow for the view. On the left, a table labeled 'CUSTOMER' has a dropdown arrow pointing to its first row. A box labeled 'NAME field of database CUSTOMER' points to this dropdown. An equals sign follows. To the right, a table labeled 'BATCH9' has a dropdown arrow pointing to its first row. A box labeled 'CUSTOMER field of database BATCH9' points to this dropdown. Arrows from both 'CUSTOMER' and 'BATCH9' tables point to a box labeled 'custView'. Below 'custView' is another table with three rows. A box containing the text 'Only those records that have the same customer name as the current record in the CUSTOMER database are shown.' has an arrow pointing to the 'custView' box.</p> </div> <p>Only those records that have the same customer name as the current record in the CUSTOMER database are shown.</p> <p>This can be used to dynamically update a Batch browse widget as the customer browse widget is stepped through.</p>
<p>Special notes on Views.</p> <p>Graphics Use</p> <p>Browse Use</p> <p>Fetch</p>	<p>You may use VIEWS in exactly the same way you would use attributes and entities. I.e. once you have created a view and given it a viewName, you can use the attributes (I.e. fields or columns) in value updates, bars, pies, edit areas etc.</p> <p>You can browse a VIEW in exactly the same way as a database entity.</p> <p>Use the FETCH command to position the current record in the view. Note that the value used in the graphics (i.e. bars, values etc.) is the value at the current record.</p>

CREATE VIEW FROM FILE <filename>

Table 57: The CREATE VIEW FROM FILE <filename> Command

Statement	Description
<p>CREATE VIEW FROM FILE <filename></p> <p>format</p> <p>arguments</p>	<p>The <code>CREATE VIEW.... FROM FILE <fileName></code> command is identical to the <code>CREATE VIEW.... FROM <entity></code> command with the exception that an Entity is not required. The command is particularly useful where files are being created regularly for example as history files and you do not wish to add new entities to the entities database.</p> <p>format</p> <pre>CREATE VIEW <viewName> AS SELECT <selectSpec> FROM FILE "<fileName>" WHERE "<filterExprn>" ORDER BY <orderSpec>;</pre> <p style="text-align: right;">→ optional</p> <p>fileName</p> <p>This is the name of the database. The file name must be a string and it may be direct or in relation to the MACRODIR environment variable.</p> <p><i>For example:</i></p> <pre>/u/macro/archive/hist4.dbf</pre> <p>or</p> <pre>../archive/hist4.dbf</pre> <p>Other arguments</p> <p>All the other arguments are identical to the standard <code>CREATE VIEW FROM <entity></code> command.</p>
<p>example</p>	<p>example</p> <pre>LET fileName = "HIST" + STR(MONTH()) + ".dbf"; CREATE VIEW CURMONTH AS SELECT * FROM FILE "../hist/" + fileName;</pre> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Now use the string variable to access the current history file.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>First create a string that uses the last characters as the month number. I.e. APRIL is HIST4</p> </div> <p>Refer to "String Related Functions" on page82 for an explanation of the <code>STR</code> command and "Database Functions" on page48 for information on the <code>MONTH()</code> command.</p>

CREATE TABLE VARIABLE

Table 58: The CREATE TABLE VARIABLE Command

Statement	Description																
<p>CREATE TABLE VARIABLE</p> <p>arguments</p>	<p>The CREATE TABLE VARIABLE command is used to set up a special kind of variable that has the same characteristics as a database. This provides a convenient structure for storing and viewing, (e.g, using the browse widget) tabular data that you don't want stored on the disk.</p> <p><code>tableName</code></p> <p>This is the name of the table. You may use this name in the same way as a view or an entity. It must be upper case.</p> <p><code>fieldName1 & fieldNameN</code></p> <p>These are the field or column names of the table variable. They may be referenced as attributes in the normal way and they must be uppercase.</p> <p><code>fieldFormat1 & fieldFormatN</code></p> <p>These are the format specifications of the new fields. They can be either of the following forms:</p> <p>NUMBER a number. (An integer.)</p> <p>VARCHAR a string.</p>																
<p>Format 1</p>	<table border="0"> <tr> <td style="vertical-align: top;"><i>form1</i></td> <td style="vertical-align: top;"><i>example</i></td> </tr> <tr> <td><code>CREATE TABLE VARIABLE</code></td> <td><code>CREATE TABLE VARIABLE</code></td> </tr> <tr> <td><code><tableName> FIELDS</code></td> <td><code>LABDATA FIELDS</code></td> </tr> <tr> <td><code>fieldName1fieldFormat1,</code></td> <td><code>TM VARCHAR,</code></td> </tr> <tr> <td><code>fieldName2fieldFormat2,</code></td> <td><code>PH NUMBER,</code></td> </tr> <tr> <td><code>.</code></td> <td><code>SAMPLE1 NUMBER,</code></td> </tr> <tr> <td><code>.</code></td> <td><code>PERIOD NUMBER,</code></td> </tr> <tr> <td><code>fieldNameNfieldFormatN;</code></td> <td><code>COMMENT VARCHAR;</code></td> </tr> </table> <p>This example creates a table variable whose fields are:</p> <p>TM Characters</p> <p>PH, SAMPLE1 Numbers</p> <p>SAMPLE2 A number</p> <p>PERIOD A number</p> <p>COMMENT A variable sized character buffer. (I.e. a string.)</p> <p>The table is empty (I.e. it has no records) when it is first created. You can use the <code>INSERT INTO <entity></code> command to add records to it.</p>	<i>form1</i>	<i>example</i>	<code>CREATE TABLE VARIABLE</code>	<code>CREATE TABLE VARIABLE</code>	<code><tableName> FIELDS</code>	<code>LABDATA FIELDS</code>	<code>fieldName1fieldFormat1,</code>	<code>TM VARCHAR,</code>	<code>fieldName2fieldFormat2,</code>	<code>PH NUMBER,</code>	<code>.</code>	<code>SAMPLE1 NUMBER,</code>	<code>.</code>	<code>PERIOD NUMBER,</code>	<code>fieldNameNfieldFormatN;</code>	<code>COMMENT VARCHAR;</code>
<i>form1</i>	<i>example</i>																
<code>CREATE TABLE VARIABLE</code>	<code>CREATE TABLE VARIABLE</code>																
<code><tableName> FIELDS</code>	<code>LABDATA FIELDS</code>																
<code>fieldName1fieldFormat1,</code>	<code>TM VARCHAR,</code>																
<code>fieldName2fieldFormat2,</code>	<code>PH NUMBER,</code>																
<code>.</code>	<code>SAMPLE1 NUMBER,</code>																
<code>.</code>	<code>PERIOD NUMBER,</code>																
<code>fieldNameNfieldFormatN;</code>	<code>COMMENT VARCHAR;</code>																

Table 59: The CREATE TABLE VARIABLE Command

Statement	Description
format 2	<p><i>form2</i> <i>example</i></p> <pre>CREATE TABLE VARIABLE <tableName> OF fieldFormat SIZE <number of fields>;</pre> <p style="text-align: right;"><pre>CREATE TABLE VARIABLE LABDATA OF NUMBER, SIZE 100;</pre></p> <p>This example creates a table variable which has numeric fields: ATTR1 to ATTR100.</p> <p>For the type, you can use the standard types of field. I.e.: These are the format specifications of the new fields. They can be either of the following forms:</p> <p>NUMBER A number. (An integer.)</p> <p>VARCHAR A string.</p>
Use	<p>You can use the table variable like any other view or entity I.e.</p> <p>In graphics</p> <ul style="list-style-type: none"> • With a chart or browse widget. • In meta scripts & graphics. • By referencing the <code>ENTITY.ATTR</code>, <code>ENTITY.ATTR[N]</code> to get the data from the current record or <code>N</code> records back from the current. • With the <code>FETCH</code> command. <p>I.e. using <code>FETCH FIRST</code>, <code>LAST</code>, <code>PREVIOUS</code> or <code>NEXT</code> commands.</p> <ul style="list-style-type: none"> • Other Database commands, <p>Such as <code>DELETE</code>, <code>UPDATE</code>, <code>INSERT</code> and <code>LOOP THROUGH</code>.</p>
List of the Table Variable	<p>The table variable only exists for the life of the session. I.e, because the information is not stored in a file on the hard disk, once the program is exited, the data is lost.</p>
Restrictions	<p>You cannot:</p> <p>Use the <code>WHERE</code> clause in <code>FETCH</code>, <code>UPDATE</code> and <code>DELETE</code> commands you will get an <code>SqlNotCapable</code> error if you use the <code>DBERR()</code> return value.</p> <ul style="list-style-type: none"> • Create a view of a table variable. <p>Change the format of a table variable without issuing a new <code>CREATE TABLE VARIABLE</code>.</p>
Typical use	<p>The table variable is typically used as a buffer to view information once it has been collected. For example, if you read data out of a text list, you can put it in a table variable and use the browse widget to view it.</p>

CREATE HISTORICAL VIEW

Table 60: The CREATE HISTORICAL VIEW Command

Statement	Description
<p>CREATE HISTORICAL VIEW</p> <p>Background</p>	<p>The <code>CREATE HISTORICAL VIEW</code> command creates a view of the <i>MacroView</i> historical database.</p> <p>The VIEW can be used in the same way as any database view or entity. I.e. it can be used in chart and browse widgets, in meta scripts etc.</p> <p>The <i>MacroView</i> historic database structure is a proprietary structure designed to optimize the disk utilization for large amounts of historic data.</p> <p>It also incorporates an in-built archival feature.</p> <p>For more information on the history capabilities, refer to the chapter on History in the Engineering Manual.</p>
<p>Format</p>	<p><i>format</i></p> <pre> CREATE HISTORICAL VIEW <tableName> AS SELECT <fieldName1>=entity1.attr1, <fieldName2>=entity2.attr2, . . . <fieldNameN>=entityN.attrN, WITH <period>(<duration>) <hForm> WHERE TIME > = <startTstamp> AND TIME < = <endTstamp>; </pre>

Table 61: The CREATE HISTORICAL VIEW Command

Statement	Description
arguments	<p><code>tableName</code></p> <p>This is the name of the table. You may use this name in the same way as you would use a view or a database entity. It must be uppercase letters.</p> <p><code>fieldName 1 & fieldNameN</code></p> <p>These are the field (or column names) that are to be created in the table to hold the historic data. They could be names such as <code>FLOW</code>, <code>SPEED</code>, <code>TIME</code> or you could create them dynamically using the <code>EXECUTE</code> command.</p> <p><code>entity1.attr1 & entityN.attrN</code></p> <p>These are the attributes of entities that have been stored in the historical database. You must configure the historian to collect this information.</p> <p>Refer to the History chapter in the Engineering Manual.</p>

Table 62: The CREATE HISTORICAL VIEW Command

Statement	Description
Note	The resulting table will also contain an additional column (the first column) called TM. This will represent the actual point in time that the historical data represents. It will be of the time stamp form as specified in the SET FORMAT TO command.
Format	<p><code>WITH <period>(<duration>) <hForm>:</code></p> <p>For example: <code>WITH MINUTES(1) AVERAGES</code> or <code>WITH SECONDS(6) SAMPLES</code> or <code>WITH HOURS(8) MAXIMUMS</code> etc.</p> <p>You can only specify <code><period>(<duration>) <hForm></code> combinations that have been specified in the historical specification configuration. Refer to the History chapter in the Engineering Manual.</p> <p>Note that <code><period></code>, <code>(<duration>)</code> and <code><hForm></code> can all be expressions.</p>
Format	<p><code>WHERE TIME >=<startTstamp></code> <code>AND TIME <=<endTstamp>;</code></p> <p>The optional WHERE clause specifies the range in time of the resulting view. Both <code><startTstamp></code> and <code><endTstamp></code> must be expressions that result in time stamp strings. (Refer to the section "Time and Date Functions" on page88.)</p> <p>You can use various forms of this expression I.e.,</p>
Example 1	<p><code>WHERE TIME >= <startTstamp></code> <code>AND TIME <= <endTstamp>;</code> <code>WHERE TIME >= <startTstamp>;</code></p> <p>The system will assume that the end point is NOW and will allow the table to grow as time progresses.</p>
Example 2	<p><code>WHERE TIME <= <endTstamp>;</code></p> <p>The system will provide a start time for you.</p>
Example 3	<p>If there is no WHERE clause, the system will choose a start time and will use NOW as the end time.</p>
	<p>Note: You cannot use the general form of the WHERE clause with historical views.</p>

Table 63: The CREATE HISTORICAL VIEW Command

Statement	Description
<p>example</p>	<p><i>example</i></p> <pre>CREATE HISTORICAL VIEW TANK9HST AS SELECT LEVEL=L109.PV, WATER= FW109.PV, CAUSTIC=FC109.PV WITH SECONDS(15) SAMPLES WHERE TIME >=(NOW() - HOURS(15));</pre> <div data-bbox="1155 344 1430 450" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Creates a view called TANK9HST</p> </div> <div data-bbox="1155 461 1430 645" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Creates fields RECNO, DT, TM, LEVEL, WATER & CAUSTIC</p> </div> <div data-bbox="523 786 1104 913" style="border: 1px solid black; padding: 5px; margin-top: 20px;"> <p>The earliest record will be 15 hours ago. The latest record will be now. The view will dynamically update every 15 seconds.</p> </div>
<p>Uses</p> <p>Restrictions</p> <p>Missing data</p>	<p>You may use the table:</p> <p>In a browse or chart widget.</p> <p>In a real-time graphic using the entity.attr format (but not the attr[n](entity) format.)</p> <p>With the <code>FETCH</code>, <code>FIRST</code>, <code>LAST</code>, <code>NEXT</code> and <code>PREVIOUS</code> commands. (But not <code>FETCH WHERE</code>.)</p> <p>With the <code>LOOP THROUGH</code> command.</p> <p>You may not:</p> <p>Use <code>DELETE</code>, <code>UPDATE</code> and <code>INSERT</code> commands.</p> <p>Use the <code>WHERE</code> clause in <code>FETCH</code> commands.</p> <p>Use the <code>CREATE VIEW</code> on a historical table.</p> <p>The HISTORICAL VIEW will always include a record for every configured time period. If the data is not present, the HISTORICAL VIEW will include a blank record.</p>

DBERR and DBERRSTR

Table 64: The DBERR and DBERRSTR Command

Statement	Description
<p>DBERR and DBERRSTR</p> <p>Format</p> <p>arguments</p>	<p>These commands are used to inform you if an error occurred in any of the database commands.</p> <p>DBERR returns a numeric code associated with the kind of error and DBERRSTR returns a string that describes the error.</p> <p>In each case, there is a code or a string associated with the last command associated with a database entity, view, table variable or historical view.</p> <p>If there was no error, a zero is returned for the code and an empty string for the DBERRSTR.</p> <p><code><entity>.DBERR</code> <code><entity>.DBERRSTR</code></p> <p>For a list of error codes, refer to the table "DB Error Codes and Strings" on page 75.</p> <p>entity:</p> <p>The returned error code or string refers to the last command associated with this entity, view or table variable.</p> <p>E.g. OPLOG.DBERR will return a code that relates to the success or failure of the last database command used on the database entity called OPLOG.</p>

Table 65: The DBERR and DBERRSTR Command

Statement	Description
<p>example</p>	<pre> <i>example</i> INSERT INTO OPLOG FIELDS DT, TM, ENTRY VALUES DATESTR(NOW()), TIMESTR(NOW()), entryValue; IF (OPLOG.DBERR) PRINTLN "Error adding record to operator logs database.\n" + OPLOG.DBERRSTR; </pre> <div data-bbox="1150 376 1430 678" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>This returns a code that is non-zero if the last command, (in this case, the INSERT command just performed,) on the entity OPLOG was not successful.</p> </div> <div data-bbox="1150 741 1430 920" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>This returns a non-empty string if the INSERT command was not successful.</p> </div> <div data-bbox="695 983 1171 1043" style="border: 1px solid black; padding: 5px; margin-top: 10px; text-align: center;"> <p>This message will be printed out.</p> </div>
<p>Example 2</p>	<pre> <i>example</i> FETCH NEXT OF OPLOG; IF (OPLOG.DBERR = SqlAtBottom) PRINTLN "You have reached the bottom of the " + "operator log database.\n"; </pre> <div data-bbox="667 1402 1270 1585" style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>In this example, we are looking specifically for the end of the file message. Note: that you can use the numeric code 20 or the name of the variable <code>SqlAtBottom</code> in the IF statement.</p> </div>

DB Error Codes and Strings

Table 66: DB Error Codes and Strings

Error Code	Reference Label	Description
0	SqlSuccess	The success indicator code.
-1	SqlError	A general error indication. In this case the error situation is not associated with a particular error number. The error description can be obtained via the DBERRSTR() function.
1	SqlNotAvailable	The functionality requested is not available from the entity referred to. E.g. inserting a row into a PLC entity.
10	SqlNoFile	The database or table specified in the command does not exist.
11	SqlNoPermission	The user does not have the appropriate permission level to perform the database command.
20	SqlAtBottom	A fetch next command was issued and it couldn't be satisfied because the system is at the last record.
21	SqlAtTop	A fetch previous command was issued and it couldn't be satisfied because the system is at the first record.
30	SqlUniqueKey	The database operation could not be performed because it would violate the configured requirement that a particular key (i.e. indexed field) was unique.
40	SqlLocked	The database operation could not be performed because a lock was present on the database in question and was not released within the internal time out period.
50	SqlLimit	The database operation could not be performed because an internal limit was reached. E.g. maximum number of records for this database was reached.

10.12 Operators and Functions

The meta script language supports a rich set of operators and functions which can be used within the commands and expressions. The summary below shows the functions used most often in bold. The functions have been separated in the same order as the tables in this section.

OPERATORS and FUNCTIONS	
OPERATORS	** ^ * / % - + = <> # < > <= >= \$.AND. .OR. .NOT.
MISCELLANEOUS	ABS BETWEEN CEILING FLASH FLOOR IIF INT MIN MAX OS OSVERSION RAND ROUND SQRT SQWAVE VERSION DBERR DBERRSTR
STRING FUNCTIONS	ALLTRIM ASC AT CHR ISALPHA ISLOWER ISNUMERIC ISUPPER LEFT LEN LOWER LTRIM PROPER RAT REPLICATE RIGHT RTRIM SPACE STR STUFF SUBSTR UPPER VAL
TRIGONOMETRIC	ACOS ASIN ATTAN COS SIN TAN DTOR PI RTOD COSH SINH TANH EXP LOG LOG10
BINARY	BINAND BINNOT BINOR BINXOR BIT

Operators

Table 67: Operators

Type	Symbol	Example	Result	Precedence	Comment
Exponentiation	** or ^	LIC100.PV^2	201.64	7	Note: The use of the carat (^) may conflict with AutoSketch when used in AutoSketch drawings.
Divide	/	LIC100.PV/2	7.1	6	
Multiply	*	LIC100.PV*2	28.4	6	
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	"STAT US = 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	

Table 68: Operators

Type	Symbol	Example	Result	Precedence	Comment
{ } Curly Brackets (or Braces)	{ }	"{L109.PV}"	value of L109.PV	4	Creates a string whose numeric value is equal to the value of the entity.attribute or variable. Refer to the note below. **
Contain	\$	"CD" \$ "ABCD"	1	4	Returns true if first string is contained in second string.
AND	AND .AND.	PMP43.PV .AND. PMP44.PV	0	3	You must have a space in front of and following the .AND.
OR	OR .OR.	PMP43.PV .OR. PMP44.PV	1	2	You must have a space in front of and following the .OR.
NOT	NOT .NOT.	.NOT. PMP43.PV	0	1	You must have a space in front of and following the .NOT.

** The evaluation of string variables can be simplified with the curly bracket (braces) notation.

I.e. { and }. The curly brackets show parts of the expression that must be evaluated first. Use the curly brackets to evaluate the variables or entity.attributes in a string first. For example:

```
LET attrSet= "TYPE = `{TypeStr}` .AND. ATTRIBUTE = `{UPPER(SetRequest)}";
```

The {} have the highest precedence and are evaluated left to right prior to the remainder of the expression.

You may only use the curly Brackets inside the double quotes. If the curly brackets are encountered outside strings, they are used to block statements for IF statements for example.

Special *MacroView* functions

Table 69: Special *MacroView* Functions

Function	Example	Comment
RAMPDELTA("ent.attr")	<code>RAMPDELTA("L109.SV")</code>	Returns the ramp delta for an attribute. I.e. how much the attribute may be incremented with the up and down keys.
CONFIRMSET("ent.attr")	<code>CONFIRMSET("L109.SV")</code>	Returns a 1 if the entity.attribute has the confirm flag set true. This will usually mean that, when in the Navigator, a message is sent to the operator asking for confirmation that an attribute must be set.
LOGSET("ent.attr")	<code>LOGSET("L109.SV")</code>	Returns a 1 if the entity.attribute has the log set flag set true. This will usually mean that a message is written to the messages database indicating that the variable has been changed.
ENTEXISTS	<code>ENTEXISTS("LC110")</code>	Returns a 1 if the entity VIEW or table exists.
ATTREXISTS	<code>ATTREXISTS("LC110.PV")</code> E.g. <code>LET exists =</code> <code>ATTREXISTS("LC110"</code> <code>+ attrName);</code>	Returns a 1 if the attribute of the entity exists.

Note: You may use the \$ Wildcard in any of the expressions. For example, in a group display, you may refer to **RAMPDELTA(\$.PV)** Here the \$ refers to the default entity within the metafile being executed.

See `SEND "SetDefaultEntity" TO "<display area>";`

In display area section of graphics chapter.

Miscellaneous Functions

Table 70: Miscellaneous Functions

Function	Example	Comment
ABS(number)	ABS(13.4-34.6) 21.2	Absolute of a real number. I.e. remove negative sign information if present.
BETWEEN(x, x1, x2)	BETWEEN(5, 0, 10) 1 I.e. true	Returns an indication of whether the first argument x is within the range specified by the second and third arguments x1 and x2 I.e. x >= x1 .AND. x <= x2. Note: that the arguments must be all of the same value type but can be numbers, strings, time stamps or time duration's.
CEILING(number)	CEILING(12.37) 13	Returns the smallest integer that is greater than or equal to the numeric argument.
GETENV(string)	GETENV("MACRODIR")	This function returns a string that is the value of the environment variable.
FLASH(number)	FLASH(10) Returns 10 then 0 then 10 then 0 etc.	This function cyclically returns the specified number and then zero continuously. The cycle period is designed for use in graphic displays where graphics need to flash to illustrate a state.
FLOOR(number)	FLOOR(12.37) 12	Returns the largest integer that is less than or equal to the numeric argument.
INT(number)	INT(12.37) 12	Returns the integer component of a number.
MIN(x, y, ...)	MIN(12.7, 26, 5.3) 5.3	Returns the minimum value in a series of numbers.
MAX(x, y, ...)	MAX(12.7, 26, 5.3) 26	Returns the maximum value in a series of numbers.
OS()	OS() "SCO Unix"	Returns the name of the operating system as a string.
OSVERSION()	OSVERSION() "3.2.4"	Returns the version number of the operating system as a string.
RAND(range)	RAND(99.9) 80.1	Returns a random number that lies between 0 and the range specified.
ROUND(num, decimals)	ROUND(12.37, 1)	Returns the numeric argument

Function	Example	Comment
	12.4	rounded of to a certain number of decimal places.
SQRT(x)	SQRT(9) 3	Returns the square root of the numeric argument.
SQWAVE(x,y)	SQWAVE (3, 1.5) 1 for 3 seconds 0 for 1.5 seconds 1 for 3 seconds etc.	Returns a 1 for x seconds and a 0 for y seconds repeatedly.
VERSION()	VERSION() "3.0.4"	Returns the <i>MacroView</i> version number as a string.
DBERR(<entity>) DBERR(<view>)	DBERR(OPLOG) 21	Returns the error number resulting from the execution of the last database command. A zero is returned if no error occurred. See the "Database Functions" on page48 for details.
DBERRSTR(entity) DBERRSTR(view)	DBERRSTR(OPLOG) "OPLOG Error 10: Opening database"	Returns the error description string resulting from the execution of the last database command. This is an empty string if no error occurred. See "DBERR and DBERRSTR" on page73 for details.

String Related Functions

Table 71: String Related Functions

Function	Example	Comment
ALLTRIM(string)	ALLTRIM(" Hello ") "Hello"	Strips both leading and trailing spaces from a string and returns the result.
ASC(string)	ASC("ABCDE") 65	Returns the ASCII code for the first character found in the string .
AT(s1, s2, number) AT(s1, s2)	AT("AUT", "Going to AUTO") 10	This function searches for string s1 within string s2 . If it finds an occurrence it returns the character position of string s1 within s2 . The optional numeric argument identifies which occurrence of s1 in s2 is required, e.g. the first, second, third etc. occurrence.
CHR(number)	CHR(66) "B"	Returns a single character string where the character used is based on the ASCII character equivalent of the number passed as an argument.
ISALPHA(string)	ISALPHA("AUT") 1 I.e. true	Returns true if all of the characters in the string are alphabetic characters. I.e. a-z or A-Z. Spaces are ignored.
ISLOWER(string)	ISLOWER("AUT") 0 I.e. false	Returns true if all of the characters in the string are lower case alphabetic characters. Spaces are ignored.
ISNULL(ent.attr)	ISNULL(LDB.VL1)	Will return a 1 (true) value only if the entity.attribute holds a null field. This is useful for reports. Refer to "LET" on page18.
ISNUMERIC(string)	ISNUMERIC("AUT") 0 I.e. false	Returns true if all of the characters in the string are valid numeric characters (E.g. 0-9 .-+E). Spaces are ignored.
ISUPPER(string)	ISUPPER("AUT") 1 I.e. true	Returns true if all of the characters in the string are upper case alphabetic characters. Spaces are ignored.
LEFT(string, n)	LEFT("ABCDE", 2) "AB"	Returns the n left most characters in the string .
LEN(string)	LEN("ABCDE")	Returns the character length of the string passed as an argument.

Function	Example	Comment
	5	
LET ent.attr = NULL	LET LDB.V1=NULL;	Sets the value of the Entity.Attribute to null.
LOWER(string)	LOWER("ABCDE") "abcde"	Returns a string with all lower case characters in it based upon the string passed as an argument.
LTRIM(string)	LTRIM(" Hello ") "Hello "	Strips leading spaces from a string and returns the results.
PROPER(string)	PROPER("fred bloggs") "Fred Bloggs"	Returns a string with the first character of every word capitalized based upon the string passed as an argument. Non alphabetical characters are not affected.
PAD(string, length)	PAD("Production",15) + PAD("Report",0); "Production Report"	Spaces will be added to the first argument to make the length of the string equal to the length specified in the second argument.
RAT(s1, s2, number) RAT(s1, s2)	RAT("AUT", "Going to AUTO") 4	This function operates in a similar way to the AT() function except that it searches starting from the end of the string s2 as opposed to the beginning.
REPLICATE(string, n)	REPLICATE("AB" , 3) "ABABAB"	Returns a string that is a replication of the string passed as an argument. The string is replicated n times.
RIGHT(string, n)	RIGHT("ABCDE" , 2) "DE"	Returns the n right most characters in the string .
RTRIM(string)	RTRIM(" Hello ") " Hello"	Strips trailing spaces from a string and returns the results.
SPACE(number)	SPACE(5) " " I.e. a string with 5 spaces	Returns a string with the specified number of spaces in it.
STR(num, width, prec) STR(num, width) STR(num)	STR(12.435, 5, 1) " 12.4"	Converts the numeric value num into a string. The string returned will have the specified width and the string representation will have prec decimal places. The width and precision specifications are optional. If the precision is not specified then the system will choose a precision to use. If the width is not specified then however many characters are needed to represent the number by a

Function	Example	Comment
		string are used. If the specified width is insufficient to represent the number then fewer decimal places will be used. If the width is still insufficient when using no decimal places then the string will be filled with asterisks to indicate that it was not possible to represent the number given the character width specified.
STUFF(s1, n1, s2, n2) STUFF(s1, n1, s2)	STUFF("Going AUT", 7, "Automatic") "Going Automatic"	This function replaces a section of one string s1 with another string s2 . The argument n1 is the starting point for replacement. The argument n2 is optional and identifies the number of characters to replace. If not specified the length of string s2 is used as a default.
SUBSTR(str, offset,num)	SUBSTR("ABCDE", 2, 3) "BCD"	Returns a substring of the string str . The substring is taken as the num characters starting at the offset position.
UPPER(string)	UPPER("abcDe") "ABCDE"	Returns a string with all upper case characters in it based upon the string passed as an argument.
VAL(string)	VAL(" 27.5") 27.5	Returns a numeric value based upon a string passed as an argument. Note: that if a nonsense string is passed then the return value will be a zero.

Trigonometric, Hyperbolic and Logarithmic Functions

Table 72: Trigonometric Functions

Function	Example	Comment
ACOS(x)	ACOS(0.87) 29.541361	Arc cosine with result in units of degrees.
ASIN(x)	ASIN(0.5) 29.541361	Arc sine with result in units of degrees.
ATAN(x)	ATAN(0.58) 30.113733	Arc tangent with result in units of degrees.
COS(x)	COS(30) 0.87	Trigonometric cosine with argument in units of degrees.
SIN(x)	SIN(30) 0.5	Trigonometric sine with argument in units of degrees.
TAN(x)	TAN(30) 0.58	Trigonometric tangent with argument in units of degrees.
DTOR(x)	DTOR(60) 1.04720	Degrees to Radians conversion function. The trigonometric functions work in degrees as a match to most user's requirements. The degrees/radians conversion functions provides a convenient means of converting between the two.
PI()	PI() 3.1415927 etc.	Returns a value for p to 12 decimal places.
RTOD(x)	RTOD (1.04719)60	Radians to Degrees conversion function.
COSH(x)	COSH(1.5) 2.3524	Hyperbolic cosine.
SINH(x)	SINH(1) 1.1752	Hyperbolic sine.
TANH(x)	TANH(3.6) 0.9985	Hyperbolic tangent.
EXP(x)	EXP(1) 2.7183	Exponential power.
LOG(x)	LOG(12) 2.4849	Logarithm (base e).
LOG10(x)	LOG10(12) 1.0792	Logarithm (base 10).

Binary Arithmetic Functions

Table 73: Binary Arithmetic Functions

Function	Example	Comment
BINAND(x, y ...)	BINAND(37, 15) 5	Performs a binary AND operation on a series of integer arguments.
BINNOT(x)	BINNOT(37) -38	Performed a binary NOT operation on an integer. The NOT calculation is performed with reference to a 32 bit word regardless of the word size of the hardware platform being used.
BINOR(x,y ...)	BINOR(37, 15) 47	Performed a binary OR operation on a series of integer arguments.
BINXOR(x,y ...)	BINXOR(37, 15) 42	Performed a binary XOR operation on a series of integer arguments.
BIT(bit, integer)	BIT(3, 37) 1	Returns the value of the specified bit number in the specified integer . The least significant bit is considered bit 1.

10.13 Time and Date Functions

Time and Date manipulations are extremely important in the SCADA, Process Control and Manufacturing Industries. To support this requirement, the meta script language offers a variety of functions and operators that are specific to time, date and duration manipulation. These functions are shown in the diagram below and discussed in more detail in the remainder of this section.

TIME and DATE FUNCTIONS								
TIME and DATE OPERATORS	/	Divide	=	Equal to	<=	Less than or equal		
	*	Multiply	<> or #	Not equal to	>=	Greater than or equal		
	%	Remainder	<	Less than				
	-	Subtract	>	Greater than				
	+	Add						
TIME and DATE FUNCTIONS	ABS	BETWEEN	CDOW	CMONTH				
	CVTDATE	CVTDURATION	CVTTIME	DATE				
	DATESTR	DAY	DAYS	DMY				
	DOW	DOY	DURATION	DURATIONSTR				
	HOUR	HOURS	MDY	MINUTE				
	MINUTES	MONTH	MONTHS	NOW				
	SECOND	SECONDS	TIME	TIMESTR				
	WOY	YEAR	YEARS					
TIME and DATE ..FORMAT TO	Year	Month	Day	Hour	Minute	Second	Millisec	Other
	CC	MMMM	DDDD	HH	mm	ss	iii	am
	YY	MMM	DDD	hh	%m	%s	ii	AM
	%Y	MM	Ddd	%H			i	pm
		%M	%D	DD			%i	PM

Time, Date and Duration Concepts

In addition to strings and normal numeric data handling, there are two additional data formats available that are specific to time and date manipulation.

These data formats are discussed below:

i. Time Stamp Format

Data in the time stamp format refers to a particular instant in time.

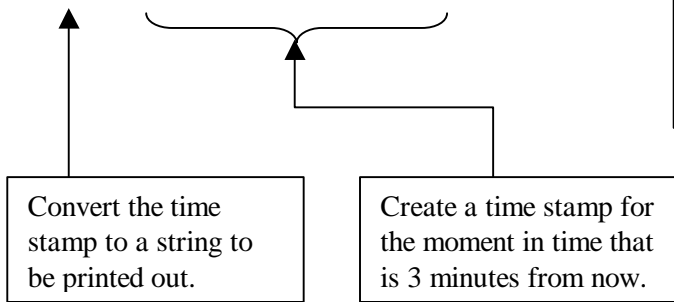
E.g. 8.32pm on Friday, 12th June.

The meta script stores the time stamp down to the millisecond resolution.

You can use the time stamp data in the various functions discussed in the tables in this section.

For example, the function NOW() returns a time stamp relating to the current time down to the millisecond resolution. A further example:

```
PRINTLN "Timeout is at " +  
TIMESTR(NOW() + MINUTES(3));
```



Note: You can also specify the format of the time printout using the **SET TIME FORMAT** command, described in the section of this chapter, section 10.9.

Note: For operating system reasons, you cannot specify a time stamp before 1970.

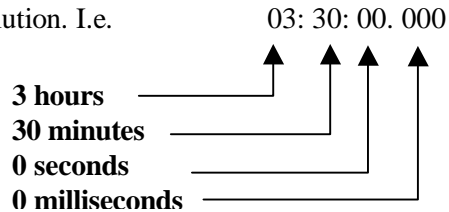
ii. Duration Format

In contrast to the time stamp format which represents an instant in time, the duration format represents a period of time once again down to the millisecond in resolution. You can use the duration format in the functions discussed in the tables in this section.

For example:

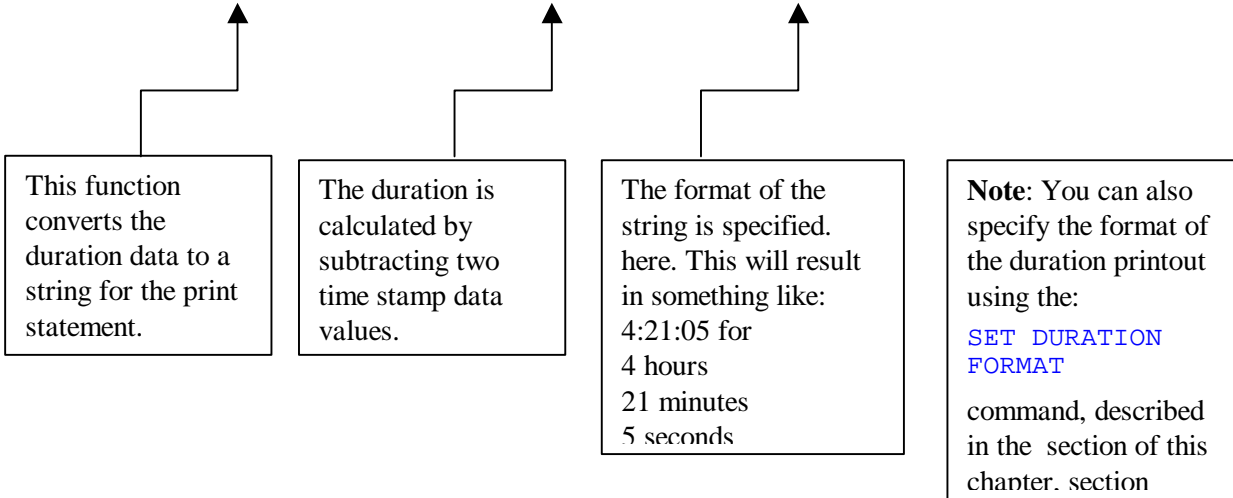
HOURS(3.5)

Will return a duration down to the millisecond resolution. I.e.



As another example, you can print out a *period* or *duration* using:

```
PRINTLN "Heating period was " +
    DURATIONSTR(NOW() - heatPeriod), "HH:mm:ss");
```



Combining Different Formats

Using the functions below, you can combine different formats in expressions. There are, however, definite rules that relate to how these expressions are used.

For example:

It is possible to subtract one *time stamp* from a second time stamp to produce a *duration*, However, adding two *time stamps* together is meaningless and is not allowed. It is most important to be aware of the format of the data.

I.e. is it a *string*, *numeric*, *time stamp* or a *duration*?

The next tables summarize the various functions, operators and formats relating to *time* and *date* manipulation.

DATE and TIME Functions

Table 74: Date and Time Functions

Function	Example	Comment
ABS(duration)	<code>ABS(EndTime - startTime)</code> <code>05:00:00</code>	Returns the absolute value of a time duration. Time duration's can have a negative component. E.g. 5 hours back in time. This version of the ABS() function removes the negative association with the time duration value and returns the result.
BETWEEN(t1, t2, t3) BETWEEN(d1, d2, d3)	<code>BETWEEN(StartTime, Now(), EndTime)</code> <code>1</code> I.e. true	Returns an indication of whether Now() is between the startTime and the endTime. This is equivalent to <code>t1 >= t2 AND t1 >= t3</code> . A similar form of functionality is applied when all arguments are time duration's.
COW(timestamp) COW()	<code>COW(DATE(1993, 8, 18))</code> <code>"Wednesday"</code>	Returns a string representation of the day of the week of the given time stamp value provided as an argument. If no argument is provided then the current day of the week is returned.
CMONTH(timestamp) CMONTH()	<code>CMONTH(NOW())</code> <code>"October"</code>	Returns a string representation of the month name of the time stamp argument. If no argument is specified then the current month is returned.
CVTDATE(datestr)	<code>CVTDATE("19910212")</code> <code>02/12/92</code>	Converts a string argument representing a point in time to a time stamp based on the default format string specified by the SET DATE FORMAT TO command.
CVTDATE (timestr, format)	<code>CVTDATE("19910212", "CCYYMMDD")</code> <code>Feb 12th 1991</code>	Converts a string (the first argument) representing a point in time to a date, (i.e. timestamp) based on a format string, (the second argument).
CVTDURATION (timestr)	<code>CVTDURATION("12:52:10")</code> <code>12:52:10.000</code>	Converts a string (the first argument) to a duration of time based on the default format string specified in the SET DURATION FORMAT TO.
VTDURATION (timestr, format)	<code>CVTDURATION("12:52:10", "HH:mm:ss")</code> <code>12:52:10.000</code>	Converts a string (the first argument) to a duration of time

Function	Example	Comment
		based on a format string (the second argument).
CVTTIME (timestr, format)	CVTTIME("19910212", "CCYMMDD") Feb 12th 1991	Converts a string (the first argument) representing a point in time to a time stamp based on a format string (the second argument).
CVTTIME (timestr)	CVTTIME("19910212") 02/12/92	Converts a string argument representing a point in time to a time stamp based on the default format string specified by the SET TIME CONVERSION FORMAT TO command.
DATE()	DATE() "1992/07/09" SUBSTR (DATE(), 6, 5) "07/09"	Returns a string which contains a representation of the current date. The format is based on the SET DATE FORMAT TO command.
DATE(y, m, d)	DATE(1993, 5, 2) May 2nd, 1993	<p>When the DATE function is supplied 3 numeric arguments, it uses these arguments to return a timestamp value which is used in Version 3 to represent both a date and a point in time.</p> <p>The three arguments represent the year, month and day of the month specifications respectively.</p> <p>The decimal places that may be present in the arguments are ignored.</p> <p>The year value must be greater than 1970.</p> <p>The month value must be between 1 and 12 inclusive.</p> <p>The day value's limits vary depending on which month and year is specified.</p> <p>The hour, minute and second values of the resulting timestamp are set to zero, i.e. 12am in the morning.</p>
DATESTR(timestamp)	DATESTR(NOW()) "Oct 18, 1993"	This DATESTR function implementation requires a single timestamp its argument. It returns a string representation of the time stamp based on the default format strings specified by SET DATE

Function	Example	Comment
		FORMAT TO command.
DATESTR (tstamp, format)	DATESTR(NOW(), "MMM DD, CCYY") "Oct 18, 1993"	The DATESTR function requires a time stamp as its first argument and then a formatting string as the second argument. It returns a string representation of the timestamp based on the format strings. The available format string structures are listed in the table "Time Stamp and Duration Format String Options." It is actually equivalent to the TIMESTR function with two arguments but has been made available if programmers wish to explicit differentiate between dates and times.
DAY() DAY(tstamp)	DAY() 22	If no argument is specified this function returns the current day of the month value (1-31). If a timestamp argument is specified then it returns the day of the month component of the timestamp.
DAYS(m) DAYS(duration)	DAYS(12) Duration of 12 days	If the DAYS function is supplied a single numeric argument, it will return a time duration based on the number of days specified by the argument. The argument may include a fractional value. This is converted to the appropriate time units and rounded to the nearest millisecond. If the argument supplied is a time duration value then the function returns a numerical value which is the number of days in the time duration argument.
DMY(tstamp)	DMY(NOW()) "18/10/93"	Returns a string representation of the timestamp specified as an argument in "DD/MM/YY" format.
DOW(tstamp) DOW()	DOW(DATE(1993, 9, 18)) 1 = Sunday 7 = Saturday	Returns a numeric representation of the day of the week of the given time stamp value provided as an argument. If no argument is supplied then the current day of the week number is returned.
DOY(tstamp)	DOY(DATE(1993, 8, 3)) 215	Returns a numeric representation of

Function	Example	Comment
DOY()		the day of the year of the given time stamp value provided as an argument. If no argument is supplied then the current day of the year number is returned. The first day of the year is day 1.
DURATION(h,m,s)	<pre>DURATION(10 , 32 , 53.231) 10:32:53.231</pre>	<p>When the DURATION function is supplied 3 numeric arguments, it uses these arguments to return a time duration value. The three arguments represent the hours, minutes and seconds specifications respectively. The decimal places that may be present in the hours and minutes specifications are ignored while in the seconds specification they are used for millisecond specification.</p> <p>The minutes values must be limited to between 0 and 59 inclusive.</p> <p>The seconds value must be limited to between 0.000 and 59.999.</p> <p>The hours value must simply be a positive number.</p>
DURATIONSTR(d1)	<pre>DURATIONSTR(HOURS(3)) " 3 Hours "</pre>	This DURATIONSTR function implementation requires a time duration as its argument. It returns a string representation of the time duration based on the default format strings specified by the SET DURATION FORMAT TO commands.
URATIONSTR (duration, format)	<pre>DURATIONSTR(NOW() - LABDATA.TM, "HH:mm:ss") "120:56:02"</pre>	The DURATIONSTR function requires a time duration as its first argument and then a formatting string as the second argument. It returns a string representation of the time stamp or time duration based on the format strings. The available format string structures are listed in the table: "Time Stamp and Duration Format String Options" on page106.
HOUR() HOUR(tstamp)	<pre>HOUR() 9</pre>	If no argument is specified then this function returns the current hour value (1-24). If a timestamp

Function	Example	Comment
		argument is specified then it returns the hour component of the timestamp.
HOURS(h) HOURS(duration)	HOURS (15) 15:00:00.000 HOURS (3.5) 03:30:00.000	If the HOURS function is supplied a single numeric argument, it will return a time duration based on the number of hours specified by the argument. The argument may include a fractional value. This is converted to the appropriate time units and rounded to the nearest millisecond. If the argument supplied is a time duration then the return value is numeric and indicates the number of hours in the time duration argument.
MDY(timestamp)	MDY (NOW ()) "10/18/93"	Returns a string representation of the timestamp specified as an argument in "MM/DD/YY" format.
MINUTE() MINUTE(timestamp)	MINUTE () 34	If no argument is specified this function returns the current minute value (0-59). If a timestamp argument is specified then it returns the minute component of the timestamp.
MINUTES(m) MINUTES(duration)	MINUTES (12.25) 00:12:15.000	If the MINUTES function is supplied a single numeric argument, it will return a time duration based on the number of minutes specified by the argument. The argument may include a fractional value. This is converted to the appropriate time units and rounded to the nearest millisecond. If the function is supplied with a time duration value as an argument then a numeric value is returned which is the number of minutes in the time duration argument.
MONTH() MONTH(timestamp)	MONTH () 3	If no argument is specified this function returns the current month number (1-12). If a timestamp argument is specified then it returns the month component of the timestamp.
MONTHS(m)	MONTHS (3)	If the MONTHS function is

Function	Example	Comment
MONTHS(duration)	Duration of 3 months	<p>supplied a single numeric argument, it will return a time duration based on the number of months specified by the argument.</p> <p>If the function is supplied with a time duration value as an argument then a numeric value is returned which is the number of months in the time duration argument.</p>
NOW()	NOW() October 23rd, 1993	Returns a timestamp value containing the current time down to a millisecond resolution. Note: that the accuracy of the time measurement is typically only accurate to 1/60th or 1/50th of a second if taken from the local system clock.
SECOND() SECOND(tstamp)	SECOND() 44	If no argument is specified this function returns the current second value (0-59). If a timestamp argument is specified then it returns the seconds and milliseconds component of the timestamp. The milliseconds are returned as the decimal part of the seconds value.
SECONDS(m) SECONDS(duration)	SECONDS(1.253) 00:00:01.253	<p>If the SECONDS function is supplied a single numeric argument. it will return a time duration based on the number of minutes specified by the argument. The argument may include a fractional value. This is rounded to the nearest millisecond.</p> <p>If the function is supplied with a time duration value as an argument then a numeric value is returned which is the number of seconds in the time duration argument.</p>
TIME()	TIME() "09:34:44" SUBSTR(TIME(), 1, 5) "09:34"	Returns a string which contains a representation of the current time. The format is based on the SET TIME FORMAT TO command.
TIMESTR(tstamp)	TIMESTR(NOW()) "Oct 18, 1993"	This TIMESTR function implementation requires a single time stamp its argument. It returns a string representation of the time stamp based on the default format

Function	Example	Comment
		strings specified by SET TIME FORMAT TO command.
TIMESTR (tstamp, format)	TIMESTR(NOW(), "MMM DD, CCYY") "Oct 18, 1993"	The TIMESTR function requires a time stamp as its first argument and then a formatting string as the second argument. It returns a string representation of the time stamp based on the format strings. The available format string structures are listed in the Table Time Stamp and Duration Format String Options.
WOY(timestamp) WOY()	WOY(DATE(1993,10,26)) 43	Returns the week number (in the year) of the timestamp passed as an argument. If no argument is supplied then the functions returns the current week number within the year. Note: the first week of the year is week 1.
YEAR() YEAR(timestamp)	YEAR() 1993	If no argument is specified this function returns the current year. If a timestamp argument is specified then it returns the year component of the timestamp.
YEARS(h) YEARS(duration)	YEARS(5) Duration of 5 years	If the YEARS function is supplied a single numeric argument, it will return a time duration based on the number of years specified by the argument. If the function is supplied with a time duration value as an argument then a numeric value is returned which is the number of years in the time duration argument.

Date and Time Operators

Table 75: Date and Time Operators

Type	Symbol	Precedence	Comments and Example
Divide duration/number duration1/duration2	/	6	The time/date version of the divide operator must either have a divisor which is a time duration and a denominator which is a number or have both divisor and denominator being time durations. The result of the operation is a time duration in the former case and a number in the latter. Time stamps can not be used with division operators. HOURS(5) / 6 00:50:00 I.e. 50 minutes DAYS(3) / MINUTES(10) 432 I.e. 432 10 minute periods in 3 days.
Multiply duration*number number*duration	*	6	The time/date version of the multiplication operator must multiply a number and a time duration together. Time stamps can not be used with multiplication operators. The result of the operation is a time duration. 3.125 * HOURS(2) 06:15:00 I.e. 6 hours and 15 minutes.
Remainder duration%numberd 1 % d2	%	6	The time/date version of the remainder operator works with similar forms of operands as the division operator does. If the first operator is a duration and the second a number then the result is a time duration. If both operands are time durations then the result is a number. HOURS(2) % 7 00:00:04 I.e. 4 seconds DURATION(2, 13, 12) % MINUTES(6) 00:01:12 I.e. 1 minute and 12 seconds.
Subtract t1 - t2 = d1 t1 - d1 = t2 d1 - d2 = d3	-	5	The time/date version of the subtraction works with both time stamps and time durations. Most combinations are allowed except for a request to subtract a time stamp from a duration. The result of time/date subtractions is a time duration when both operands are time stamps or when both operands are time durations. The result is a time stamp if the first operand is a time stamp and the second a duration. NOW() - DURATION(8,0,0) 02:16:19

Type	Symbol	Precedence	Comments and Example
			<p><code>TIME(2,9,15) - HOURS(8)</code> <code>18:09:15</code> I.e. 6:09:15pm the previous day.</p> <p><code>HOURS(8) - MINUTES(45) - SECONDS(12)</code> <code>07:14:48</code></p>
<p>Add $t1 + d1 = t2$ $d1 + t1 = t2$ $d1 + d2 = d3$</p>	+	5	<p>The time/date version of the addition operator works with both time stamps and time durations. Most combinations are allowed except for adding two time stamps. The result of time/date addition is a time stamp in all cases except when both operands are time durations.</p> <p><code>NOW() + DURATION(8,0,0)</code> <code>22:16:19</code> I.e. 10:16:19pm the same day.</p> <p><code>TIME(2,9,15) + HOURS(8)</code> <code>10:09:15</code> I.e. 10:09:15pm the same day.</p> <p><code>HOURS(8) + MINUTES(45) + SECONDS(12)</code> <code>08:15:12</code></p>
<p>Equal to $t1 = t2$ $d1 = d2$</p>	=	4	<p>The time/date version of the equality operator works with both time stamps and time duration but the type of each operand must be the same. The equality operator returns a numeric value. I.e. 0 for false, non zero for true.</p>
Not equal to	<code><></code> or <code>#</code>	4	See equality operator comment.
Less Than	<code><</code>	4	See equality operator comment.
Greater Than	<code>></code>	4	See equality operator comment.
Less Than or Equal To	<code><=</code>	4	See equality operator comment.
Greater Than or Equal to	<code>>=</code>	4	See equality operator comment.

Date and Time Related Set Actions

Table 76: Date and Time Related Set Actions

Action Statement	Description
<code>SET DATE FORMAT TO <expr>;</code>	The display format of any date values used in the system can be modified from within the meta script. The expression reference must result in a string that contains combinations of the specification string listed in the next table.
<code>SET TIME FORMAT TO <expr>;</code>	The display format of any time values used in the system can be modified from within the meta script. The expression reference must result in a string that contains combinations of the specification string listed in the next table.
<code>SET DURATION FORMAT TO <expr>;</code>	The display format of any time values used in the system can be modified from within the meta script. The expression reference must result in a string that contains combinations of the specification string listed in the next table.

Time Stamp and Duration Format String Options

Table 77: Time Stamp and Duration Format String Options

Format Substring	Description	Valid for Timestamps	Valid for Duration's
CC	Century number	Yes	No
YY	Year number within the century padded with zeros if necessary to take up 2 character positions.	Yes	Yes
%Y	The number of years in the time stamp or duration. Note that a suffix of " year(s)" is always included.	No	Yes
MMMM	Whenever four M's are grouped together, it indicates that the full month name is to be used e.g. "January", "February", ... "December".	Yes	No
MMM	Indicates that the 3 character abbreviation for month name is to be used e.g. "Jan", "Feb", ... "Dec".	Yes	No
MM	The month number padded with zeros to take up 2 character positions.	Yes	Yes
%M	The month number. Note a suffix of "month(s)" is always included.	No	Yes
DDDD	Whenever four D's are grouped together, it indicates that the full day of the week name is to be used e.g. "Monday", "Tuesday" etc.	Yes	No
DDD	Indicates that the 3 character abbreviation for day of the week	Yes	No

Format Substring	Description	Valid for Timestamps	Valid for Duration's
	name is to be used e.g. "Mon", "Tue" etc.		
Ddd	Indicates that the day of the month number is to be used but suffixed with the appropriate English suffix for that number e.g. 1st, 2nd, 3rd, 4th etc.	Yes	No
DD	The day of the month number padded with zeros to take up 2 character positions.	Yes	Yes
%D	The day of the month number. Note a suffix of "day(s)" is always included.	No	Yes
HH	24 hour clock hour indication padded with zeros to take up two character positions. If more than 99 hours are relevant then more character positions are used up. The situation of having more than 24 hours can happen with duration's.	Yes	Yes
hh	12 hour clock hour indication padded with zeros to take up two character positions.	Yes	No
%H	The hour number. Note that a suffix of "hour(s)" is always included.	No	Yes
mm	The minute indication padded with zeros to take up two character positions.	Yes	Yes
%m	The minute number. Note that a suffix of "minute(s)" is always included.	No	Yes
SS	The second indication (without milliseconds) padded with zeros to take up two character positions.	Yes	Yes
%S	The seconds indication. Note that a suffix of "second(s)" is always included.	Yes	Yes
iii	The number of milliseconds indication with zero padding to take up 3 character positions.	Yes	Yes
ii	The number of hundredths of a second with zero padding to take up 2 character positions. Note that the millisecond value is rounded off to the nearest hundredth of a second.	Yes	Yes
i	The number of tenths of a second. Note that the millisecond value is rounded off to the nearest tenth of a second.	Yes	Yes
%i	The number of milliseconds. Note: that a suffix of "millisecond(s)" is always included.	No	Yes
Am AM Pm PM	The am or pm indication.	Yes	No