Table of Contents

Overview of SCADA Basic

See Also

User programs can be written in a proprietary language known as SCADA Basic. This is a block structured, interpreted language with a syntax closely related to industry standard Basic.

A project may be configured to run programs on various occasions:

- At start-up of the Supervisor.
- On change of a variable in the database.
- Cyclically.
- From the keyboard.
- From control zones on the screen.
- When called from other programs.

Overview of this Help

See Also

Table of contents

In the Language Reference section of table of this Help's contents, the instructions are arranged into categories (by type of instruction) and alphabetically (as individual topics, e.g. ACOS)

Navigating and searching the Help

You can also locate topics in these ways:

- See Also: click on a 'See Also' link in a topic to open a related topic.
- Index: double-click on a keyword or phrase in the Index, or select then click on the Display button.
- Search: type a word or phrase into the Search tab and select List Topics, then double-click on a line or select a line and click on Display.

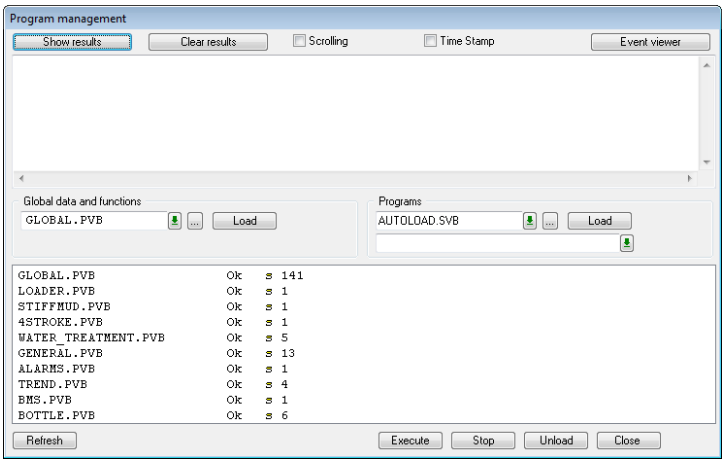
Conceptual topics

Program Management

See Also

Program files may be created, debugged and tested online using the facilities provided in the Program Management dialog box. This is displayed either from the menu Configure.Actions.Programs or by selecting key F9. Programs are stored as text files in folder P of a project.

The Program Management dialog has several distinct areas. Show picture



The Results Area

The results area is at the top of the dialog box and displays messages from programs (using the PRINT instruction), program trace messages and syntax errors. The following buttons are provided.

- Show Results - Display messages.
- Clear Results - Clear the messages.
- Event Viewer - Display the Loading (Trace) window.
- Scrolling - Change the operation of the results window so that errors are always displayed. (Normally, errors are only displayed on request.)

The Global Function Area and Program Area

These two areas are similar in appearance and operation, except that one deals with the global declaration program and the other deals with all other programs. The following options are available.

- A program may be selected by typing its name or displaying it from a list box containing a list of existing programs. A new program may be created by typing in a new name, and then using the edit facility to create and save it.
- The Edit button displays the file for the selected program in the editor window, from which it may be viewed, edited and saved.
- Load: the selected program is loaded ready to be run. If an existing program of the same name is running it will be stopped and the new program will be loaded (but not run). Any syntax errors are reported at this stage in the results window (see Show Results and Scrolling above).

The global program must always be loaded before loading any other programs.

See the topic The Global Program for important information about the operation and behaviour of the global program.

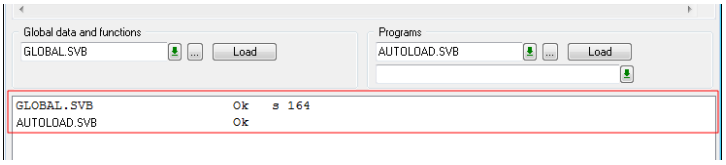
The Program Status Area

The lower area of the dialog box provides a list of programs that are currently loaded. By clicking on a program status line and using the control buttons beneath it, programs may be run, stopped and unloaded.

The global declaration program may only be loaded; it cannot be run.

Example

The status line for the program AutoLoad might show the following information. Show picture

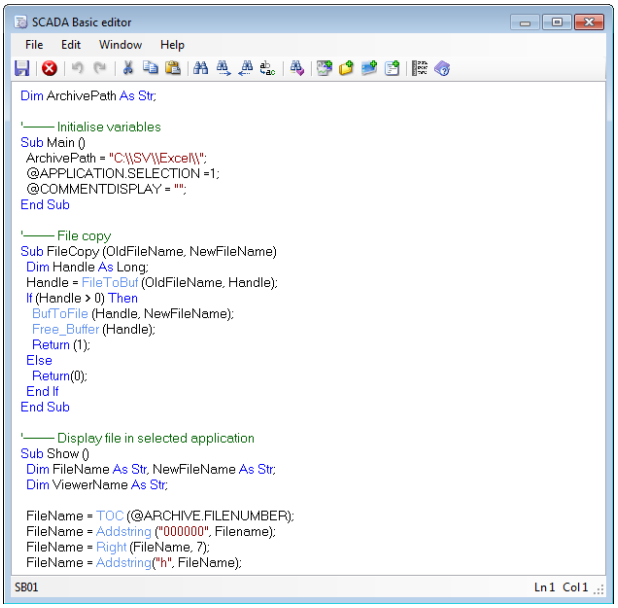


Item	Meaning
AUTOLOAD.SVB	The name of the program.
OK	The program status. OK means that the program is healthy. NOK means that there is an error with the program.
Sleep (10)	The current instruction being executed. In practice, as the status is only updated when the program ends or reaches a DELAY instruction this will always display Sleep or be empty.
l 0010	The line number currently being executed - in this example 10 (Note that the first character is a lower-case 'L')
s 5	The number of symbols (subroutines) declared in the program.

The editor window

Overview of the Editor Window

The editor window is used to view, edit and save SCADA Basic programs. Show picture



Main features

- Standard Windows keystrokes and tools for cut, copy, paste etc. See the topic The editor toolbar
- Colour coded text to aid program development.
  - Green - comments
  - Blue - keywords
  - Dark red - strings
  - Bright red - reserved words
- Keyword and parameter auto completion. See the topic Using keyword and parameter auto completion
- Special insert facilities (variable name etc.) See the topic Special insert facilities
- Navigation to line number or function. See the topic Navigating the program file
- Status line including the name of the program being edited and the current cursor position (line and column number).
- Context sensitive help. Pressing F1 in the editor window will open the SCADA Basic help. If the cursor is on a recognised keyword, the help will go directly to the relevant topic.

Editor options

The font type and size used by the editor, plus the tab size, can be changed from the General tab of the Options dialog.

- Open the Windows.Options command on the menu to open the Options dialog.

The Editor Toolbar

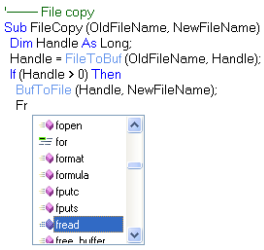
Icon	Keyboard	Function
	Ctrl+S	Save the program file.
	Ctrl+X	Close the editor. If you have made any changes you are prompted to save them.
	Ctrl+Z	Undo the previous edit. The undo command works an unlimited number of times.
	Ctrl+Y	Redo the previous undo. The redo command works an unlimited number of times.
	Ctrl+C	Copy the selection to the clipboard.
	Ctrl+V	Copy the contents of the clipboard to the file at the cursor position.
	Ctrl+F	Open the Find dialog.
	F3	Find the next occurrence of the string entered in the Find dialog.
	Ctrl+F3	Find the previous occurrence of the string entered in the Find dialog.
	Ctrl+H	Open the Find and Replace dialog.
	Ctrl+Shift+F	Open the Goto Function dialog.
	Ctrl+I, Ctrl+V	Open the Variable Selector from where a variable name may be selected and inserted.
	Ctrl+I, Ctrl+B	Open the Branch Selector from where a branch name may be selected and inserted.
	Ctrl+I, Ctrl+M	Open the Mimic Selector from where a mimic name may be selected and inserted.
	Ctrl+I, Ctrl+P	Open the Program Selector from where a program name may be selected and inserted.
	F11	Display line numbers in a column on the left side of the editor dialog.
		Open the SCADA Basic Help file.

Using Keyword and Parameter Auto Completion

The SCADA Basic edit window includes both keyword and parameter auto completion.

Keyword completion

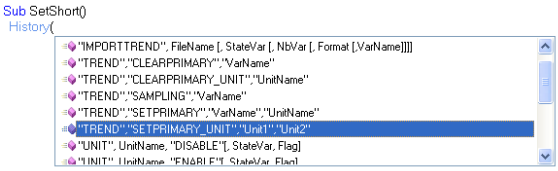
Pressing the Ctrl+Spacebar keys simultaneously displays a pop-up window containing a list of all keywords. If you type one or more characters on a line before using Ctrl+ Spacebar, the list will automatically scroll to the first relevant keyword. For example if you type Fr and then Ctrl+Spacebar the list will scroll to FREAD. Show picture



The list can be scrolled using the arrow keys and the required keyword selected using either the Enter or Tab keys or by clicking with the mouse.

Parameter completion

On typing an opening bracket, the editor checks the text preceding the bracket. If the text is identified as a keyword the Parameter Completion pop-up opens displaying a list of options for that keyword. Show picture



The list can be scrolled using the arrow keys and the required parameters selected using either the Enter or Tab keys or by clicking with the mouse.

Character casing

By default auto-completed text is inserted as it appears in the keyword or parameter lists. Using the options in the Completion tab of the Options dialog you can force the text to upper or lowers case. The Options dialog is displayed from the Window.Options command on the menu.

Special Insert Capabilities

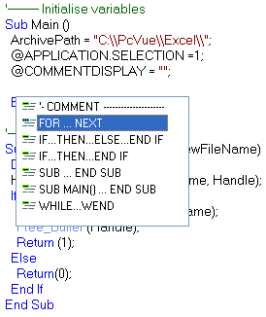
The editor contains a number of special insert commands specific to the SCADA Basic environment.

Inserting the name of a SCADA object

Object	Keyboard	Comments
Variable	Ctrl+I+V	Displays the variable selector from where the variable name can be selected.
Branch	Ctrl+I+B	Displays the branch selector from where the branch can be selected.
Mimic	Ctrl+I+M	Displays a list of mimics available to the project from where a mimic name can be selected.
Program	Ctrl+I+P	Displays a list of programs available to the project from where a program name can be selected.

Inserting a code snippet

A code snippet is a small section of pre-defined program code. A list of the available code snippets is displayed using the keystrokes Ctrl+I+X. Show picture



- Scroll the list using the arrow keys and the required snippet inserted using either the Enter or Tab keys or by clicking with the mouse.

Commenting out a section of code

When debugging a program it is often useful to 'comment out' a section of program code. That is, to convert the code into comments by preceding each line with a single speech mark (comment) character.

- To 'comment out' a section of code select the lines and then use the Ctrl+I+C keystrokes.
- To return the lines back to code, select them again and use the Ctrl+I+U keystrokes.

Navigating the Program File

Going directly to a line using the line number

Programming errors are generally reported with a line number and when debugging it is useful to be able to go directly to that line.

- You can open a dialog in which you can enter the line number using either the keystroke Ctrl+G or by double clicking on the line number in the status bar.

To display line numbers in the editor, use the keystroke F11 or the Line Numbers tool on the toolbar.

Going directly to a function

- You can navigate directly to a specific function using the Goto Function dialog opened using the keystroke Ctrl+Shift+F.

The dialog contains a list of functions within the program being edited and clicking on the function name causes the editor to scroll directly to that function.

Searching and replacing

The editor supports standard Windows search and replace functions activated either using the keystrokes Ctrl+F and Ctrl+H or the toolbar. The search string combo box remembers up to 20 previous search strings.

Loading and Verifying Programs

SCADA BASIC programs differ from normal BASIC in that they must be loaded before they can be run. At load time the program is pre-compiled. The syntax of the instructions is verified and the program is converted into a series of tokens in which each of the instructions is represented by a 1 byte code. At that point, references to variables are not verified, nor is the existence of called functions, as they may be loaded later.

When the program is executed, variable names are calculated (using a branch if specified), and their existence is verified. The same applies to any functions; they must be loaded before execution.

Where speed of execution is of primary importance, it is recommended that CPU intensive functions are written in a compiled language and stored in a dynamic link library (DLL). The functions may then be called from a SCADA BASIC program.

Creating a user program

If you have not created a global program, or opened a project that already contains one, you must first create one as described in the Program Management topic.

To create and load a user program:

1. If it is not already open, press function key F9 on the keyboard to open the Program Management window of SCADA BASIC.
2. Type a program name (e.g. 'user1') into the Programs box.
3. Click on the editing button to the right of it ('...') to open the Editor window (which is described in the next topic).
4. Type the code for one user SUB into it, e.g.:

```
SUB user1 ( )
'declarations...
'instructions...
END SUB
```

5. Click on OK to close the Editor window and return to the Program Manager window.
6. Click on the Load button in the Programs section. Its name will appear in the Program area to show that it is loaded.

Some Programming Guidelines

As SCADA BASIC must share the processing capacity with the many other tasks the computer performs, it is worth thinking carefully about the design of programs to be written so as to avoid excessive CPU loading and loss of system performance. The following guidelines are provided to assist the design process:

- As soon as significant processing is required, for example in a complex calculation, it is preferable to call a dynamic link library (DLL) since it is compiled and hence much faster in execution.
- DLL routines must be developed so they require the minimum amount of time to execute. While a DLL is being executed all operations in the Supervisor are suspended.
- Independent programs such as a spreadsheet or database (EXCEL, ACCESS) may be executed without suspending the Supervisor, using DDE for data transfer.
- Use the TREE instruction for writing generic programs that may be re-used with variables of a different branch.
- As a general rule CYCLIC programs should be used with moderation and EVENT driven programs should be used in preference to them.

Use the DELAY instruction with extreme care. The program becomes suspended for the DELAY period. No other function can be executed in the program in that period.

The use of DELAY within a function that is called cyclically may produce unpredictable results and should be avoided.

- Assignment of a value to a variable only takes place after return of control, that is, when a DELAY is encountered or the program ends.
- When a SCADA BASIC program takes control, it only relinquishes it voluntarily. It is therefore essential to avoid long processing loops (such as WHILE or FOR), without periodic return of control.
- For calculating the values of variables, consider using expressions.

How to Run a SCADA Basic Program

SCADA Basic programs may be started in a variety of ways depending on the project's requirements. Below is a comprehensive list.

- From an event action. Triggered by a change in value of a variable. See the help book The Application Explorer.Actions.Event actions for information on how to configure an event action.
- From a cyclic action. Periodically every N seconds. See the help book The Application Explorer.Actions.Cyclic actions for information on how to configure a cyclic action.
- From the Scheduler. According to a schedule (For example, on the 1st day of each month at 12 PM.) See the help book The Application Explorer.Actions.Scheduled actions for information on how to configure a schedule.
- When the user presses a key combination. See the help book The Application Explorer.Actions.Function keys for information on how to configure function keys.
- When the Supervisor starts. The Supervisor's startup options can be configured to run programs before and / or after communications is started. See the help topic The Application Explorer.Project settings.Station startup for more information on how to configure the program to be run at startup.
- By the User from a control zone on a mimic. See the topic Developing the HMI.Animation.Run.Running a SCADA Basic Program.
- When a User logs on and/or off. See the book The Application Explorer.User Accounts.Configuring profiles.
- When a mimic opens or closes. See the topic Developing the HMI.Working with mimics.Window properties dialog.Programs for more information.
- From an Alarm Viewer using the associated action of an alarm variable. See the topic The Application Explorer.Variables.The variable tree.Configuring variables.Associating behavior with a variable.Configuring an associated action.
- From another program.

You can run up to five lines of SCADA Basic using the Macro animation without having to write an actual program. See the topic Developing the HMI.Animation.Run.Starting a SCADA Basic Program.

1.
2.
3.
4.

Running a SCADA Basic Program From a Mimic Displayed in a WebVue Client

You can run a SCADA Basic program from a mimic displayed in WebVue, in the same way as from a mimic displayed in the Supervisor, using the animation Run-Program. However there are some differences in behavior and limitations.

- The program runs on the Supervisor that is the WebVue backend.
- Instructions which interact with mimic elements, for example CHECKLIST, do so on the mimic displayed in Webvue.
- For all other instructions any action produced by the instruction takes place on the Supervisor that is the WebVue back end. This is very important to take into account when the program is interacting with external resources such as files.
- Most instructions are fully supported when a program is ran from WebVue. However there are some instructions not supported and others that only some modes are supported. WebVue support information is supplied in the topic for each instruction.

Modes marked as Do not use must not be used in WebVue as they may produce unexpected results.

Program conventions

Program Structure

See Also

Each program is composed of an optional declaration part and a number of functions (subroutines). The declaration part must always appear before the functions and is used to define any external functions (DLL) that are to be called by the program.

The program must have at least one function, called MAIN, which is the starting point for execution (except for the global declaration program - see The Global Program).

The structure of a program is as follows.

The items identified in square brackets are optional. The square brackets are not part of the SCADA Basic code.

```
[external declaration 1];
[external declaration 2];
. . .
[external declaration n];

SUB main ()
[declaration 1];
. . .
[declaration N];
[instruction 1];
. . .
[instruction N];
END SUB

[SUB proc1 ([parameters])]
[END SUB]
[SUB proc2 ([parameters])]
[END SUB]

. . .

[SUB procN ([parameters])]
. . .
[END SUB]
[             'This is a comment.]
```

Any text to the right of an apostrophe is treated as a comment.

Functions

See Also

A function is composed of an optional declarations part and a sequence of instructions. The declaration part always comes before the instruction sequence.

Execution

A function may receive up to 10 parameters on entry, with the exception of the main function which receives none. Parameters are passed by value not reference, i.e. a parameter's value is passed, not its name.

The structure of a function is as follows :

```
SUB proc name ([parameter list])
  [declaration 1];
  [declaration 2];
  .
  .
  [declaration N];
  [instruction 1];
  [instruction 2];
  .
  .
  [instruction N];
END SUB
```

Notes

Instructions and declarations always end with a semicolon.

In some of the examples of code in this Help, an instruction is split across lines. The symbol ↵ is used to indicate where that occurs. This is only due to formatting restrictions in this manual. In a real program a complete instruction must be contained on a single line.

Each program runs in a single thread. This means that two functions from the same program cannot be executed at the same time. Thus two functions launched with EVENT, CYCLIC, CRONTAB, KEY or SELECTOR cannot run at the same time.

To avoid this situation, the DELAY instruction must be used with extreme caution in event triggered functions.

Asynchronous calling

The effects of a function call do not occur in a fixed sequence (see the example below).

Example

In this example, program P1 calls program P2 with one argument.

Program P1 :

```
SUB main()
    PROGRAM("FUNCTION","P2","", "message1");
    PRINT("message 2");
END SUB
```

Program P2 :

```
SUB message1()
    PRINT("message1");
END SUB
```

When program P1 is run, it may display:

```
    message1
    message2
```

or:

```
    message2
    message1
```

The Global Program

See Also

The global declaration program is used to declare working variables and functions that are used throughout a project. It does not directly run functions itself. The global program can have any name and its source file is stored in the same folder (P) as the other programs.

An example of a simple global program follows.

```
' Declare global working variables
DIM WatchDog AS SINGLE;
DIM RunTime AS SINGLE;
DIM WindowName AS STR

' Global functions

SUB setdog ( )
  watchdog = 1;
END SUB

SUB cleardog ( )
  watchdog = 0;
END SUB
```

Essential features to be understood about the global program

- **The global program must always be loaded before running any other programs. If you do not load a global program the first other program that is loaded will be treated as the global program with corresponding differences in behaviour.**
- Only one global declaration program may be loaded at a time.
- A global program does not have a MAIN function and can only be loaded. It cannot be run.
- Any working variable that has been declared in a global program is known to all other programs and is referenced by name only. It must not be declared again in any other program.
- Any function that has been declared in the global program is known to all other programs and is referenced by name only. It must not be declared again in any other program.
- The global program may be (re)loaded at any time from the Program Management dialog box. However re-loading the global program will cause all other programs to stop and be un-loaded.
- To load a global program automatically on project startup, add it to the Station Start-up configuration (Application Explorer.Settings.Stations Startup).

Variable declaration

Variables Tree

See Also

All variables are available to all programs.

If a variable is to be changed from within a program then it must have the Control attribute set.

If a variable is to be substituted, the declaration must be made inside a Sub function.

It must not be not made globally in the program header nor passed as an argument from a Run Program animation. See example in the DIM instruction's topic.

Variables may be referenced directly using their full name, or by using the branch instruction and then using a name relative to the branch. A character string may also be used as part or all of a variable name.

When referring to variables within a program, you must ensure that your program manages the variable type correctly as in the table below.

Variable	Type in program	Range
Bit	INTEGER	0 or 1.
Register	SINGLE	-3.37E+38 to +3.37E+38.
String	STR	Up to 32,000 characters.

Using a Branch

See the topic Using Programs With a Branch.

Using Deferred Naming

Normally when referring to variables in a program, you hard code the variable name as part of the program. Sometimes it is better to construct the name of the variable in the program itself, for example if you want to pass the name of the variable to the program as an argument. When using this technique the variable name is stored in another (string) variable.

If deferred naming is being used, the variable that contains the name of the variable must be declared inside the SUB function where it is used. See the examples below.

To refer to the variable you use the name of the string preceded by a question mark "?". This is best illustrated by an example.

```
SUB Init ()
  DIM VarName As Str;
  DIM Index as Integer;
  For Index = 0; Index < 11; Index ++
    VarName = AddString("Tank", TOC(Index));
Varname = Addstring(Varname, ".Level.SP");
?Varname = 0;
Delay(0.1);
Next Index
END SUB
```

You can also use deferred addressing to refer to only part of a variable name (although it must be a complete branch). The previous example could be re-written as follows..

```
SUB Init ()
DIM VarName As Str;
DIM Index as Integer;
For Index = 0; Index < 11; Index ++
VarName = AddString("Tank", TOC(Index));
?Varname.Level.SP = 0;
Delay(0.1);
```

```
Next Index
END SUB
```

Temporary Variables

See Also

Temporary variables may be created by using the instruction TEMPORARY_DB. They are visible to both the program language and the HMI. Temporary variables use far less memory than conventional variables. They are ideal for use in large projects where a program is calculating a variable that will only be used for purposes of animation.

For more information on temporary variables see the book on The Variables Tree and the topic on the instruction TEMPORARY_DB.

Working Variables

See Also

Working variables are for internal use within a program. They are not included in the variables tree.

Types of working variables

Type	Description	Range
Integer	2 Byte integer	-32, 768 to +32, 767
Long	4 Byte integer	-2, 147, 483, 648 to +2, 147, 483, 647
Longlong	8 Byte integer	-9, 223 ,372 ,036 ,854 ,775 ,808 to 9, 223, 372, 036, 854, 775 ,807
Single	4 Byte real	1.175494 E–38 to 3.402823 E+38 for positive values
Double	8 Byte real	2.225074 E–308 to 1.79769313 E+308 for positive values
Str	Character string	2047 characters
Const	Numeric constant	2.225074 E–308 to 1.79769313 E+308 for positive values

Working variables cannot be used for triggering actions on event.

Changing the maximum length of a character string

The default maximum length of a character string is 2047. This can be changed by adding the following line to the global program.

```
SYS MAXARGSTRINGS=nnnn
```

Where nnnn is the maximum number of characters in the range 2047 to 8191.

Examples

```
Sub func1 ()
'--------local variable
  Dim cMystring as STR;
  cMmystring = Addstring("", "…..."); 'up to 32000 chars

'--------variable in variables tree
  @textvar = mystring; 'up to 32000 chars
End sub
```

```
Sub func2 ()
'--------64-bit variable
  Dim LL1 as longlong;
  Dim LL2 as longlong;
  Dim LLRes as longlong;
  LL1 = 2147491969;
  Print(LL1); '2147491969
  LL2 = 1;
  LLRes = logical64("AND", LL1, LL2);
  Print(LLRes); '1
End Sub
```

Variable scope

A variable's visibility to the program environment is determined by where it is declared. The behaviour from version 11.2 onwards is strictly enforced and is as follows.

Declared in	Visible to
Within the global program	All other programs and functions.
Outside of the functions at the head of a program	All functions in that program and cannot be re-declared in any function of that program.

Within a function	Only in that function.

Prior to version 11.2 the scope was not so strictly enforced and the following exceptions were permitted.

- Any variable declared in a function was also visible to any functions that it called. For example:

```
Sub Func1()
  Dim I as Integer;
  Func2();
End Sub

Sub Func2()
  I = 99;
  Print(I);
End Sub
```

- A variable declared at the head of the program could be re-declared in any function. For example:

```
Dim cString as Str;

Sub Main ()
  Dim cString as Str;
  cString = "Hello";
End Sub
```

Variable scope backwards compatibility

Backwards compatibility of variable scope is controlled by the Working variables scope control setting in the Program Settings dialog (Application Explorer).

Language reference

Operators

Overview of Operators

There are two kinds of operators: arithmetic and logical. These are described in the next two topics.

All of the operators may only be used on numeric variables or expressions (Integer, Long, Single, Double and Const). The variables or expressions must be of the same type.

Logical Operators

The logical operators are:

Symbol	Operation
!=	Inequality
==	Equality
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!	Negation
||	Logical Or
&&	Logical And

Example 1

```
Const Buffer_Size = 500;
DIM I1 As Integer, I2 As Integer;
I1 = I1+ 5; 'Correct
I1 = I1+ I2; 'Correct
I1 = I2+ Buffer_Size; 'Correct
I1 = 5+ I1; 'Incorrect
```

Example 2

```
DIM Str1 As Str, Str2 AS STR;
Str1 = "I Am A String" 'Correct
Str2 = Str1+Str2; 'Incorrect
```

String concatenations and comparisons may only be done using specialised functions.

Logical operations on Registers are made using the LOGICAL instruction.

Arithmetic Operators

See Also

The arithmetic operators are:

Symbol	Operation
=	Assignment
+	Addition
-	Subtraction
*	Multiplication
/	Division
++	Increment By 1
--	Decrement By 1

Constants may be combined with any numeric type if they are placed to the right in the expression.

Example

```
Const Buffer_Size = 500;
DIM I1 As Integer, I2 As Integer;
I1 = I1+ 5; 'Correct
I1 = I1+ I2; 'Correct
I1 = I2+ Buffer_Size; 'Correct
I1 = 5+ I1; 'Incorrect
```

Example

```
DIM Str1 As Str, Str2 AS STR;
Str1 = "I Am A String" 'Correct
Str2 = Str1+Str2; 'Incorrect
```

Only the assignment operator is allowed on character strings.

Instructions by category

Overview of the Categories of Instructions

See Also

This topic explains how to access topics through categories of related instructions plus 'See Also' links for themes that associate the instructions, concepts and reference material.

The instructions are specified in the next book, 'Instruction reference A to Z' with one topic per instruction.

Some instructions have multiple syntaxes and modes of operation. The final section below describes how to access Help on the syntaxes and modes of such an instruction.

Lists of instructions

This Help book provides one-sentence summaries of what the instructions do. The instructions are arranged into these categories:

- Buffer management
- Communication
- Debug
- Declaration
- File management
- Historic data
- HMI
- Input/Output
- Mathematical functions
- Miscellaneous
- Program Execution
- Program Structure
- String manipulation
- Type conversion
- Time and date
- Variables tree
- Window

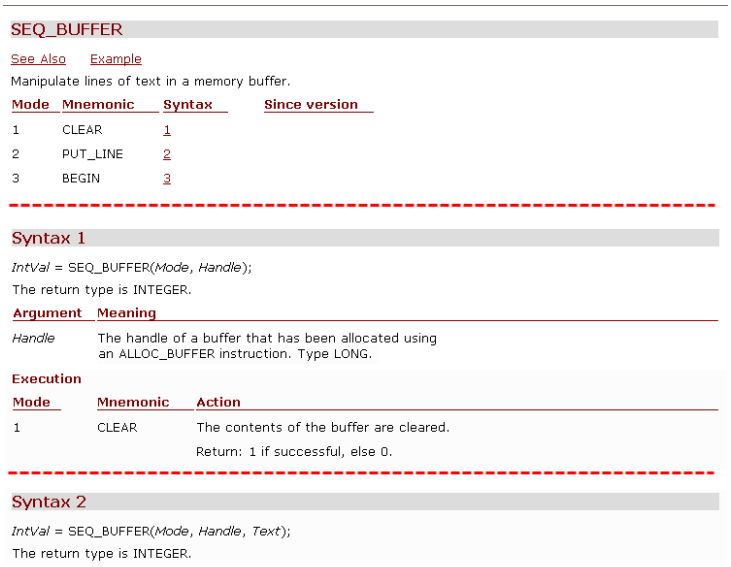
You can use the 'See Also' link at the top of each list to access the topics of instructions in that category.

Links in the instruction topics

In the book 'Instruction Reference A to Z', the individual topic for each instruction also has a 'See Also' link. It provides access through a pop-up window to the index of other instructions in the same category (or several indexes for different categories) and to individual topics that relate closely to its topic.

Syntaxes and modes

An instruction's topic specifies the syntax format(s) for the instruction. If there is more than one syntax to allow for different parameters, the topic has a section for each syntax. Show example



SEQ_BUFFER

See Also Example

Manipulate lines of text in a memory buffer.

Mode	Mnemonic	Syntax	Since version
1	CLEAR	1	
2	PUT_LINE	2	
3	BEGIN	3	

Syntax 1

IntVal = SEQ_BUFFER(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a buffer that has been allocated using an ALLOC_BUFFER instruction. Type LONG.

Execution

Mode	Mnemonic	Action
1	CLEAR	The contents of the buffer are cleared. Return: 1 if successful, else 0.

Syntax 2

IntVal = SEQ_BUFFER(*Mode*, *Handle*, *Text*);

The return type is INTEGER.

An instruction may have several modes with which to achieve different effects. If so, the instruction's topic starts with an index of which syntax to use for each mode. The Syntax column of the index provides links to the syntax sections. The effects of each mode are defined in the syntax section concerned.

With functions that use a MODE argument, the argument can either be an integer value 1,2 ... or

a text string within quotation marks (e.g.: "OPEN", "CLOSE").

Buffer Management Instructions

Instruction	Action
ALLOC_BUFFER	Allocate a memory area of N bytes.
BUFTOFILE	Create a file using the contents of a memory buffer.
BUFTOEXCEL	Create an Excel file in XLSX format using the contents of a memory buffer.
CGET_BUFFER	Recover a character located in a memory area reserved by ALLOC_BUFFER.
COPY_BUFFER	Copy the contents of a buffer to another buffer.
DGET_BUFFER	Recover a DOUBLE located in a memory area reserved by ALLOC_BUFFER.
EXCELTOBUF	Create a memory buffer from an Excel file in XLSX format.
FILETOBUF	Create a memory buffer using the contents of a file.
FREE_BUFFER	Free a memory area reserved by ALLOC_BUFFER.
IGET_BUFFER	Recover an INTEGER located in a memory area reserved by ALLOC_BUFFER.
LGET_BUFFER	Recover a LONG located in a memory area reserved by ALLOC_BUFFER.
PUT_BUFFER	Store data in a memory area reserved by ALLOC_BUFFER.
SEQ_BUFFER	Enter a string into a memory area reserved by ALLOC_BUFFER.
SGET_BUFFER	Recover a SINGLE located in a memory area reserved by ALLOC_BUFFER.

Communication Instructions

These series of instructions allows you to control the behavior of data acquisition drivers. They offer what it takes to start & stop communication, verify communication status, monitor and control various communication parameters as well as change some configuration elements.

The exact list of modes and capabilities vary depending on the data acquisition driver.

Instruction	Action
BACNET	Manages communication via the BACnet driver.
CIMWAY	Stop, start or test the state of communications for most of the polling-based data acquisition drivers.
DDE	Manages DDE exchanges with DDE server applications not already configured in the Supervisor.
DDECONV	Manage DDE exchanges with configured DDE server applications.
FILETRANSFER	Manage transfer of files between the communication object and the Supervisor.
LAN	Manages the Supervisor' client/server communication.
LONWORKS	Manages communication via the LonWorks driver.
M104	Manages communication via the IEC 60870-5-104 client driver.
MDNP3	Manages communication via the DNP3 client driver.
M61850	Manages communication via the IEC 61850 client driver.
OPC	Manage exchanges with OPC servers.
SNMP	Manages communication via the SNMP Manager driver.

Debug Instructions

Instruction	Action
ERROR	Return the error code, program name, function name, or line number of the last error.
PRINT	Display data in the program monitor window.
TRACE	To run a trace in SCADA BASIC without using the PRINT function.
TRACEON / TRACEOFF	Enable tracing of function calls.

Declaration Instructions

See Also

Instruction	Action
CONST	Declare a constant.
DECLARE FUNCTION	Declare an external function.
DECLARE SUB	Declare an external sub-program.
DIM	Declare a variable or array.
REGVAR2D	Control the use of register variables as type DOUBLE in a program.
SUB ... END SUB	Start and end of a subroutine.

File Management Instructions

Instruction	Action
FCLOSE	Close the specified file.
FCOPY	Copy a file from source to destination.
FEOF	Determine whether the end of a file has been reached.
FGETC	Read a character from a file.
FGETS	Read a character string from a file.
FMOVE	Move a file from source to destination.
FOPEN	Open the specified file in accordance with the specified access mode.
FPUTC	Write a character to a file.
FPUTS	Write a character string to a file.
FREAD	Read N data items of the same type from a file and store them in a buffer allocated by ALLOC_BUFFER.
FSEEK	Move the file pointer to a new position.
FSTAT	Enable size and date information on a file to be obtained.
FWRITE	Write N data items of the same type to a file from data stored in a buffer allocated by ALLOC_BUFFER.
RENAME	Rename a file.
UNLINK	Delete a file.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Historic Data Instructions

Instruction	Action
EXPORT	Run a previously configured (Application Explorer) export
EXPORT_LOG	Generate historic log data using the functionality provided by Data Export.
EXPORT_TREND	Generate historic trend data using the functionality provided by Data Export.
HISTORY	Import and export a variable's historic data.
SVALA	Extract alarm information.
SVBATCH	Produce a user configurable batch record.
SVLOG	Extract log information.
SVTREND	Extract historic information for a group of variables.

HMI Instructions

Instruction	Action
ANIMATION	Enable an animation to be run from SCADA BASIC.
ALARMDISPLAY	Reproduce the functions available on the lower border of an Alarm Viewer.
CHART	Manage the operation of an XY Chart.
CHECKLIST	Access properties of the Check-box list form control.
COMBOBOX	Access properties of the Combo-box form control.
LISTBOX	Access properties of the List-box form control.
LOGDISPLAY	Simulate the control buttons of a Log Viewer.
MAPDISPLAY	Manage the operation of a Map Control.
OPTIONLIST	Access properties of the Option-button list form control.
SELECTOR	Select the data that appears in a grid control animation.
TEXTBOX	Access properties of the Text-box form control.
TREEVIEW	Access properties of the Tree View form control.
TREND	Modify the operation of a Trend Viewer.

Input-Output Instructions

Instruction	Action
EMAIL	Basic email functions using Microsoft's MailSender component.
FTP	Copies a file to or from a file-transfer server (FTP site).
HARDCOPY	Initiate a hard copy of the screen on the Windows default printer.
KEY	Program the keyboard.
LPRINT	Send a message to a printer.
MULTIMEDIA	Drive the video and audio peripherals.
PRINTER	Manage the operation of a printer.
SVKEY	Read the contents of the protection key on multi-station systems.
SVSQL	Execute an SQL command.
SQL_COMMAND	Manage a database connected to the Supervisor using pre-configured Sql connections.
SQL_CONNECTION	Manage a database connected to the Supervisor using pre-configured Sql connections.

Mathematical Function Instructions

See Also

Instruction	Action
ACOS	Arc cosine function.
ASIN	Arc sine function.
ATAN	Arc tangent function.
COS	Cosine function.
EXP	Exponential function (base e).
IRAND	Return a random number.
LOG	Natural logarithm function (base e).
LOGICAL	Logical operations on register variables for 32-bit integers.
LOGICAL64	Logical bitwise operation on 64-bit integers.
POW	Power function.
SIN	Sine function.
SQRT	Square root function.
TAN	Tangent function.

Miscellaneous Instructions

Instruction	Action
APPLICATION	Launch an executable Windows program in parallel.
ASSOCIATEDACTIONS	Obtain the associated actions for an alarm, from a file or a buffer.
ASSOCLABEL	Changes the configuration of an associated label.
BEEP	Sound the PC's speaker.
CRONTAB	Activate or modify a scheduler action.
EXPRESSION	Import expression models or expressions on variables from a file or buffer.
FORMULA	Activate or modify a calculation formula.
GETPROJECTDIR	Get the name of the project folder.
RECIPE	Manage recipes.
SESSION	Session management.
SMS	Generating short messages.
SYSTEM	Read and write the internal system time, change the graphic mode and execute a system function.
WEBVUE	Manage the connections to WebVue clients.
XMLPATH	Process XML formatted data according to XPath (XML Path Language) specifications.

Program Execution Instructions

Instruction	Action
CYCLIC	Cyclic execution of a function.
DELAY	Suspend execution of the current program for N seconds.
EVENT	Execution of a function on the transition of a bit variable.
GETARG	Return the parameters of the calling context of a function.
PROGRAM	Enable a program to be executed, loaded, unloaded or stopped.
STOP	Stop the current program.

Program Structure Instructions

See Also

Instruction	Action
BREAK	Force an exit from a block of instructions.
FOR...NEXT	Repeat a group of instructions a certain number of times.
IF...THEN...ELSE ...END IF	Conditional execution of instructions according to the result of a logical expression.
RETURN	Exit from a subroutine and return a value.
WHILE...WEND	Execute a series of instructions in a loop while a given condition remains true.

String Manipulation Instructions

Instruction	Action
ADDSTRING	Concatenation of two character strings.
ASC	Return the ASCII code of the first character of a string.
ASCIIFIELD	Retrieve, from an ASCII buffer, ASCII fields separated by a given character.
CHR	Return the ASCII character of the code passed as an argument.
CMPSTRING	Compare two character strings.
FORMAT	Format a string, using the supplied format parameters.
LCASE	Convert all the characters of a string to lower case.
LEFT	Return the first *N* characters of a string.
LEN	Return the length of a string.
LTRIM	Return a copy of a string without any leading spaces.
MID	Return a sub-string of a character string.
RIGHT	Return the last *N* characters of the string.
RTRIM	Return a copy of a string without any trailing spaces.
SPACE	Return a string composed of *N* spaces.
STRING	Return a string composed of *N* instances of a character.
TEXTVAR	Manipulate character strings.
UCASE	Convert all the characters of a string to upper case.

Time and Date Instructions

See Also

Instruction	Action
DATETIME	Return the components of a date and time string.
DATETIMESTRING	Convert a time and date (stored as a DOUBLE) to a chain of characters.
DATETIMEVALUE	Return the number of milliseconds since 1980.
TOHMS	Convert a value in seconds to the format Hours:Minutes:Seconds.

Data Type Conversion Instructions

See Also

Instruction	Action
BIN	Return a string representing, in binary, a number passed in base 10.
CONVERT	Convert alphanumeric string to & from Hexadecimal, Octal, Binary and BCD.
DVAL	Return the numeric value of a character string as a DOUBLE.
HEX	Return a string representing, in hex, a number passed in base 10.
IVAL	Return the numeric value of a character string as an INTEGER.
LVAL	Return the numeric value of a character string as a LONG.
OCT	Return a string representing, in octal, a number passed in base 10.
SVAL	Return the numeric value of a character string as a SINGLE.
TOC	Convert a number to a STR (character string).
TOD	Convert a number to a DOUBLE.
TODOUBLE	Convert a variable to a DOUBLE value.
TOI	Convert a number to an INTEGER by truncation.
TOL	Convert a number to a LONG.
TOLL	Convert a number to a LONGLONG.
TOS	Convert a number to a SINGLE.

Variables Tree Instructions

Instruction	Action
ALARM	Test or force the state of an alarm.
GETTREE	Return the current branch.
GROUPALARM	Start or stop a selected group or all groups of alarms.
POPULATION	Create a variable population.
REFRESH_DB	Asynchronous assignment of value to variables from an ASCII file.
SENDLIST	Send a list of variables to an item of equipment.
SET	Assign a value to a variable.
STATION_FILTER	Apply a population.
SVBRANCH	Change the status of a group of variables according to their branch.
TEMPORARY_DB	Creation and deletion of temporary variables.
TREE	Selection of a branch of the database.
VARIABLE	Test or force the status of a variable.

Window Instructions

Instruction	Action
CAPTION	Display text in the title bar of the main window.
LANGUAGE	Change the working language.
REGION	Manage the number of regions on the display.
WEBVUE	Manage the connections to WebVue clients.
WINDOW	Closing and opening windows.

A

ACOS

See Also

Arc cosine function.

WebVue support - Yes.

Syntax

DblVal = ACOS(A*rc*);

The return type is DOUBLE.

Argument	Meaning
A*rc*	Any numeric type but must be in the range -1 to +1, otherwise the function returns 0.

Execution

The return value is expressed in degrees.

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
'variables required:
' @ARC   - type REGISTER, value range -1 to +1
' @ANGLE - type REGISTER, value range 0° to 360°

SUB Main()
  DIM sngarc As Single;
  DIM dblangle As Double;
  sngarc = @ARC;
  'Apply Arc cosine function
  dblangle = ACOS (sngarc);
  @ANGLE = TOS(dblangle);
END SUB
```

ADDSTRING

See Also

Concatenation of up to 10 character strings.

WebVue support - Yes.

Syntax

StrVal = ADDSTRING(*string1*, *string2[, string3[, string4[, string5[, string6[, string7[, string8[, string9[, string10]]]]]]]]*);

The return type is STR.

Argument	Range
string1, *string2 etc.*	The strings to be concatenated. Minimum of two, maximum of ten.

Execution

The maximum length of a string is 2047 characters. When the resulting length exceeds that limit, the string is truncated.

Example

This example concatenates 2 character strings.

```
'variables required:
' @TEXT01 - type TEXT
' @TEXT02 - type TEXT
' @TEXT03 - type TEXT
SUB Main()
'Declare STR variables
DIM strch1 As STR;
DIM strch2 As STR;
DIM strch3 As STR;
'Initialise
strch1="TEXT01";
strch2="TEXT02";


'Concatenate
strch3 = ADDSTRING(strch1,strch2);
PRINT("Result of ADDSTRING(strch1,strch2) = ", strch3);
END SUB
SUB AddStringProc1()
'Same function with variables
@TEXT03=ADDSTRING(@TEXT01,@TEXT02);
END SUB
```

For a further example, select the Example link above.

ALARM

See Also

Test or force the state of an alarm.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	ACK	1, 5
2	MASK	1
3	UNMASK	1
4	VALUE	1
5	ACKALL	2
6	ACKLAST	2
7	ACKTAG	3
9	ACKOLDEST	2
11	ACKDOMNAT	4
12	ACKPRIO	6
13	SETMAINTENANCE	1, 5
14	RESETMAINTENANCE	1, 5
15	GETMAINTENANCE	1
16	SETALARMLEVEL	7
17	RESTOREALARMLEVEL	1, 5
18	ALARMON	8
19	RESETANDSETALARM	1
20	GETACKLEVEL	1
21	GETMASKLEVEL	1
22	GETMAINTENANCELEVEL	1

Syntax 1

IntVal = ALARM (*Mode*, *Alarm_name*);

The return type is INTEGER.

Argument	Meaning
Alarm_name	Name of the alarm.

Execution

Mode	Mnemonic	Action
1	ACK	Acknowledge the named alarm. Return: 1 if successful, else 0.
2	MASK	Mask the named alarm. Return: 1 if successful, else 0.
3	UNMASK	Unmask the named alarm. Return: 1 if successful, else 0.
4	VALUE	Test the state of the named alarm. Return: 0 Off. 1 On, not acknowledged.

		2 Off, not acknowledged. 3 On, acknowledged. 4 Invalid (Masked).
13	SETMAINTENANCE	Force an alarm to maintenance mode. Return: 1 if successful, else 0.
14	RESETMAINTENANCE	Clear maintenance mode from an alarm. Return: 1 if successful, else 0.
15	GETMAINTENANCE	Check an alarm's maintenance status. Return: 0 The alarm does not exist. 1 Alarm in maintenance mode. 2 Alarm not in maintenance mode.
17	RESTOREALARMLEVEL	Restore the priority level of a alarm. Return: 1 if successful, else 0.
19	RESETANDSETALARM	Toggles (resets and set) an alarm that is on. It has no effect if the alarm is off. Return: 1 if successful, else 0.
20	GETACKLEVEL	Return the acknowledged level of the alarm.
21	GETMASKLEVEL	Return the mask level of the alarm.
22	GETMAINTENANCELEVEL	Return the maintenance level of the alarm.

Maintenance mode may only be set when an alarm is On and acknowledged. Maintenance mode may only be cleared when an alarm is Off.

Syntax 2

IntVal = ALARM(*Mode* [, *Flag*]);

Argument	Meaning
Flag	Enable logging of the acknowledgement as in a user acknowledgement. If the flag is 1 it will enable logging, if 0 the logging is disabled.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
5	ACKALL	Acknowledge all alarms. Return: 1 if successful, else 0.
6	ACKLAST	Acknowledge latest alarm. Return: 1 if successful, else 0.
9	ACKOLDEST	Acknowledge oldest alarm. Return: 1 if successful, else 0.

Syntax 3

IntVal = ALARM(*Mode*, *Alarm_ID* [, *Flag*]);

Argument	Meaning
Flag	Enable logging of the acknowledgement as for acknowledgement by a user. If the flag is 1 it will enable logging, if 0 the logging is disabled.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
7	ACKTAG	Acknowledge the alarm designated by the alarm ID. The alarm ID is retrieved using ALARMDISPLAY(Mode 18). Return: 1 if successful, else 0.

Syntax 4

IntVal = ALARM(*Mode*, *Domain*, *Nature*[, *Flag*]);

Argument	Meaning
Flag	Enable logging of the acknowledgement as for acknowledgement by a user. 1: enable 0: disable

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
11	ACKDOMNAT	Acknowledge all alarms with the specified Nature and Domain. Return: 1 if successful, else 0.

The time this takes to acknowledge all alarms will depend on the size of the database and the number of alarms.

A significant delay may be introduced into the Execution of the program.

Syntax 5

IntVal = ALARM (*Mode*, *Alarm_name[*, *Flag]*);

Argument	Meaning
Flag	Enable logging of the alarm for acknowledgement by a user. 1: enable (logging for the User) 0: disable

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	ACK	Acknowledge the named alarm.
13	SETMAINTENANCE	Force an alarm to maintenance mode.
14	RESETMAINTENANCE	Clear maintenance mode from an alarm.
17	RESTOREALARMLEVEL	Restore the priority level of a alarm.

Return: 1 if successful, else 0.

Syntax 6

IntVal = ALARM(*Mode*, *Priority*[, *Flag*]);

Argument	Action
Priority	An alarm priority. Type INTEGER.
Flag	Enable logging of the alarm for acknowledgement by a user.

1: enable (logging for the User)

0: disable

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
12	ACKPRIO	Acknowledge all alarms with specified priority. Return: 1 if successful, else 0.

The time taken to acknowledge all alarms will depend on the size of the database and the number of alarms. The execution of the program may be noticeably delayed .

Syntax 7

IntVal = ALARM(*Mode*, *Variable_name, Level*);

Argument	Action
Variable_name	Name of the alarm variable whose level is to be changed.
Level	Priority level of alarms (between 0 and 29).

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
16	SETALARMLEVEL	Change the priority level of an alarm. Return: 1 if successful, else 0.

The change is temporary. It does not affect the variable's configuration so it only persists during the runtime session.

Syntax 8

IntVal = ALARM(*Mode*, *Variable_name, Transition*);

Argument	Action
Variable_name	Name of the alarm variable to be changed.
Transition	The transition that turns on the alarm. 0 or 1

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
18	ALARMON	Dynamically modify the transition that turns on the alarm. If *Transition* is 0 then the alarm is turned on with a 1 to 0 transition. If *Transition* is 1 then the alarm is turned on with a 0 to 1 transition. Return: 1 if successful, else 0.

The transition is temporary. It does not affect the variable's configuration so it only persists during the runtime session.

Example

This example tests or forces the state of a variable. It uses the ACK mode (syntax 1) of ALARM.

```
SUB AckVar (variance)
DIM res As Integer;
res = ALARM(1,VarName); ' acknowledge alarm
Print ("AckVar returned ",res);
END SUB

SUB AckAllAlarms ()
DIM res As Integer;
res = ALARM("ACKALL");
Print ("AckAllALarms returned ",res);
END SUB
```

This example puts Alarm1 into maintenance mode:

```
ALARM("SETMAINTENANCE", "Alarm1", 1 );
```

This example takes Alarm1 out of maintenance mode:

```
ALARM("RESETMAINTENANCE", "Alarm1", 1);
```

For more examples, select the Example link above.

ALARMDISPLAY

See Also Example Further information

Alarm Viewer management including list navigation, actions on alarms and filtering.

Partial WebVue support - see mode table.

If using a project with multiple regions, you must set the region before executing any instructions that interacts with the HMI. For further information see the REGION topic.

Mode	Mnemonic	Syntax	WebVue
0	BEGIN	1	Yes
1	BEFORE	1	Yes
2	SELECT	8	No
3	AFTER	1	Yes
4	END	1	Do not use
5	LINEUP	1	Yes
6	LINEDOWN	1	Yes
7	MODE	1	No
8	DYNAMIC	1	No
9	DOMAIN	2	Yes
10	NATURE	2	Yes
11	MINPRIO	3	Yes
12	MAXPRIO	3	Yes
13	ACK_ON	4	Yes
14	ACK_OFF	4	Yes
15	NOACK_ON	4	Yes
16	NOACK_OFF	4	Yes
17	NS	4	Yes
18	SELECTED	5	Yes
19	SELECNAME	5	Yes
20	ACTION1	1	No
21	ACTION2	1	No
22	FORMAT	15	No
23	USERMASK	4	Yes
24	LIST	1	No
25	PRINTALL	1, 11	No
26	FILTER	7	Yes
27	ACK_SELECTED	1	Yes
28	ACK_DISPLAY	1	Yes
29	MASK_SELECTED	1	Yes
30	UNMASK_SELECTED	1	Yes
31	PRINT_SELECTED	1, 11	No
32	PRINT_DISPLAY	1, 11	No
33	PROGMASK	4	Yes
34	VARMASK	4	Yes
35	IS_ACTION1	1	No
36	IS_ACTION2	1	No
37	DATERANGE	6	No
38	MAINTENANCEMASK	4	Yes
39	UNSELECTALL	1	No
40	ONLINESELECT	9	Yes
41	SELECDATETIME	10	No
42	MASK_DISPLAY	1	Yes
43	UNMASK_DISPLAY	1	Yes
44	LINESELECT	12	Yes
45	SETSORT	13	Yes
46	GETSORT	14	Yes
47	ISLINEVISIBLE	16	No
48	ISLINESELECTED	16	No

49	GETLINECOUNT	1	No
50	GET_NAME_FROM_LINE	17	No
51	COPY_CLIPBOARD	18	No
52	GETCELL	20	No
53	GETLINES	19	No
54	GETSELECTEDLINES	19	No

Arguments for all modes

Argument	Meaning
Window	The name of the window that contains the Alarm Viewer to be used. Type STR.
Branch	The branch (if any) of the window. Use "#B" to indicate the current branch of the program. Type STR
Identity	The identity of the Alarm Viewer animation in the specified window. Type STR.

Syntax 1

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	BEGIN	Go to the start of the list (oldest alarm).
1	BEFORE	Scroll list up one page.
3	AFTER	Scroll list down one page.
4	END	Go to end of list (newest alarm).
5	LINEUP	Scroll list up one line.
6	LINEDOWN	Scroll list down one line.
7	MODE	Change from scroll mode to list mode.
8	DYNAMIC	Change to scroll mode.
20	ACTION1	Execute the action no. 1 associated with the selected alarm.
21	ACTION2	Execute the action no. 2 associated with the selected alarm.
24	LIST	Change from any mode to list mode.
25	PRINTALL	Print the entire contents of the current buffer on the printer configured in the Alarm Viewer.
27	ACK_SELECTED	Acknowledge the selected alarm.
28	ACK_DISPLAY	Acknowledge all alarms visible in the display.
29	MASK_SELECTED	Mask the selected alarm.
30	UNMASK_SELECTED	Unmask the selected alarm.
31	PRINT_SELECTED	Print the selected alarm.
32	PRINT_DISPLAY	Print all alarms visible in the display.
		Return for all modes above: 1 if successful, else 0.
35	IS_ACTION1	Check whether the selected alarm has a configured associated Action1.
36	IS_ACTION2	Check whether the selected alarm has a configured associated Action2.
		Return for modes 35 and 36: 1 if an action is configured

		0 if no action is configured -1 if no alarm is selected.
39	UNSELECTALL	Deselects all lines of the display.
42	MASK_DISPLAY	Mask all alarms visible in the display.
43	UNMASK_DISPLAY	Unmask all alarms visible in the display.
49	GETLINECOUNT	Return the number of lines in the display. Return for all modes unless noted otherwise: 1 if successful, else 0 (non-existent window, animation of wrong type, non-existent position).

Syntax 2

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Name*);

The return type is INTEGER.

Argument	Meaning
Name	The name of the new Domain or Nature for the Alarm Viewer. To specify all Domains and all Natures, Name must be a null string "". Type STR.

Execution

Select a new Domain or Nature for the Alarm Viewer.

Mode	Mnemonic	Action
9	DOMAIN	The argument *Name* supplies a new Domain.
10	NATURE	The argument *Name* supplies a new Nature.
		Return for all modes: 1 if successful, else 0.

Syntax 3

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Priority*);

The return type is INTEGER.

Argument	Meaning
Priority	The new priority level for the alarm filter.

Priority must be a numerical type. If the priority is out of range (<0 or >29) the function returns 0.

Execution

Select a new minimum or maximum alarm priority for the alarm filter.

Mode	Mnemonic	Action
11	MINPRIO	The argument *Priority* supplies a new minimum priority.
12	MAXPRIO	The argument *Priority* supplies a new maximum priority. Return for both modes: 1 if successful, else 0.

Syntax 4

Ret = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Flag*);

The return type is INTEGER.

Argument	Meaning
Flag	An integer which when set to 1 will activate, and when set to 0 will deactivate, the

selected mode.

Execution

Enable and disable alarm states in the alarm filter.

Mode	Mnemonic	Action
13	ACK_ON	Alarm state ON ACKNOWLEDGED.
14	ACK_OFF	Alarm state OFF.
15	NOACK_ON	Alarm state ON NOT ACKNOWLEDGED.
16	NOACK_OFF	Alarm state OFF NOT ACKNOWLEDGED.
17	NS	Alarm state INVALID.
23	USERMASK	Alarm state MASKED.
33	PROGMASK	Alarm state MASKED BY PROGRAM.
34	VARMASK	Alarm state MASKED BY VARIABLE.
38	MAINTENANCEMASK	Alarm state MAINTENANCE.
		Return for all modes: 1 if successful, else 0.

Syntax 5

StrVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*);

The return type is STR.

Execution

Mode	Mnemonic	Action
18	SELECTED	Return the ID of the last selected alarm on the display. This is used in conjunction with ALARMDISPLAY mode 7 (MODE).
19	SELECNAME	Return the name of the last selected alarm on the display. This is used in conjunction with ALARMDISPLAY mode 7 (MODE). A selected alarm is shown in reverse video.

Syntax 6

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *StartDate*, *EndDate*);

The return type is INTEGER.

Argument	Meaning
StartDate	The start date for selection, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue.
EndDate	The end date for selection, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue.

Execution

Mode	Mnemonic	Action
37	DATERANGE	Selects the current realtime alarms by date and time and displays them in the animation. Return: 1 if OK, else 0 (window does not exist, etc.)

To cancel the filter, you must set Start Date and End Date to 0.

Notes for particular cases:

If only the Start Date is set (i.e. if End Date is 0): the animation will display only alarms from the Start Date onwards, plus those occurring in real time up to the number of alarms set in the list.

If only the End Date is given (i.e. if Start Date is 0): the animation will return only alarms occurring since the oldest effective one, up to the End Date and subject to the number of alarms set in the list).

Syntax 7

Ret = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Filter*);

Argument	Meaning
Filter	A filter expression. See the topic Native Filter Expressions for information on filter expressions. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
26	FILTER	Selects information to be displayed, by applying the filter expression. Return: 1 if successful, else 0 (the window does not exist, etc.).

Syntax 8

IntVal =ALARMDISPLAY(*Mode, Window, Branch, Identity, AlarmName, Selected*);

Argument	Meaning
AlarmName	The name of the alarm to which the selected line refers.
Selected	Required state of the line: selected (0), not selected (1).

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	SELECT	Select or de-select a line of the Alarm Viewer, in List mode. Return: 1 if successful, else 0 (non-existent window, animation of wrong type, non-existent position, variable not found).

Syntax 9

IntVal =ALARMDISPLAY(*Mode, Window, Branch, Identity, Program, Branch, Function, Farg*);

Argument	Meaning
Program	The program to be used.
Function	The function to be run.
Farg	Contains from 1 to 8 arguments separated by "," (comma). (Optional, maximum 255 characters)

You can use the GETARG command, at the beginning of the function that is to be ran, to retrieve the values of the arguments contained in *Farg*.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
40	ONLINESELECT	Respond to the event of selection of a line. Enables you to specify a function that will be run when a line in the display animation is selected, either by mouse click or by program with mode SELECT. Return: 1 if successful, else 0 (non-existent window, animation of wrong type, non-existent position).

Syntax 10

DblVal =ALARMDISPLAY(*Mode, Window, Branch, Identity*);

The return type is DOUBLE. (It can be 0 if no selection has occurred.)

Execution

Mode	Mnemonic	Action
41	SELECDATETIME	Retrieve the date-time of the most recently selected alarm in an Alarm Viewer. Return: The date and time as the number of milliseconds since the 1st January 1980.

The instruction's mnemonic is 'SELECDATETIME' (not 'SELECT...').

Syntax 11

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*[, *PrintFormat*]);

The return type is INTEGER.

Argument	Meaning
PrintFormat	Optional parameter for selecting the format for printing: 0: Print the alarm(s) using the display format defined in the Alarm Viewer's Display tab. (default). 1: Print the alarm(s) using the print format defined in the Alarm Viewer's Execution tab.

Execution

Mode	Mnemonic	Action
25	PRINTALL	Print the entire contents of the current buffer on the printer configured in the Alarm Viewer.
31	PRINT_SELECTED	Print the selected alarm.
32	PRINT_DISPLAY	Print all alarms visible in the display. Return for these modes: 1 if successful, else 0.

Syntax 12

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Program, ProgramBranch, Function, Arguments*);

Argument	Meaning
Program	Module of the program to be executed.
ProgramBranch	Branch for the program.
Function	Function of the program to be executed.

Arguments	Arguments for the program.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
44	LINESELECT	Specifies a program to be executed when a line select/unselect occurs. Return: 1 if successful, else 0 for a bad parameter.

The XMLPATH instruction is used to get information about the LINESELECT mode. A namespace is created automatically using Branch, Mimic and Identity parameters. See the topic on XMLPATH for its format.

The meanings of the XML paths are as follows.

Path	Meaning
lineselect/variable	Variable name.
lineselect/x	X position
lineselect/y	Y position
lineselect/selected	Selected/unselected
lineselect/time	Date-time
lineselect.element[1-8].value	Column element

Syntax 13

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Column*, *Sort*);

Argument	Meaning
Column	Values 0 to 7 to identify the column, -1 to reset the sort.
Sort	Values 1 for ascending, 0 for descending.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
45	SETSORT	Dynamically sort an alarm list by a column. Return: 1 if successful, else 0 for a bad parameter.

Syntax 14

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Column*, *Sort*);

Argument	Meaning
Column	Values 0 to 7 to identify the column, -1 to reset the sort.
Sort	Values 1 for ascending, 0 for descending.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
46	GETSORT	Obtain the status of the sort. Return: 1 if successful, else 0 for a bad parameter.

The XMLPATH instruction is used to get information about the GETSORT mode. A namespace is created automatically using Branch, Mimic and Identity parameters. See the topic on XMLPATH for its format.

The meanings of the XML paths are as follows.

Path	Meaning
getsort/column	Values 0 to 7 to identify the column, -1 to reset the sort.
getsort/sort	Values 1 for ascending, 0 for descending.

Syntax 15

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *DateFormat*);

The return type is INTEGER.

Argument	Meaning
DateFormat	A string defining a new display format for dates in the alarm data.

Execution

Select a new display format for the Alarm Viewer.

Mode	Mnemonic	Action
22	FORMAT	The argument D*ateFormat* supplies a new display format. An example of its syntax is as follows: #h:#m:#s\t#D/#M/#y\t#T\t#E where \t marks the start of each column and the substitution characters are those used in configuring an alarm viewer. It is not possible to change the column titles dynamically.

Return: 1 if successful, else 0.

Syntax 16

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *LineNumber*);

The return type is INTEGER.

Argument	Meaning
LineNumber	A number corresponding to the line position.

Execution

Select a new display format for the Alarm Viewer.

Mode	Mnemonic	Action
47	ISLINESELECTED	Test if the given line is selected. Return: 1 if the line is selected, 0 if the line is not selected, -1 if the line does not exist.
48	ISLINEVISIBLE	Test if the given line is visible. Return: 1 if the line is visible, 0 if the line is in the buffer but not visible, -1 if the line does not exist

Syntax 17

StrVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *LineNumber*);

The return type is STRING.

Argument	Meaning
LineNumber	A 1 based number corresponding to the line position.

Execution

Select a new display format for the Alarm Viewer.

Mode	Mnemonic	Action
50	GET_NAME_FROM_LINE	Return the alarm name for the given line.

Syntax 18

IntVal = ALARMDISPLAY(*Mode*, *Window*, *Branch*, *Identity, Separator*);

Argument	Meaning
Separator	The character or characters used to delimit the contents of each column. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
51	COPY_CLIPBOARD	Copy the selected lines to the Windows clipboard. Return: 1 if successful, else 0

Syntax 19

StrVal = ALARMDISPLAY(*Mode, Window, Branch, Identity, LineStart[, LineEnd, ColSeparator, LineSeparator]*);

Argument	Meaning
Linestart	The 1 based numeric index of a line in the Alarm Viewer buffer. Type INTEGER.
LineEnd	The 1 based numeric index of a line in the Alarm Viewer buffer. Type INTEGER.
ColSeparator	The character or characters used to separate the columns in the returned string. The default is the comma ",". Type STR.
LineSeparator	The character or characters used to separate the lines in the returned string. The default is new line "\n". Type STR.

The return type is STR.

Execution

Mode	Mnemonic	Action
53	GETLINES	Return the lines from LineStart to LineEnd (inclusive) as a single string delimited using the ColSeperator and LineSeperator characters. If LineEnd is omitted only a single line is returned with the columns delimited using the default characters. Return: Selected lines of the Alarm Viewer.
54	GETSELECTEDLINES	Same as GETLINES except that only selected lines are returned.

Syntax 20

StrVal = ALARMDISPLAY(*Mode, Window, Branch, Identity, Line, Column*);

Argument	Meaning
Line	The 1 based numeric index of a line in the Alarm Viewer buffer. Type INTEGER.
Column	The 1 based numeric index of a column in the Alarm Viewer buffer. Type INTEGER.

The return type is STR.

Execution

Mode	Mnemonic	Action
52	GETCELL	Return the contents of the specified cell.

Example

For an example, select the Example link above. For syntaxes 12 and 14 in particular, see similar examples of LOGDISPLAY modes 23 and 25.

ALLOC_BUFFER

See Also

Allocates a memory area of *N* bytes.

WebVue support - Yes.

Syntax

LongVal = ALLOC_BUFFER(*N*);

The return type is LONG.

Execution

The return value points to the start of the allocated block and is used subsequently for referencing the reserved memory area.

Changing the buffer size

The maximum buffer size can be changed from its default by adding the following lines to the configuration file UICONF.DAT found in the project's C folder.

```
[ScadaBasic\Alloc_Buffer]
MaxSize = 10
```

The maximum permitted size is 100 Mb. If set to 0 then the default of 128 Kb is used.

If you are allocating large memory buffers you must pay particular attention to de-allocating the memory buffer after it has been used.

Example

This example allocates a memory area of *n* bytes.

```
SUB Main()
DIM hbuffer As Long;
DIM intvalue As Integer;
intvalue = 50;

hbuffer = ALLOC_BUFFER(intvalue);
PRINT("Buffer handle = ",hbuffer);

'After using the memory area created with ALLOC_BUFFER,
'always release the memory area
FREE_BUFFER(hbuffer);
END SUB
```

ANIMATION

See Also

Executes the equivalent action of a user clicking on a control zone type animation.

WebVue support - No.

Mode	Mnemonic	Syntax
1	EXECUTE	1

Syntax 1

ANIMATION(*Mode*, *Window*, *Branch*, *Identity*)

The return type is INTEGER.

Argument	Meaning
Window	The name of the window that contains the animation to be used. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR
Identity	The identity of the animation in the specified window. Type STR.

Execution

Mode	Mnemonic	Action
1	EXECUTE	Executes Simulates the equivalent action of a uUser clicking on a control zone type animation. Returns: 1 if OK, else 0.

Example

This example initiates an action from an animation. To use it:

1. Create a mimic called MENU.
2. Draw a button in the mimic and configure its identity using the Graphic Explorer as BUTTON01.
3. Add a control zone type animation (for example Send Bit) to the button.

```
SUB AnimationExecute()
ANIMATION("EXECUTE","MENU","","BUTTON01");
END SUB
```

APPLICATION

See Also Example

Launch an executable program. If executed in a WebVue session context the application is started on the computer that is running the web back end.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	ISLOADED	1
2	ACTIVATE	1
3	LOAD	2
4	UNLOAD	1

Syntax 1

IntVal = APPLICATION(*Mode*, *Command*);

The return type is INTEGER.

Argument	Meaning
Command	The command line which is used to launch the program. This would typically include the full path for the executable. Type STR.

Execution

Mode	Mnemonic	Action
1	ISLOADED	Test if a program is already loaded. Return: 1 if loaded, else 0.
2	ACTIVATE	Activate a program that is already loaded. The program window is brought to the front and given focus. Return: 1 if successful, else 0.
4	UNLOAD	Unload a program. Return: 1 if successful, else 0.

Syntax 2

IntVal = APPLICATION(*Mode*, *Command*[, *Argument*[, *Start*][,*Folder*]]);

The return type is INTEGER.

Argument	Meaning
Command	The program file name. This would typically include the full path (drive and folder) for the executable. Type STR.
Argument	Options for the command line of the application. Optional. Type STR.
Start	The mode in which the program will start. Type INTEGER. The program will be loaded: 1 Normal, active. This is the default. 2 Normal, minimised. 3 Normal, maximised. 4 Normal, inactive (window in background). 5 Minimised, inactive (window in background).
Folder	The working path of the program. The character string must present a full path including drive and folder. Type STR.  If the path is incorrect, the program will have the Supervisor's drive and folder.

Execution

Mode	Mnemonic	Action
3	LOAD	Load and run a program. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

ASC

See Also

Return the ASCII code of the first character of a string.

WebVue support - Yes.

Syntax

IntVal = ASC(*string*[, *iExtended*]);

The return type is INTEGER.

Execution

Argument	Meaning
string	A text string. Only the first character is used.
iExtended	For use with characters with the extended ANSI codes. Optional. 0: (default) returns a negative value when the code is greater than 127. To get the correct code 256 must be added. 1: returns a positive value when its code is greater than 127.
IntVal	The return value is in the range 0 to 255. If the string is null the function returns 0.

The return value is in the range from 0 to 255. If the chain is null the function returns 0.

Example

This example lists the ASCII codes of numbers up to 256.

```
Sub TestASC()
Dim i As Integer;
Dim c As Str;
Dim j As Integer;

For (i=0; i<256; i++)
 c = Chr (i);
 j = Asc (c, 1);
 Print (i, " ", c, " ", j );
Next
End Sub
```

ASCIIFIELD

See Also Example

Retrieve, from an ASCII buffer, ASCII fields separated by a given character.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	LEN	1
2	COUNT	2
3	STR	1
4	INTEGER	1
5	LONG	1
6	DOUBLE	1

The ASCII buffer should be terminated by a linefeed character - either \n or CHR(10).

If you do not terminate the buffer correctly the last (right-most) field is not recognized.

You cannot enter a linefeed in the Supervisor's Send Text animation.

Syntax 1

RetVal = ASCIIFIELD(*Mode*, *Handle*, *Index* [, *Sepa*]);

Argument	Meaning
Handle	The handle of the buffer on which the operation is performed. Type LONG.
Index	The index of the field starting at 1. Type INTEGER.
Sepa	The separation character used. The default value is the comma: ",". Type STR.

Execution

Mode	Mnemonic	Action
1	LEN	Return the length of the field specified by the index. If the index is not given, the length (size) of the buffer is returned. The return type is INTEGER.
3	STR	Return the character string of the field specified by the index. If the index is out of range the character string will be empty. The return type is STR.
4	INTEGER	Return the INTEGER value contained in the field specified by the index. If the index is out of range the function returns 0. The return type is INTEGER.
5	LONG	Return the LONG value contained in the field specified by the index. If the index is out of range the function returns 0. The return type is LONG.
6	DOUBLE	Return the REAL value contained in the field specified by the index. If the index is out of range the function returns 0. The return type is DOUBLE.

Syntax 2

RetVal = ASCIIFIELD(*Mode*, *Handle* [, *Sepa*]);

Argument	Meaning
Handle	The handle of the buffer on which the operation is performed. Type LONG.
Sepa	The separation character used. The default value is the comma: ", ". Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	COUNT	Return the number of fields in the buffer.

Example

For an example, select the Example link above.

ASIN

See Also

Arc sine function.

WebVue support - Yes.

Syntax

DblVal = ASIN(*arc*);

The return type is DOUBLE.

Execution

Argument	Meaning
arc	Any numeric type but must be in the range -1 to 1, otherwise the function returns 0. The return value is expressed in degrees.

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
'variables required:
' @ARC   - type REGISTER, value range -1 to +1
' @ANGLE - type REGISTER, value range 0° to 360°

SUB Main()
DIM sngarc As Single;
DIM dblangle As Double;
sngarc = @SB_EXAMPLES.TRIG.ARC;

'Apply arc sine function
dblangle = ASIN (sngarc);
@SB_EXAMPLES.TRIG.ANGLE = TOS(dblangle);
Delay(0.1);
PRINT (@SB_EXAMPLES.TRIG.ANGLE);
END SUB
```

ASSOCIATEDACTIONS

See Also File formats

Import the Associated Actions for an alarm, from a file or a buffer.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	IMPORTBYFILE	1
1	IMPORTBYHANDLE	2

The file formats are shown in the tables of the topic Associated Action Formats.

Syntax 1

IntVal = ASSOCIATEDACTIONS(*Mode*, *FileName*);

The return type is INTEGER.

Argument	Meaning
FileName	Name of the text file in the TP folder of the project or in a folder selected by an absolute path. Type STR.

Execution

Mode	Mnemonic	Action
0	IMPORTBYFILE	Import the associated action configuration for the alarms from the specified file and folder.

Returns: 0 if OK, else 1.

Example

```
ASSOCIATEDACTIONS(0, "addAAA.dat");
ASSOCIATEDACTIONS("IMPORTBYFILE", "addAAA.dat");
```

Syntax 2

IntVal = ASSOCIATEDACTIONS(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning
Handle	Address of the buffer memory area as allocated by ALLOC_BUFFER. Type LONG.

Execution

Mode	Mnemonic	Action
1	IMPORTBYHANDLE	Import associated action configiration for the alarms from a buffer. Returns: 1 if OK, else 0.

Returns: 0 if OK, else 1.

Example

```
DIM hbuf AS LONG;
hbuf = FILETOBUF("addAAA.dat");
ASSOCIATEDACTIONS(1, hbuf);
'OR
ASSOCIATEDACTIONS("IMPORTBYHANDLE", hbuf);
```

ASSOCLABEL

See Also Example

Changes the configuration of an associated label.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	SETLABEL	1

Syntax 1

IntVal = ASSOCLABEL(*Mode*, *AssocLabelName*, *EventCode*, *Label* [, *Language*]);

The return type is INTEGER.

Argument	Meaning
AssocLabelName	Name of the associated label.
EventCode	Code for the type of event: 0 state is 0 1 state is 1 2 change to 0 3 change to 1 4 command to 0 5 command to 1
Label	The new string for the event.
Language	Number of language: 0 or 1. Return: 1 if successful, else 0.

Execution

Mode	Mnemonic	Action
1	SETLABEL	Change the label associated with the *EventCode.*

Example

For an example, select the Example link above.

ATAN

See Also

Arc tangent function.

WebVue support - Yes.

Syntax

DblVal = ATAN(*arc*);

The return type is DOUBLE.

The argument *arc* may be any numeric type.

Execution

Argument	Meaning
arc	The return value is expressed in degrees.

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
'variables required:
' @ARC   - type REGISTER, value range -1 to +1
' @ANGLE - type REGISTER, value range 0° to 360°

SUB Main()
DIM sngarc As Single;
DIM dblangle As Double;
sngarc = @ARC;

'Apply Arc tangent function
dblangle = ATAN (sngarc);
@ANGLE = TOS(dblangle);
END SUB
```

B

BACNET

See Also Example

To execute commands relating to the BACnet protocol.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	RESET_PRIORITY	1
1	RESET_ALL_PRIORITIES	2
2	TIME_SYNCHRONIZATION	3
3	START_NETWORK	4
4	STOP_NETWORK	4
5	START_DEVICE	5
6	STOP_DEVICE	5
7	WRITE_PRIORITY	6
8	RESTART_NETWORK	4
9	START_NOTIFICATION	7
10	STOP_NOTIFICATION	7
11	START_LOG	8
12	STOP_LOG	8
13	LOG_RETRIEVAL	9
14	BACKUP_DEVICE	5
15	RESTORE_DEVICE	5

You can specify the default value for writing and resetting the priority in a variable's Advanced properties: BACnet.Write priority.

Syntax 1

Return = BACNET(*Mode*, *VariableName*, *Priority*[, *ResultVar*]);

Argument	Meaning
VariableName	The name of a BACNET variable in the Supervisor. Type STR.
Priority	The priority of the BACnet variable in the Supervisor (between 1 and 16, or 0 to use the default Write priority). Type INTEGER.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. Optional. Its values are: 0: The command is pending. 1: The command has finished. -1: The variable is not recognized. -2: The variable is not a BACnet variable. -3: The BACnet item is disabled. -4: The priority number is invalid. -5: The reset has failed. Return: 0: OK.. -1: The command mode is not recognized. -2: The variable name is missing. -3: The priority is missing.

Execution

Mode	Mnemonic	Action
0	RESET_PRIORITY	Resets the value written at a particular priority level.

Syntax 2

Return = BACNET(*Mode*, *VariableName*[, *ResultVar*]);

Argument	Meaning
VariableName	The name of a BACnet variable in the Supervisor. Type STR.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has finished. -1: The variable is not recognized. -2: The variable is not a BACnet variable. -3: The BACnet item is disabled. Return: 0: OK.. -1: The command mode is not recognized. -2: The variable name is missing.

Execution

Mode	Mnemonic	Action
1	RESET_ALL_PRIORITIES	Resets the priority values of variables at all priority levels.

Syntax 3

Return = BACNET(*Mode*, *NetworkName*[, *ResultVar*]);

Argument	Meaning
NetworkNam e	The name in the Supervisor of a BACnet network. Type STR.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are:0: The command is pending. 1: The command has finished. -1: The network is not recognized. The result variable confirms the sending of the time synchronization command but the BACnet devices do not send an acknowledgement. Return: 0: OK. -1: The command mode is not recognized. -2: The network name is missing.

Execution

Mode	Mnemonic	Action
2	TIME_SYNCHRONIZATION	To synchronize the time for all devices on a network.

Syntax 4

Return = BACNET(*Mode*, *NetworkAlias*[, *ResultVar*]);

Argument	Meaning
NetworkAlias	The name in the Supervisor of a BACnet network. Type STR.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has been sent to the device. -1: The network is not recognized. In the case that the command fails (-2) you can check the BACnet status variables for more information. Return:0: OK. -1: The command mode is not recognized. -2: The network name is missing.

Execution

Mode	Mnemonic	Action
3	START_NETWORK	Starts the network.
4	STOP_NETWORK	Stops the network.
8	RESTART_NETWORK	Restarts the network.

Syntax 5

Return = BACNET(*Mode*, *NetworkAlias*, *DeviceAlias*[, *ResultVar*]);

Argument	Meaning
NetworkAlias	The name in the Supervisor of a BACnet network. Type STR.
DeviceAlias	The name in the Supervisor of a BACnet device. Type STR.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has been sent to the device. -1: The device is not recognized. -2: The command has failed. -3: The command has failed as a backup or restore is already in progress. In the case that the command fails (-2) you can check the BACnet status variables for more information. Return: 0: OK. -1: The command mode is not recognized. -2: The network name is missing. -3: The device name is missing.

Execution

Mode	Mnemonic	Action
5	START_DEVICE	Starts the device.
6	STOP_DEVICE	Stops the device.
14	BACKUP_DEVICE	Backup the device configuration.
15	RESTORE_DEVICE	Restore the device configuration.

Syntax 6

Return = BACNET(*Mode*, *VariableName*, *Value*, *Priority*[, *ResultVar*]);

Argument	Meaning
VariableName	The name of a BACnet variable in the Supervisor. Type STR.
Value	The value to be written. Type - must be the same type as the variable that is to be written.
Priority	The priority of the BACnet variable in the Supervisor (between 1 and 16, or 0 to use the default Write priority). Type INTEGER.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has finished. -1: The variable is not recognized. -2: The variable is not a BACnet variable. -3: The BACnet item is disabled. -4: The priority number is invalid. -5: The write has failed. Return:0: OK. -1: The command mode is not recognized. -2: The variable name is missing. -3: The priority is missing. -4: The type of variable is not recognized.

-5: The value type is different from the variable type.

Execution

Mode	Mnemonic	Action
7	WRITE_PRIORITY	Writes the variable with a particular priority.

Syntax 7

Return = BACNET(*Mode*, *NetworkAlias*, *DeviceAlias*, *NotificationAlias*[, *ResultVar*]);

Argument	Meaning
NetworkAlias	The name, in the Supervisor, of a BACnet network. Type STR.
DeviceAlias	The name, in the Supervisor, of a BACnet device. Type STR.
NotificationAlias	The name, in the Supervisor, of a BACnet notification. Type STR.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has been sent to the device. -1: The notification does not exist. -2: The command has failed. In the case that the command fails (-2) you can check the BACnet stat variables for more information. Return: 0: OK. -1: The command mode is not recognized. -2: The network name is missing. -3: The device name is missing. -4: The notification name is missing.

Execution

Mode	Mnemonic	Action
9	START_NOTIFICATION	Start notifications.
10	STOP_NOTIFICATION	Stop notifications.

Syntax 8

Return = BACNET(*Mode*, *NetworkAlias*, *DeviceAlias*, *LogAlias*[, *ResultVar*]);

Argument	Meaning
NetworkAlias	The name, in the Supervisor, of a BACnet network. Type STR.
DeviceAlias	The name, in the Supervisor, of a BACnet device. Type STR.
LogAlias	The name, in the Supervisor, of a BACnet log. Type STR.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has been sent to the device. -1: The log does not exist. -2: The command has failed. In the case that the command fails (-2) you can check the BACnet statu variables for more information. Return: 0: OK. -1: The command mode is not recognized. -2: The network name is missing. -3: The device name is missing. -4: The log name is missing.

Execution

Mode	Mnemonic	Action

11	START_LOG	Start log.
12	STOP_LOG	Stop log.

Syntax 9

Return = BACNET(*Mode*, *NetworkAlias*, *DeviceAlias*, *LogAlias*, *Date, Count*[, *ResultVar*]);

Argument	Meaning
NetworkAlias	The name, in the Supervisor, of a BACnet network. Type STR.
DeviceAlias	The name, in the Supervisor, of a BACnet device. Type STR.
LogAlias	The name, in the Supervisor, of a BACnet log. Type STR.
Date	The reference timestamp for the first record to read, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE.
Count	The number of records to read. Type INTEGER.
ResultVar	The name of a register variable to contain the result of the instruction. Type STR. (Optional) Its values are: 0: The command is pending. 1: The command has been sent to the device. -1: The log does not exist. -2: The requested read range does not exist. In the case that the command fails (-2) you can check the BACnet status variables for more information. Return: 0: OK. -1: The command mode is not recognized. -2: The network name is missing. -3: The device name is missing. -4: The log name is missing. -5: The result variable does not exist. -6: The date is incorrect. -7: The count is less that -1000, or greater than 10000

Execution

Mode	Mnemonic	Action
13	LOG_RETRIEVAL	Retrieve the log buffer using the supplied parameters. If the date is empty or defined and the count is 0 or not defined, the buffer will be read in its entirety device allows.

Example

For an example, select the Example link above.

BEEP

See Also

Sound the PC's speaker. To generate a sound on a WebVue client see WEBVUE.

WebVue support - Not applicable. Returns the unsuccessful code if used in this context.

Syntax

IntVal = BEEP([Freq, [Period]]);

The return type is INTEGER.

Argument	Meaning
Freq	The frequency of the generated sound in hertz. Type DOUBLE.
Period	The length of the generated sound in milliseconds. Type DOUBLE.

Execution

Plays a sound through the desktop client PC's speaker.

Return: 1 if successful, else 0.

There is a delay built in to the use of the BEEP instruction which means it can only be used about once every 2 seconds.

Avoid long periods as, while BEEP is being executed, operation of the HMI is suspended.

Example

This example causes a sound at 1000Hz for a tenth of a second.

```
SUB main()
DIM dblfrequency as DOUBLE; 'in Hz
DIM dbldelay as DOUBLE;  'in ms

dblfrequency = 1000;
dbldelay = 100;
BEEP(dblfrequency,dbldelay);
END SUB
```

BIN

See Also

Return a string representing, in binary, a number passed in base 10.

WebVue support - Yes.

Syntax

StrVal = BIN(*N*);

The return type is STR.

N may be of any numeric type.

Execution

Argument	Meaning
N	A string representing, in binary, a number in base 10.

The length of the returned string depends on the value of the number to be converted.

Example

This example gives the binary equivalent of a decimal number.

```
SUB Main()
DIM intNumber as integer;
DIM strResult as Str;

intNumber = 3;
strResult = BIN(intNumber);
PRINT(intNumber," in base 10 => ",strResult," in base 2");
'3 in base 10 => 11 in base 2
END SUB
```

BREAK

See Also

Force an exit from a block of instructions.

WebVue support - Yes.

Syntax

BREAK;

No return.

Execution

Forces an exit from a block of instructions.

Example

```
SUB main()
DIM i As Integer;
For (i=1;i<1000;i++)
    If (i>500) Then
       BREAK;
    End If
Next

PRINT("value = ",i);
'value = 501
END SUB
```

BUFTOEXCEL

See Also

Create an Excel file in XLSX format using the contents of a memory buffer.

WebVue support - Yes.

Syntax

IntVal = BUFTOEXCEL (*Handle*, *LineSeparator*, *ColumnSeparator*, *WorkbookPath*, *SheetName*, *FileMode*, [Offset*LineNumber*, Offset*ColumnNumber*])

Argument	Meaning
Handle	The handle of a memory buffer. Type LONG.
LineSeperator	The line separator used in buffer (one character such as ';' or '\n'). Type STR.
ColumnSeperator	The column separator used in buffer (one character such as ',' or '\t'). Type STR.
WorkBookPath	The full path of the workbook including the file name. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the path must be valid to that computer. For example E:\\WorkBooks\\MyBook.xlsx.
SheetName	The name of the worksheet. Type STR.
FileMode	The mode for writing the file. Type STR. CLEARSHEET - The workbook and the sheet are created and characters from the memory buffer are written to it until the first null character is encountered in the buffer. If the sheet already exists any text content in the cells is cleared. MERGE - The workbook and the sheet are created and characters from the memory buffer are written to it until the first null character is encountered in the buffer. If the sheet already exists any existing content of written cells is overwritten. APPEND - The workbook and the sheet are opened and characters from the memory buffer are appended after the last line until the first null character is encountered in the buffer.
Offset*LineNumber*	Offset of the line to write in the Excel sheet. For the APPEND mode it is an offset from the last line otherwise it is from the origin. The default value is 1. Type INTEGER.
Offset*ColumnNumber*	Offset of the column to write in Excel sheet. The default value is 1.

Execution

Action

Create an Excel file in XLSX format using the contents of a memory buffer.
Return:
 0 - OK
-1 - Buffer parameter invalid
-2 - Buffer unknown
-3 - Line separator parameter invalid
-4 - Column separator parameter invalid
-5 - WorkbookPath parameter invalid
-6 - SheetName parameter invalid
-7 - FileMode parameter invalid
-8 - FileMode unknown
-9 - OffsetLineNumber parameter must greater than 1 when used
-10 - OffsetColumnNumber parameter must greater than 1 when used
-11 - Bad buffer content (column count is not the same for each line)
-12 - Write operation failed (details in Event Viewer)

BUFTOFILE

See Also Example

Create an ASCII file using the contents of a memory buffer.

WebVue support - Yes.

Action	Syntax
Create or overwrite	1
Write or append	2

Syntax 1

IntVal = BUFTOFILE(*Handle*, *FileName*);

The return type is INTEGER.

Argument	Meaning
Handle	The start of the memory block. Type LONG.
FileName	The name of the file to be created or overwritten. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file is created on that computer.

Execution

The named file is created using the contents of the specified memory buffer.

If the file already exists it will be overwritten.

Return: 1 if successful, else 0.

Syntax 2

IntVal = BUFTOFILE(*Handle*, *FileName*, *BufferMode*[, *FileMode*]);

The return type is INTEGER.

Argument	Meaning
Handle	The start of the memory block. Type LONG.
FileName	The name of the file to be created or appended. Type STR. If executed in a WebVue session this instruction is processed by the computer that hosts the web back end and the file will be created on that computer.
BufferMode	Always "USEFULL_PART". Type STR.
FileMode	A string indicating the way in which the target file is opened. Type STR.

Execution

Argument	Meaning
FileMode	WRITE or omitted: The file is created and characters from the memory buffer are written to it until the first null character is encountered in the buffer. If the file already exists it will be overwritten. APPEND: The file is opened and characters from the memory buffer are appended

to the end until the first null character is encountered in the buffer.

Return: 1 if successful, else 0.

All the file management functions operate in the project folder TP.

Example

```
DIM Handle As Long;
Handle = Alloc_Buffer(50);
Put_Buffer (Handle,0, "Hello World");
Buftofile(Handle,"GREET.TXT");
Free_Buffer(Handle);
```

For further examples, select the Example link above.

C

CAPTION

See Also

Display text in the title bar of the workspace of the desktop client..

WebVue support - Not applicable. Returns the unsuccessful code if used in this context.

Syntax

StrVal = CAPTION(*NewTitle*);

The return type is STRING.

Argument	Meaning
NewTitle	The new caption for the workspace title bar. Type STR.

Execution

CAPTION enables display of the character string passed as an argument in the title bar of the workspace. The option for display of the title bar must be enabled in the workspace preferences.

Return: 1 if successful, else 0.

Example

```
SUB Main()
DIM strCaption as Str;
DIM intResult as Str;

strCaption="Main Project 1";
intResult = CAPTION(strCaption);
END SUB
```

CGET_BUFFER

See Also Example

Read a number of characters located in a memory area.

WebVue support - Yes.

Syntax

StrVal = CGET_BUFFER(*Handle*, *Offset* [, *Chars*[, *Flag*]]);

The return type is STR.

Argument	Meaning
Handle	The location of the memory buffer. Type LONG.
Offset	The offset in bytes (1 byte per character) representing the point at which the read will start. Any numeric type.
Chars	The number of characters which are to be returned. If omitted one character will be returned. Any numeric type.
Flag	Conversion of the character string from ASCII to ANSI. A value of 1 disables conversion, 0 enables it.

Execution

Read a number of characters from the specified memory location and return them in a string.

The character string returned is converted to ANSI code for Windows unless *Flag* = 1.

Example

```
SUB Main()
DIM bufh as LONG;
CONST SIZE = 0;
CONST MODI = 4;
CONST ALLOC = 22;
bufh = ALLOC_BUFFER(ALLOC);
PRINT(FSTAT("myfile.dat", bufh));
PRINT("Size (bytes) :",IGET_BUFFER(bufh,SIZE));
PRINT("Date changed :",CGET_BUFFER(bufh,MODI));
FREE_BUFFER(bufh);
END SUB
```

For further examples, select the Example link above.

CHART

See Also Example

Manage the operation of an XY Chart in the HMI.

WebVue support - No.

Mode	Mnemonic	Syntax
1	AXE_SETRANGE	1
2	AXE_SETTITLE	2
3	LINE_CLEAR	3
4	LINE_START	4
5	LINE_END	4
6	PRINT	5
7	RANGE_ADDPOINT	6
8	RANGE_CLEAR	7
9	RANGE_REMOVEPOINT	8
10	REFRESH	9
11	SERIES_ADDPOINT	10
12	SERIES_CLEAR	11
13	SERIES_GETCOUNT	11
14	SERIES_GETTS	12
15	SERIES_GETXVALUE	12
16	SERIES_GETYVALUE	12
17	SERIES_IDXFROMX	13
18	SERIES_IDXFROMXY	14
19	SERIES_REMOVEPOINT	12
20	SERIES_SETVAR	15
21	SERIES_SHOW	16
22	SERIES_SORT	11
23	SERIES_STATS	17
24	SERIES_YCOUNTFROMX	18
25	SERIES_YVALUEFROMX	13
26	SETDATETIME	19
27	SETENDTIMEPERIOD	20
28	SETSTARTTIMEPERIOD	21
29	SETSAMPLEPERIOD	22
30	SETSOURCEVAR	23
31	SETTITLE	24
32	SERIES_TOFRONT	11
33	COPY_CLIPBOARD	9
34	SAVE_IMAGE	25
35	SHOW_LEGEND	26

Properties Common to all Modes

Argument	Meaning
Window	The name of the window containing the chart. Type STR.
Branch	The branch of the window containing the chart. Type STR.
Identifier	The idenitier of the chart (within the window). Type STR

Syntax 1

LongVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *Axis*, *Min, Max*);

Argument	Meaning
Axis	A number representing the axis that is to be modified. Type LONG. 1 = X

	2 = Y (Left) 3 = Y (Right)
Min	The new value for the axis minimum. Type DOUBLE.
Max	The new value for the axis maximum. Type DOUBLE.

Execution

Mode	Mnemonic	Action
1	AXE_SETRANGE	Set the axis range. If 0 is used for both max and min the range is set to automatic. Return: 1 if OK, 0 if NOK.

Syntax 2

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *Axis*, *Title*);

Argument	Meaning
Axis	A number representing the axis that is to be modified. Type LONG. 1 = X 2 = Y (Left) 3 = Y (Right)
Title	The new axis title. Type STR.

Execution

Mode	Mnemonic	Action
2	AXE_SETTITLE	Set the axis title. Return: 1 if OK, 0 if NOK.

Syntax 3

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *LineID*);

Argument	Meaning
LineID	The ID of the Line object. Either 1 or 2. Type LONG.

Execution

Mode	Mnemonic	Action
3	LINE_CLEAR	Clear the line object. Return: 1 if OK, 0 if NOK.

Syntax 4

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *LineID*, *X*, *Y*);

Argument	Meaning
LineID	The ID of the Line object. Either 1 or 2. Type LONG.
X, Y	The new X and Y co-ordinates. Type DOUBLE.

Execution

Mode	Mnemonic	Action
4	LINE_END	Set new co-ordinates for the end point of the line. Return: 1 if OK, 0 if NOK.
5	LINE_START	Set new co-ordinates for the start point of the line. Return: 1 if OK, 0 if NOK.

Syntax 5

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier* [, *Orientation* [, *Printer*]]);

Argument	Meaning

Orientation	The orientation of the printed output. Optional. Type LONG. 1 = Landscape 2 = Portrait
X, Y	The name of the printer to be used for output. Type STR.

Execution

Mode	Mnemonic	Action
6	PRINT	Print the Chart on the selected printer. Return: 1 if OK, 0 if NOK.

Syntax 6

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *RangeID*, *X, YLow, YHigh*);

Argument	Meaning
RangeID	The ID of the Range object Either 1 or 2. Type LONG.
X	The X value of the point. Type DOUBLE.
YLow	The Y low value of the point. Type DOUBLE.
YHigh	The Y high value of the point. Type DOUBLE.

Execution

Mode	Mnemonic	Action
7	RANGE_ADDPOINT	Add a point to the range object. Return: 1 if OK, 0 if NOK.

Syntax 7

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *RangeID*);

Argument	Meaning
RangeID	The ID of the Range object Either 1 or 2. Type LONG.

Execution

Mode	Mnemonic	Action
8	RANGE_CLEAR	Clear the range object. Return: 1 if OK, 0 if NOK.

Syntax 8

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *RangeID*, *Index*);

Argument	Meaning
RangeID	The ID of the Range object Either 1 or 2. Type LONG.
Index	A zero based index of the point to be removed. Type DOUBLE.

Execution

Mode	Mnemonic	Action
9	RANGE_REMOVEPOINT	Remove a point from the range object. Return: 1 if OK, 0 if NOK.

Syntax 9

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*);

Execution

Mode	Mnemonic	Action
10	REFRESH	Refresh the chart re-processing the historical request. Return: 1 if OK, 0 if NOK.

33	COPY_CLIPBOARD	Copy an image representing the chart to the clipboard so that it can be used in other applications (Word etc.). Return: 1 if OK, 0 if NOK.

Syntax 10

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID, X, Y* [, *TS*]);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
X	X value of the point. Type DOUBLE.
Y	Y value of the point. Type DOUBLE.
TS	Timestamp of the point. Optional. Type DOUBLE.

Execution

Mode	Mnemonic	Action
11	SERIES_ADDPOINT	Add a datapoint to the series. Return: 1 if OK, 0 if NOK.

Syntax 11

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.

Execution

Mode	Mnemonic	Action
12	SERIES_CLEAR	Clear the series. Return: 1 if OK, 0 if NOK.
13	SERIES_GETCOUNT	Return the number of points in the series. Return: The number of points in the series if no error, otherwise -1.
22	SERIES_SORT	Sort the series against the X-axis. Return: 1 if OK, 0 if NOK.
32	SERIES_TOFRONT	Move the series to the front of the chart. Return: 1 if OK, 0 if NOK.

Syntax 12

DblVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *Index*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
Index	A zero based index of the point. Type DOUBLE.

Execution

Mode	Mnemonic	Action
14	SERIES_GETTS	Get the timestamp of the point. Return: The timestamp value if OK, -1 if index is out of range, 0 for any other errors.
15	SERIES_GETXVALUE	Get the X value of the point. Return: The X value if OK, -1 if index is out of range, 0 for any other errors.
16	SERIES_GETYVALUE	Get the Y value of the point. Return: The Y value if OK, -1 if index is out of range, 0 for any other errors.
19	SERIES_REMOVEPOINT	Remove a data point from a series. Return: 1 if OK, -1 if index is out of range, 0 for any other errors.

Syntax 13

DblVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *X [, Rank]*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
X	Given X value. Type DOUBLE.
Rank	The rank of the data point having the X value. Optional. Type DOUBLE.

Execution

Mode	Mnemonic	Action
17	SERIES_IDXFROMX	Return: The index of the point if it exists, -1 if the point doesn't exist, 0 for all others errors. See also Further Information below.
25	SERIES_YVALUEFROMX	Return: The Y value if the point exists, -1 if the point doesn't exist, 0 for all other errors.

Syntax 14

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *X, Y [, IndexStart [,IndexEnd]]*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
X	Given X value. Type DOUBLE.
Y	Given Y value. Type DOUBLE.
IndexStart	The index from where to start the search. Type DOUBLE.
IndexEnd	The index at which to end the search. Type DOUBLE.

Execution

Mode	Mnemonic	Action
18	SERIES_IDXFROMXY	Return: The index of the point if it exists, -1 if the point doesn't exist, 0 for all others errors. See also Further Information below.

Syntax 15

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *VarName*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
VarName	The name of a Variables Tree variable. Type STR.

Execution

Mode	Mnemonic	Action
20	SERIES_SETVAR	Change the variable used for the series source. Return: 1 if OK, 0 if NOK.

Syntax 16

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *Show*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
Show	Flag to select show or hide. 1 = Show 0 = Hide

Execution

Mode	Mnemonic	Action
21	SERIES_SHOW	Show or hide the series. Return: 1 if OK, 0 if NOK.

Syntax 17

DblVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *Stats*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
Show	Flag to select which statistic to return. 1 = X minimum 2 = X maximum 3 = Y minimum 4 = Y maximum 5 = Average 6 = Median

Execution

Mode	Mnemonic	Action
23	SERIES_STATS	Return: The selected statistic.

Syntax 18

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SeriesID*, *Show*);

Argument	Meaning
SeriesID	The ID of the series. Range 1 to 8. Type LONG.
X	Given X value

Execution

Mode	Mnemonic	Action
25	SERIES_YCOUNTFROMX	Return: The number of points having the given X value if no error, else 0.

Syntax 19

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *StartTime [, EndTime]*);

Argument	Meaning
StartTime	Start time. Type DOUBLE.
EndTime	End time. Optional. Type DOUBLE.

Execution

Mode	Mnemonic	Action
26	SETDATETIME	Initiate a historical request from the start date to the end date. If the end time is omitted the current time is used. Return: 1 if OK, -1 if the resulting number of points is greater than maximum allowed, -2 for invalid time range, 0 for all other errors.

Syntax 20

LongVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *EndTime, Period [, TimeUnit]*);

Argument	Meaning
EndTime	End time. Type DOUBLE.
Period	Historic data period. Type DOUBLE.
TimeUnit	The time unit for the period. 0 = Millisconds

1 = Seconds
2 = Minutes
3 = Hours
4 = Days

Execution

Mode	Mnemonic	Action
27	SETENDTIMEPERIOD	Initiate a historical request with the Begin time set to EndTime minus the Period and the End time set to EndTime. Return: 1 if OK, -1 if the resulting number of points is greater than maximum allowed, -2 for invalid time range, 0 for all other errors. See also Further Information below.

Syntax 21

LongVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *StartTime, Period [, TimeUnit]*);

Argument	Meaning
StartTime	Start time. Type DOUBLE.
Period	Historic data period. Type DOUBLE.
TimeUnit	The time unit for the period. 0 = Millisconds 1 = Seconds 2 = Minutes 3 = Hours 4 = Days

Execution

Mode	Mnemonic	Action
28	SETSTARTTIMEPERIOD	Initiate a historical request with the Begin time set to StartTime and the End time set to the StartTime plus the Period. Return: 1 if OK, -1 if the resulting number of points is greater than maximum allowed, -2 for invalid time range, 0 for all other errors. See also Further Information below.

Syntax 22

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *SamplePeriod [, TimeUnit]*);

Argument	Meaning
SamplePeriod	Sample period. Type DOUBLE.
TimeUnit	The time unit for the period. 0 = Millisconds 1 = Seconds 2 = Minutes 3 = Hours 4 = Days

Execution

Mode	Mnemonic	Action
29	SETSAMPLEPERIOD	Set the sample period that is to be used for subsequent historical requests. Return: 1 if OK, -2 for invalid time range, 0 for all other errors.

Syntax 23

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *VarName*);

Argument	Meaning

VarName	The name of a Variables Tree variable. Type STR.

Execution

Mode	Mnemonic	Action
30	SETSOURCEVAR	Change the variable used for the X axis. Return: 1 if OK, 0 if NOK.

Syntax 24

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *Title*);

Argument	Meaning
Title	The title for the chart. Type STR.

Execution

Mode	Mnemonic	Action
31	SETTITLE	Change the title displayed in the chart. Return: 1 if OK, 0 if NOK.

Syntax 25

IntVal = CHART (*Mode*, *Window*, *Branch*, *Identifier*, *Path, ImageFormat*);

Argument	Meaning
Path	Either the full path for the image. For example E:\\ChartImages\\hangar18.png. Or just the image name in which case the image will be saved in the project's TP folder. Type STR.
ImageFormat	The image format. 0 = BMP 1 = GIF 2 = PNG

Execution

Mode	Mnemonic	Action
34	SAVE_IMAGE	Save an image of the chart to the specified location in the specified format. Return: 1 if OK, 0 if NOK.

Syntax 26

LongVal = Chart (*Mode, Window, Branch, Identity[, Show]*)

Argument	Meaning
Show	A flag. Type INTEGER. 0 = Hide the legend. Default setting. 1 = Show the legend.

Execution

Mode	Mnemonic	Action
35	SHOW_LEGEND	Show or hide the chart legend. Return: 1 if OK, 0 if NOK.

Further Information

Consider the series S0 which is comprised of the following points: P1(10,15), P2(11,20), P3(15,20), P4(18,25), P5(18,25), P6(21,100) ,P7(21,105).

SERIES_IDXFROMXY

SERIES_IDXFROMXY returns the index of the first point in the series with the given X and Y. So the code:

```
CHART ("SERIES_IDXFROMXY", cWindow, cBranch, cIdentifier, cSeriesID, 18, 25);
```

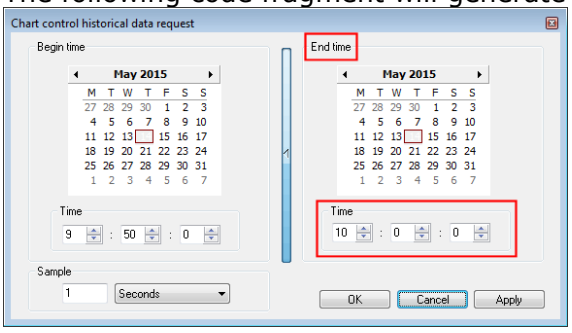
would return 4 representing P4(18,25). You can also specify an optional start point and end point for the search.

SERIES_IDXFROMX

SERIES_IDXFROMX is similar to SERIES_IDXFROMXY except that you only specify X. So for the series S0, if X = 15, it will return 3 representing P3(15,20). You can also specify an optional rank . For example, if you were looking for the second point with X = 21 you would specify a rank of 1 (rank of 0 is the first point). The return would be 7 representing P7(21,105).

SETENDTIMEPERIOD

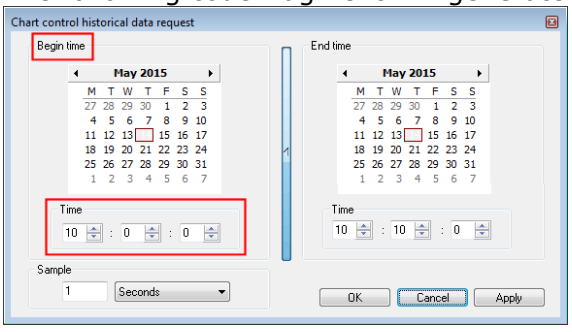
The following code fragment will generate the equivalent of this historical request. Show picture




```
Dim dTime As Double;
Dim lReturn As Long;
dTime = DateTimeValue("14/05/2015", "10:00:00");
lReturn = Chart("SETENDTIMEPERIOD", "MyChartMimic", "", "Chart1", dTime, 10, 2);
```

SETSTARTTIMEPERIOD

The following code fragment will generate the equivalent of this historical request. Show picture




```
Dim dTime As Double;
Dim lReturn As Long;
dTime = DateTimeValue("14/05/2015", "10:00:00");
lReturn = Chart("SETSTARTTIMEPERIOD", "MyChartMimic", "", "Chart1", dTime, 10, 2);
```

CHECKLIST

See Also Example

Access properties of the Supervisor's Check-box list form control.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	Webvue
1	COUNT	1	Yes
2	GETSELECTEDINDEX	1	Yes
3	SETSELECTEDINDEX	2	Yes
4	GETTEXT	4	Yes
5	GETUSERDATA	4	Yes
6	GETSTATE	4	Yes
7	SETSTATE	3	Yes
8	LOAD	5	Yes
9	GETEVENTTYPE	1	Yes
10	INSERT	6	No
11	REMOVE	7	No
12	SORT	8	No
13	CLEAR	1	No

An important note about the terminology used and operation of the Supervisor's Check List Control.

- An item that is SET is one in which the centre of the corresponding square is ticked. Pear
- The item that is SELECTED is the one whose text is highlighted (by changing its background colour

An item that is SET is not necessarily SELECTED (and vice versa). Programmatically you are normally on interested in which items are SET. The item that is SELECTED has little meaning.
For more information on the behaviour of the Check List Control see the Supervisor's main help.

All syntaxes

Argument	Meaning
Window	The name of the window that contains the form control. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR.
Identity	The identity of the form control in the specified window. Type STR.

Syntax 1

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*);

Execution

Mode	Mnemonic	Action
1	COUNT	Returns the number of items. Type LONG.
2	GETSELECTEDINDEX	Returns the index of the item currently selected. Type LONG.
9	GETEVENTTYPE	Returns the type of the most recent event. Type LONG. The return is binary weighted value. Bit 0: The selection has been changed. Bit 1: An item's state has been changed.
13	CLEAR	Clears the list contents and any selection. Return: 1 if successful, else 0.

Syntax 2

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*, *Notification*);

Argument	Meaning

Index	The index of the item to select. Type LONG. If the index is -1 then no item is selected.
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control. Type INTEGER.  For this to work, it also has to be enabled in the Operations tab of the Check-box List properties dialog. The function will scroll the control if necessary to display the item.

Execution

Mode	Mnemonic	Action
3	SETSELECTEDINDEX	Selects the item corresponding to the supplied index. Type LONG. Return: 1 if successful, else 0.

Syntax 3

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*, *State*,*Notification*);

Argument	Meaning
Index	The index of the item to select. Type LONG. If the index is -1 then no item is selected.
State	State to be set. The value can be 0 or 1. Type INTEGER
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control.  For this to work, it also has to be enabled in the Operations tab of the Check-box List properties dialog. Type INTEGER. The function will scroll the control if necessary to display the item.

Execution

Mode	Mnemonic	Action
7	SETSTATE	Sets the item corresponding to the supplied index an state. Type LONG. Return: 1 if successful, else 0.

Syntax 4

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of the item to be retrieved.

Execution

Mode	Mnemonic	Action
4	GETTEXT	Returns the Text of the item as defined for the language currently in use. Type ST
5	GETUSERDATA	Returns the User Data associated with the item. Type STR.
6	GETSTATE	Returns the state of an item (ticked or not). Type LONG.

Syntax 5

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*, *FileName*);

Argument	Meaning
FileName	The name of the file that contains the form control's data.

Execution

Mode	Mnemonic	Action
8	LOAD	Loads the file's content into the form control's list of items. Type INTEGER. Return: 1 if successful (file loaded), else 0.

Syntax 6

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*, *Text*, *Userdata*, [*Index*]);

Argument	Meaning
Text	The text to insert. Type STR.
Userdata	The user data to insert. Type STR.
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
10	INSERT	Inserts the Text and optional Userdata in the form control list. If Index is omitted text is inserted at the start. If Index is -1 the text is inserted at the end. Otherwis text is inserted at the position indicated by Index. Return: 1 if successful, else 0.

Syntax 7

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
11	REMOVE	Removes the selected item from the list. Return: 1 if successful, else 0.

Syntax 8

Return = CHECKLIST(*Mode*, *Window*, *Branch*, *Identity*[, *Sortorder*, *Orderby*]);

Argument	Meaning
Sortorder	A flag indication the sort order. 0 = ascending, 1 = descending
Orderby	A flag selecting which data to sort by. 0 = Tex, 1 = Userdata.

The return type is LONG.

Execution

Mode	Mnemonic	Action
12	SORT	Sort the form control list using the specified criteria. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

CHR

See Also

Return the character corresponding to ASCII code.

WebVue support - Yes.

Syntax

StrVal = CHR(*AsciiCode*);

The return type is STR.

Argument	Meaning
AsciiCode	The ASCII code which is to be converted. Any numeric type.

Execution

A character is returned corresponding to an ASCII code in the range 0 to 255. Outside of that range, modulus 256 is used.

Example

```
SUB Main()
DIM intCharCode as integer;
DIM strChar as Str;

'ASCII code - Integer type
intCharCode = 97;
strChar=Chr(intCharCode);
PRINT ("ASCII code 97 corresponds to:", strChar);
'Displays "ASCII code 97 corresponds to: a"
END SUB
```

CIMWAY

See Also Example Communication Object Parameters

Stop, start and test the state of the communications.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	OFF	1
1	ON	1
2	STATCIM	1
3	SCANMDF	2
4	STATUS	3
5	ERROR	4
7	EXIST	3
8	CFG	6
9	START	3
10	STOP	3
11	READAFTERWRITE	7
12	SCANALWAYS	1
13	WRITEGROUP	8
14	OVERWRITEFILE	9
15	SENDMSG	12
17	SEQUENCINGWRITE	1, 10
18	GETPENDINGWRITE	1
19	READFRAME	11

Syntax 1

IntVal = CIMWAY (*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	OFF	Stop the communication manager. Return: 1 if successful, else 0.
1	ON	Start the communication manager. Return: 1 if successful, else 0.
2	STATCIM	Return the state of the communication manager. Return: 1 if running, else 0.
12	SCANALWAYS	Force all cyclic frames to be scanned even if they are not connected to any variables. Return: 1 if successful, else 0.
17	SEQUENCINGWRITE	Enable or disable the sequential mode of recording. Return: 1 if successful, else 0
18	GETPENDINGWRITE	Obtain the number of entries in the communication manager's queue.* Return: number of entries.

*GETPENDINGWRITE returns a LONG.

Syntax 2

IntVal = CIMWAY(*Mode*, *Network.Node.Frame*, *Period*);

The return type is INTEGER.

Argument	Meaning
Period	The new period in seconds at which the specified frame is to be scanned. The maximum number is 1600. Any numeric type.

Execution

Mode	Mnemonic	Action
3	SCANMDF	Modify the scan period of a communication frame. The behaviour caused by the Period argument is as follows. >0 Use that period. 0 Disable scanning. -1 Use the previous period only if the current period is 0 or after a CIMWAY("STOP", "frame object"). Else nothing happens. -2 Use the scan period from the frame configuration. Return: 1 if the frame exists, else 0.

Example

Using a configured frame scan period of 20 seconds.

Period	Actual scan period
5	5 seconds
-2	20 seconds
10	10 seconds
0	Stop scanning
-1	10 seconds
5	5 seconds
-1	5 seconds

Syntax 3

IntVal = CIMWAY(*Mode*, *ComObj*);

The return type is INTEGER.

Argument	Meaning
ComObj	The communication object. It may be a network, a node or a frame. Network: *Network name* Node: *Network_name.Node_name* Frame: *Network_name.Node_name.Frame_name*

Execution

Mode	Mnemonic	Action
4	STATUS	Returns the current status of the communication object. Return: 1 if OK, 0 if failed, -1 if it does not exist.
7	EXIST	Test if a communication object exists. Return: 1 if it exists, else 0.
9	START	Enable the communication object. Return: 1 if OK, 0 if syntax error or the object is unknown.

10	STOP	Disable the communication object. Return: 1 if OK, 0 if syntax error or the object is unknown.

Variables linked to the communication object are not flagged as invalid.

Syntax 4

LongVal = CIMWAY(*Mode*, *ComObj*);

Argument	Meaning
ComObj	The communication object. It may be a network, a node or a frame. Network: *Network name* Node: *Network_name.Node_name* Frame: *Network_name.Node_name.Frame_name*

The return type is LONG.

Execution

Mode	Mnemonic	Action
5	ERROR	Return the error counter for the specified communication object. Return: The value of the error counter.

The error counter represents the number of failed requests.

Syntax 6

IntVal = CIMWAY(*Mode*, *ComObj*, *Modif*, *Param*, [, *VarName*]);

The return type is INTEGER.

Argument	Meaning
ComObj	The communication object: a network, a node or a frame.
Modif	The communication object you want to modify: PORT_NUMBER Modification of the port number. EQT_ADDRESS Modification of the equipment address. MEMORY_ADDRESS Modification of the address of the frame.
Param	This parameter field has different meanings depending on the value of *Modif*. See the topic Communication Object Parameters for their settings. Type STR.
VarName	The name of a database register. Type STR.

Execution

Mode	Mnemonic	Action
8	CFG	Modify the parameters of a communication object. The status of the modification can be returned in the optional register *VarName*: 0 = REQUEST 1 = ACK 2 = NACK Return: 1 if OK, 0 if the communication object is unknown or if the parameters are incorrect.

Syntax 7

IntVal = CIMWAY(*Mode*, *network_name.Equipment_name.Frame_name*, *RM*);

The return type is INTEGER.

Argument	Meaning
RM	The read after write mode: 1 - YES 2 - NO

Execution

Mode	Mnemonic	Action
11	READAFTERWRITE	Control read after every write. Return: 1 if OK, 0 if the frame is unknown or syntax error.

Syntax 8

IntVal = CIMWAY(*Mode*, *network_name.Equipment_name.Frame_name*, *Delay*, *Mode*, *Flag*);

The return type is INTEGER.

Argument	Meaning
Delay	A period in seconds.
Mode	A flag corresponding to the mode in the instruction SENDLIST: 0 for block mode. 1 for multiple mode.
Flag	Either 1 or 0.

Execution

Mode	Mnemonic	Action
13	WRITEGROUP	Optimise processing of write frames by CIMWAY. Normally a write frame is processed when any of the variables linked to it change. This instruction causes changes to be buffered and the frame processed once every *Delay* number of seconds. If the flag is set to 1 then all changes of a variable are sent. If the flag is 0 then changes may be lost. For example if the delay is set to 5 and a variable changes from 1 to 0 and then back to 1 again within the processing period, the change will not be seen. Return: 1 if OK, 0 if the frame is unknown or there is a syntax error.

If you enter a fraction for *Delay*, it will be rounded down to a whole number (e.g. 0.5 will become 0 and 1.8 will become 1).

Syntax 9

IntVal = CIMWAY (*Mode*, *FileName*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
14	OVERWRITEFILE	Replaces the communication configuration file. The file *FileName* is renamed to COMM.DAT and copied to the C and TEMP folders of the project.

Return: 0 if successful, else:

-1 if the file to be imported is not found
-2 is the file name is for a folder
-3 if writing to COMM.DAT is not possible (write permissions etc.)
-4 if writing to CTEMP\COMM.DAT is not possible (write permissions etc.)

The original COMM.DAT file is discarded.

Syntax 10

IntVal = CIMWAY (*Mode*, *Sequence/not*);

The return type is INTEGER.

Argument	Meaning
Sequence/not	"YES" - enable. "NO" - disable.

Execution

Mode	Mnemonic	Action
17	SEQUENCINGWRITE	Enable or disable the sequential mode of recording. Return: 1 if successful, else 0.

Syntax 11

IntVal = CIMWAY (*Mode*, *Frame, VarName*);

The return type is INTEGER.

Argument	Meaning
Frame	To identify the frame, e.g. "NETWORK.EQUIPMENT.FRAME1"
VarName	Name of variable in which the status of the reading is to be provided.

Execution

Mode	Mnemonic	Action
19	READFRAME	To request updating of the frame and obtain the status of the reading via the variable. Return: 1 if successful, else: 0 if the arguments are incorrect. 2 if the variable is already in use by another READFRAME request. States indicated in the variable named *VarName*: 0 Request in progress. 1 Reading was successful: variables updated. 2 Reading error: the variables are incorrect.

Syntax 12

IntVal = CIMWAY (*Mode*, *ObjectId*, *CommandString*);

The return type is INTEGER.

Argument	Meaning

ObjectId Identifies the target object identifier, for access to:

All networks.

One network: *NetworkId*.
One item of equipment: *NetworkID.EquipmentId*.
One frame: *NetworkID.EquipmentId.FrameId*.

Type STR.

CommandString Free text with contents specific to the protocol that will receive it. Type STR.

Execution

Mode	Mnemonic	Action
15	SENDMSG	Sends a message to communication objects. Return: 1 if successful (communication object took account of the message), else 0.

CIMWAY does not interpret the content.

Examples

To activate the sequential mode of recording:

```
CIMWAY("SEQUENCINGWRITE", "YES");
```

To obtain the number of entries in the queue:

```
IntVal = CIMWAY ("GETPENDINGWRITE");
```

To update the communication frame:

```
CIMWAY("READFRAME","NETWORK.EQT1.WORD","RESULT");
```

For a larger example, select the Example link above.

CMPSTRING

See Also Example

Comparison of two character strings.

WebVue support - Yes.

Syntax

IntVal = CMPSTRING(*string1*, *string2*);

The return type is INTEGER.

Execution

The two strings are compared using the ASCII value of the characters. If the string is longer than 1 character, then each character is compared in turn until an inequality is found.

Return	Meaning
-1	*string1* is less then string2.
0	The two strings are equal.
1	*string1* is greater than string2.

Example

```
DIM Chain1 As Str, Chain2 As Str;
DIM Chain3 As Str, Chain4 As Str;
DIM Res As Integer;
Chain1 = "A";
Chain2 = "B";
Chain3 = "ABCDEG";
Chain4 = "ABDDEF";
Res = CmpString(Chain1, Chain1); ' Res = 0
Res = CmpString(Chain1, Chain2); ' Res = -1
Res = CmpString(Chain2, Chain1); ' Res = 1
Res = CmpString(Chain3, Chain4); ' Res = -1
```

For further examples, select the Example link above.

COMBOBOX

See Also Example

Access some properties of the Supervisor's Combo-box form control.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	WebVue
1	COUNT	1	Yes
2	GETSELECTEDINDEX	1	Yes
3	SETSELECTEDINDEX	2	Yes
4	GETTEXT	3	Yes
5	GETUSERDATA	3	Yes
6	LOAD	4	Yes
7	INSERT	5	No
8	REMOVE	6	No
9	SORT	7	No
10	CLEAR	1	No

All syntaxes

Argument	Meaning
Window	The name of the window that contains the form control. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR.
Identity	The identity of the form control within the specified window. Type STR.

Syntax 1

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*);

The return type is LONG.

Execution

Mode	Mnemonic	Action
1	COUNT	Returns the number of items. Type LONG.
2	GETSELECTEDINDEX	Returns the index of the item currently selected. Type LONG.
10	CLEAR	Clears the list contents and any selection. Return: 1 if successful, else 0.

Syntax 2

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*, *Index*, *Notification*);

Argument	Meaning
Index	The index of the item to select. Type LONG. If the index is -1 then no item is selected.
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control. Type INTEGER.

The return type is type LONG.

Execution

Mode	Mnemonic	Action
3	SETSELECTEDINDEX	Select an item according to its index. Type LONG. Return: 1 if successful, else 0.

Syntax 3

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of the item to be retrieved.

The return type is STR.

Execution

Mode	Mnemonic	Action
4	GETTEXT	Returns the Text of the item as defined for the language currently in use. Type ST
5	GETUSERDATA	Returns the User Data associated with the item. Type STR.

Syntax 4

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*, *FileName*);

Argument	Meaning
FileName	The name of the file that contains the form control's data.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	LOAD	Loads the file's content into the form control's list of items. Type INTEGER. Return: 1 if successful (file loaded), else 0.

Syntax 5

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*, *Text*, *Userdata*, [*Index*]);

Argument	Meaning
Text	The text to insert. Type STR.
Userdata	The user data to insert. Type STR.
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
7	INSERT	Inserts the Text and optional Userdata in the form control list. If Index is omitted text is inserted at the start. If Index is -1 the text is inserted at the end. Otherwis text is inserted at the position indicated by Index. Return: 1 if successful, else 0.

Syntax 6

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action

8	REMOVE	Removes the selected item from the list. Return: 1 if successful, else 0.

Syntax 7

Return = COMBOBOX(*Mode*, *Window*, *Branch*, *Identity*[, *Sortorder*, *Orderby*]);

Argument	Meaning
Sortorder	A flag indication the sort order. 0 = ascending, 1 = descending
Orderby	A flag selecting which data to sort by. 0 = Tex, 1 = Userdata.

The return type is LONG.

Execution

Mode	Mnemonic	Action
9	SORT	Sort the form control list using the specified criteria. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

CONST

See Also

Declaration of a constant.

WebVue support - Yes.

Constants must always be declared in the header block of the program (outside of the functions).

Syntax

CONST *ConstName* = *Value*;

Argument	Meaning
ConstName	The name of the constant.
Value	The value which the constant represents.

Execution

Constants may only be numeric. Unnamed numeric values are also known as constants.

Example

```
'Declare constants
Const constant1 = 10;
Const constant2 = 10;

SUB Main()
DIM intparameter as integer;
DIM intresult as integer;

intparameter= 15;
intresult = (intparameter + constant1)/ constant2;
PRINT("Result is: ",intresult);
'Display: Result is: 2
END SUB
```

CONVERT

See Also Example

Convert an alphanumeric string to Hexadecimal, Octal, Binary or BCD; and vice-versa.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	BINTOA	1
2	OCTTOA	1
3	HEXTOA	1
4	BCDTOA	1
11	ATOBIN	2
12	ATOOCT	2
13	ATOHEX	2
14	ATOBCD	2

Syntax 1

StrVal = CONVERT(*Mode*, *Val*);

Argument	Meaning
Val	Numerical value to be converted. Any numeric type.

The return type is STR.

Execution

Mode	Mnemonic	Action
1	BINTOA	Convert a numerical value (base 10) into a character string representing the binary value.
2	OCTTOA	Convert a numerical value (base 10) into a character string representing the octal value.
3	HEXTOA	Convert a numerical value (base 10) into a character string representing the hexadecimal value.
4	BCDTOA	Convert a numerical value (base 10) into a character string representing the BCD value.

Syntax 2

LongVal = CONVERT(*Mode*, *Val*);

Argument	Meaning
Val	The string to be converted. Type STR.

The return type is LONG

Execution

Mode	Mnemonic	Action
		Return a numerical value (base 10) of a character string represented in:
11	ATOBIN	- its binary form.
12	ATOOCT	- its octal form.
13	ATOHEX	- its hexadecimal form.

14 ATOBCD - its BCD form.

Example

```
SUB main()
DIM ch1 as str;
ch1 = "01111111111111111111111111111111";
print( convert("ATOBIN",ch1)); 'Return: 2147483647
ch1 = "A9F7";
print( convert("ATOHEX",ch1)); 'Return: 43511
ch1 = "24";
print(convert("BINTOA",convert("ATOBCD",ch1) ));
'Return: 100100
END SUB
```

For further examples, select the Example link above.

COPY_BUFFER

See Also Example

Copy the contents of a buffer into another buffer.

WebVue support - Yes.

Syntax

IntVal=COPY_BUFFER (*Target*, *Source*[, *OffsetTarget*]);

Argument	Meaning
Target	The destination handle allocated by ALLOC_BUFFER. Type LONG.
Source	The source handle allocated by ALLOC_BUFFER. Type LONG.
OffsetTarget	The specified offset of the destination handle. Default value = 0. Any numeric type.

The return type is INTEGER.

Execution

Execution

Return	Meaning
IntVal	1 if OK -1 if the buffer *Target* has not been allocated -2 if the buffer *Source* has not been allocated -3 if the size of the buffer *Target* is less than the size of the buffer whose handle is at *Source* + *OffsetTarget*

Example

For an example, select the Example link above.

COS

See Also

Cosine function.

WebVue support - Yes.

Syntax

DblVal = COS(Angle);

Argument	Meaning
Angle	The angle in degrees of which the cosine is to be taken. Any numeric type.

The return type is DOUBLE.

Execution

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
'Variables
'@ARC - Register type
'@ANGLE - Register type

SUB Main()
'Declare
DIM dblangle as double;
DIM dblarc as double;

dblangle = 45;
dblarc = COS(dblangle);
print("COS(",dblangle,") = ",dblarc);
END SUB

SUB Proc2()
'COS() used in a variable function
'using TOS to convert DOUBLE to SINGLE (Register variable)
@ARC= TOS( COS (@ANGLE) ) ;
END SUB
```

CRONTAB

See Also Example

Create or modify a scheduled event.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	ADDPROG	1
2	DELPROG	1
3	ADDVAR	2
4	DELVAR	2
6	DELALL	3
7	TABLE_STATUS	4
8	TABLE_ELAPSED	4
9	TABLE_REMAINING	4
11	NETWORKBROADCAST	5
12	SAVE	3
13	TRACE	6

Syntax 1

IntVal = CRONTAB(*Mode*, *CRONTYPE*, *Date*, *Time*, *Program*, *Branch*, *Function* [, *Farg* [, *ActivBit*]]);

Argument	Meaning
CRONTYPE	The agenda type of the event:
1	ONCE
2	EACH_HOUR
3	EACH_DAY
4	EACH_WEEK
5	EACH_MONTH
Date	A string representing either the date or the day according to the type of event as specified by CRONTYPE. A date is specified in either dd/mm/yy or dd/mm/yyyy format, for example 05/08/2003. If specifying a day, the name of the day is given. "monday" or "mon" "tuesday" or "tue" "wednesday" or "wed" "thursday" or "thu" "saturday" or "sat" "sunday" or "sun"
Time	Represents the time (hh:mm) or the minute (mm) according to the type of event CRONTYPE. "17:38" or "38". Type STR.
Program	The name of the program that contains the function to be executed. Type STR.
Branch	The branch for the program. Type STR.
Function	The name of the function to be executed. Type STR.
Farg	Optional. A string of 2,047 characters maximum used to provide up to 8 arguments (separated by a comma) to the function.

	The arguments are retrieved in the function using the verb GETARG.
ActivBit	Optional. The name of the bit or alarm variable which enables the event.

Execution

Mode	Mnemonic	Action
1	ADDPROG	Create an event, or modify it if it exists, to run a program. The created event is temporary. It is not possible to modify a permanent event created from CONFIGURE.Actions.Scheduler. Return: 1 if the event has been correctly created, else 0.
2	DELPROG	Delete an event, which runs a program. It is not possible to delete a permanent event. Return: 1 if the event has been successfully deleted, else 0.


The program must be pre-loaded. The program must be pre-loaded. A cyclic function may not be changed once it is running. It must be stopped and re-programmed.


The use of DELAY within a function tat is called cyclically may produce unpredictable results and should be avoided.


The format of the date when using the mode ONCE can be either *dd/mm/yy* or *dd/mm/yyyy*.

Syntax 2

IntVal =CRONTAB(*Mode*, *TYPE*, *Date*, *Time*, *Variable*, *Send_Mode*, *Delay* [, "", *ActivBit*]);

Argument	Meaning
TYPE	The agenda type for the event: ONCE 1 EACH_HOUR 2 EACH_DAY 3 EACH_WEEK 4 EACH_MONTH 5
Date	A string representing either the date or the day according to the type of event as specified by TYPE. A date is specified in either dd/mm/yy or dd/mm/yyyy format, for example 05/08/2003. If specifying a day, the name of the day is given. "monday" or "mon" "tuesday" or "tue" "wednesday" or "wed" "thursday" or "thu" "saturday" or "sat" "sunday" or "sun"
Time	Represents the time (hh:mm) or the minute (mm) according to event *TYPE*. "17:38" or "38". Type STR.
Variable	The full name of the variable to be forced (Bit or alarm only).
S_Mode	0: set to 0. 1: set to 1.

2: reverse state.

Delay	If not 0, the variable is forced as a pulse. The duration is specified in seconds.
""	Unused parameter. Must be included if using the optional ActivBit parameter. Type STR.
ActivBit	Optional. The name of the bit or alarm variable which enables the event.

The return type is INTEGER

Execution

Mode	Mnemonic	Action
3	ADDVAR	Create (or modify if existing) an event which forces a variable. The created event is temporary. It is not possible to modify a permanent event. Return: 1 if the event has been correctly created, else 0.
4	DELVAR	Delete an event which forces a variable. It is not possible to delete a permanent event. Return: 1 if the event has been successfully deleted, else 0.

Syntax 3

IntVal = CRONTAB(*Mode*);

Execution

Mode	Mnemonic	Action
6	DELALL	Delete all the temporary events. Return: 1 if successfully deleted, else 0.
12	SAVE	Forces a save of the Scheduler configuration in the file CRON.DAT. Return: always 0

Syntax 4

LongVal = CRONTAB(*Mode*, *TableName*);

Argument	Meaning
TableName	The name of a timetable as created from Scheduler option of the ACTIONS sub-menu. Type STR.

Execution

Mode	Mnemonic	Action
7	TABLE_STATUS	Tests whether any of the timetable's periods is currently active, i.e. a period has started but not finished. Return: 1 if the time is included, 0 if it is not and -1 if the timetable does not exist.
8	TABLE_ELAPSED	Returns the number of seconds since the previous timetable transition. A transition is either the start or end of a period. Return: -1 if the timetable does not exist or the calculation is impossible.
9	TABLE_REMAINING	Returns the number of seconds until the next timetable transition. A transition is either the start or end of a period. Return: -1 if the timetable does not exist or the calculation is impossible.

Syntax 5

IntVal =CRONTAB(*Mode*, *List*, *HStatus*, *EventVar*, *LocalRemote*);

Argument	Meaning
List	The name of the list containing the names of the network stations to receive modifications (as created from the command Configure.Communication.Network Stations). Type STR.
HStatus	The handle of a buffer allocated by ALLOC_BUFFER. After the instruction has been executed the buffer will contain a list of stations and their update status in the format: *StaNum*,*Status*,...; where *StaNum* is the number of the station and *Status* is 1 if successful, else 0. Type LONG.
EventVar	The name of a bit variable that will be set to 1 when the instruction has been executed. Type STR.
LocalRemote	To determine where the changed Scheduler configuration file CRON.DAT is applied: 1: All stations (including the local station). 0: Remote stations only. (Default)

Execution

Mode	Mnemonic	Action
11	NETWORKBROADCAST	Sends the Scheduler configuration file CRON.DAT to all stations in the modification clients list. Return: 1 successful, else 0.

Syntax 6

IntVal =CRONTAB(*Mode*, *SubMode*);

Argument	Meaning
SubMode	Type STR. Either "ON" or "OFF".

Execution

Mode	Mnemonic	Action
1	TRACE	Activate ("ON") or deactivate ("OFF") trace messages for the Scheduler. Trace messages appear in the Event window (F7). Return: always 1.

Example

For an example, select the Example link above.

CYCLIC

See Also Example

Cyclic execution of a function.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	ADD	3
1	ADDPROG	1
2	ADDPROG_EX	1
5	DEL	1, 3
6	DELALL	2
7	DEL_EX	1
8	ADDPROGBYID	4

From version 8.2 of the software, the instruction ADDPROG is identical to ADD.

Syntax 1

IntVal = CYCLIC(*Mode*, *Delay*, *ProgName*, *Branch*, *Function* [, *Farg* [, *ActivBit*]]) ;

The return type is INTEGER.

Argument	Meaning
Delay	The delay in seconds between each activation of the program. Type INTEGER. The maximum delay is 2 x 10^6 or about 20 days.
ProgName	The name of the program that contains the function. Type STR.
Branch	The branch to pass to the called function. Type STR. A null string may be used to indicate no branch.
Function	The function which is to be run from the specified program. Type STR. A null string may be used to indicate no function (i.e. just to run the program).
Farg	Optional. A string of 2,047 characters maximum used to pass from 1 to 8 arguments (separated by a comma ",") to the function. The arguments can be retrieved from the function using the verb GETARG.
Activbit	Optional. The name of the bit or alarm variable, if there is one, that enables the event. Type STR.

Execution

Mode	Mnemonic	Action
1	ADDPROG	Add a cyclic action for a function. You can create several cyclic action for the same function with different values for delay. If an existing cyclic action has the same values for *Delay*, *ProgName*, *Branch* and*Function*, it is overwritten. If the name of a function is not specified then the MAIN function is executed. Return: 1 if successful, else 0.
2	ADDPROG_EX	Add a cyclic action for a function. Like mode 1, except that the fields *Farg* and *ActivBit* can be used to distinguish between cyclic actions that are otherwise identical. It is possible to create more than one cyclic action for the same function as long as one of the properties *Delay*, *Branch*, *Farg* or *ActivBit* are different.
5	DEL	Delete an existing cyclic action for a function.

		The value of the fields *Delay*, *ProgName*, *Branch* and *Function* must match the original values. If the delay is 0 then all existing cycles for the function are deleted. If *Farg* and *ActivBit* are not specified then all cycles of the function are deleted. If *Farg* or *ActivBit* is empty then all cycles of the function are deleted whatever the value of the argument. Return: 1 if successful, else 0.
7	DEL_EX	Delete an existing cyclic action for a function. If called with *Delay* = 0, then deletes all cyclic functions calling the function *Function* in program *ProgName* with branch *Branch* (behaves the same as the DEL mode by ignoring the *Farg* and *ActivBit* passed as parameters at the time of creation of the cyclic). If called with a *Delay* different from 0 then it only deletes the cyclic function calling the function *Function* in program *ProgName* with branch *Branch* with the *Delay*, *Farg* and *ActivBit* passed as arguments. If no such cyclic function exists, an error is returned (IntVal = 0). Return: 1 if successful, else 0.

The program must be pre-loaded (PRELOAD) to be run. It must not contain a SUB MAIN.

The use of DELAY within a function called cyclically may produce unpredictable results and should be avoided.

Syntax 2

IntVal = CYCLIC(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	DELALL	Delete all existing cyclic functions. Return: Always 1.

The program must be pre-loaded.

A cyclic action cannot be changed once it is running. It must be stopped and re-programmed.

The use of DELAY within a function called cyclically may produce unpredictable results and should be avoided.

Syntax 3

IntVal = CYCLIC(*Mode*, *Delay*, *ProgName*, *Branch*, *Function, Farg, ActivBit*) ;

The return type is INTEGER.

Argument	Meaning
Delay	The delay in seconds between each activation of the program. Type INTEGER. The maximum delay is 2 x 10^6 or about 20 days.
ProgName	The name of the program that contains the function. Type STR.

Branch	The database branch to pass to the called function. Type STR. A null string may be used to indicate no branch.
Function	The function which is to be run from the specified program. Type STR. A null string may be used to indicate no function (i.e. just to run the program).
Farg	A string of 2,047 characters maximum used to pass from 1 to 8 arguments (separated by a comma ",") to the function. The arguments can be retrieved from the function using the verb GETARG.
Activbit	The name of the bit or alarm variable, if there is one, that enables the event. Type STR.

Execution

Mode	Mnemonic	Action
0	ADD	Add a cyclic action for a function.
5	DEL	Delete an existing cyclic action for a function. The values of the fields must match the original values, as in Syntax 1 above. Return: 1 if successful, else 0.

From version 8.2 onwards modes ADD and ADDPROG became the same.

Syntax 4

IntVal = CYCLIC(*Mode*, *Name*, *Description*, *ServerListName*, *Delay*, *ProgName*, *Branch*, *Function*, [*Farg*[, *FactiveBit*]]);

The return type is INTEGER.

Argument	Meaning
Name	The ID of the cyclic action which will appear in the Name field in the Application Explorer. Type STR.
Description	A description of the cyclic action which will appear in the Description field in the Application Explorer. Type STR.
ServerListName	The name of a Server List from the network configuration. The cyclic will only be active if the station appears in the list. Can be a null string in which case the cyclic will be active on all stations. Type STR.
Delay	The delay in seconds between each activation of the program. Type INTEGER. The maximum delay is 2×10^6 or about 20 days.
ProgName	The name of the program that contains the function. Type STR.
Branch	The database branch to pass to the called function. Type STR. A null string may be used to indicate no branch.
Function	The function which is to be run from the specified program. Type STR. A null string may be used to indicate no function (i.e. just to run the program).
Farg	A string of 2,047 characters maximum used to pass from 1 to 8 arguments (separated by a comma ",") to the function. The arguments can be retrieved from the function using the verb GETARG.
Activbit	The name of the bit or alarm variable, if there is one, that enables the event. Type STR.

Execution

Mode	Mnemonic	Action
8	ADDPROGBYID	Add a cyclic action for a function using the supplied name, description and server list.

Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

D

DATETIME

See Also Example

Return the components of a date and time string.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	DAY	1
2	MONTH	1
3	YEAR	1
4	HOUR	1
5	MINUTE	1
6	SECOND	1
7	MILLISECOND	1
8	WEEKDAY	1
9	YEARDAY	1
10	ISLEAPYEAR	1

Syntax 1

IntVal = DATETIME(*Mode* [, *DateTime*]);

The return type is INTEGER.

Argument	Meaning
DateTime	The date and time on which to operate. If it is not supplied the current date and time is used. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and, if *DateTime* is omitted, the current date and time from the web back end is used.

Execution

Mode	Mnemonic	Action
1	DAY	Returns the day of the month (1 to 31).
2	MONTH	Returns the month number (1 to 12).
3	YEAR	Returns the year (1980 to 2106).
4	HOUR	Returns the hour (0 to 23).
5	MINUTE	Returns the number of minutes (0 to 59)
6	SECOND	Returns the number of seconds (0 to 59).
7	MILLISECOND	Returns the number of milliseconds (0 to 999).
8	WEEKDAY	Returns the day of the week. (0 to 6: Sunday = 0, Monday = 1 etc.)
9	YEARDAY	Returns the day of the year (1 to 366).
10	ISLEAPYEAR	Returns 1 if a leap year, else 0.

Example

For an example, select the Example link above.

DATETIMESTRING

See Also

Convert a time and date (stored as a double) to a chain of characters.

WebVue support - Yes.

This instruction will not work with values equivalent to dates earlier than 01/01/1980. The starting time is 00:00:00.000 on that date.

Syntax

StrVal = DATETIMESTRING(*TimeDate* [, *Format*]);

The return type is STR.

Argument	Meaning
TimeDate	A time and date represented by the number of milliseconds since 1980. Type DOUBLE.
Format	A chain of characters representing the format of the string.
U	UTC format (equivalent to ###Y-#M-#DT#h:#m:#s:##lZ)
#D	Day
#M	Month
#Y	Year as two characters (Example 93)
###Y	Year as four characters (Example 1994)
#h	Hours
#m	Minutes
#s	Seconds
##l	Milliseconds
D	Use the format #D/#M/#Y
T	Use the format #h:#m:#s
(null)	Use a default of #D/#M/#Y #h:#m:#s

Execution

The time and/or date is/are returned as a string in the specified format.

Example

```
SUB Main
'Declare variables
DIM dbldatetime as double;
DIM strformatdefault as Str;

dbldatetime = DATETIMEVALUE ( );
'Retrieve the current date & time
strformatdefault=DATETIMESTRING(dbldatetime,"D");
END SUB
```

DATETIMEVALUE

See Also

Return the number of milliseconds since the first of January 1970 at 00:00:00.000.

WebVue support - Yes.

Although this instruction returns the number of milliseconds since 1970, it will not work with dates earlier than 01/01/1980.

Condition	Syntax
Current time	1
Date and time as comma separated parameters	2
Date and time as formatted strings	3

Syntax 1

DblVal = DATETIMEVALUE();

The return type is DOUBLE.

Execution

Return the number of milliseconds at the time the function is executed. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the current date and time from the web back end is used.

Syntax 2

DblVal = DATETIMEVALUE(*Day*, *Month*, *Year*, *Hour*, *Minute*, *Second*[, *Millisecond*]);

The return type is DOUBLE.

Argument	Meaning
Day	Day of the month (1 to 31). Type INTEGER.
Month	Month number (1 to 12). Type INTEGER.
Year	Year (1980 to 2106). Type INTEGER.
Hour	Hour (0 to 23). Type INTEGER.
Minute	Minutes (0 to 59) Type INTEGER.
Second	Seconds (0 to 59). Type INTEGER.
Millisecond	Milliseconds (0 to 999). If not specified then the default of 0 is used. Type INTEGER.

Execution

Return the number of milliseconds for the specified date and time.

Syntax 3

DblVal = DATETIMEVALUE(*DD/MM/YY*, *HH:MM:SS*[:*Msc*]);

The return type is DOUBLE.

Argument	Meaning
DD/MM/YY	The date in *DD/MM/YY* or *DD/MM/YYYY* format. For example, 01/01/94 or 28/04/1984. Type STR.

HH:MM:SS The time in *HH:MM:SS* or *HH:MM:SS:Msc* format. For example 22:30:00 or 10:30:05:500. Type STR.

Execution

Return the number of milliseconds for the specified date and time.

Example

```
SUB Main()
'Declare variables
DIM dbldatetime as double;
DIM dbldatetime2 as double;
DIM dbldatetime3 as double;

dbldatetime = DATETIMEVALUE();
'Retrieve the current date & time
PRINT(dbldatetime);
dbldatetime2 = DATETIMEVALUE(4, 6, 2002, 22, 12, 12 , 654);
PRINT(dbldatetime2);
dbldatetime3 = DATETIMEVALUE("04/06/2002", "22:12:12:654");
PRINT(dbldatetime3);
END SUB
```

DDE

See Also Example

Manage DDE exchanges with a DDE server application not already configured in the Supervisor.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	INITIATE	1
2	TERMINATE	2
3	TERMINATEALL	3
4	TIMEOUT	4
5	EXECUTE	5
6	REQUEST	6
7	POKE	7

Syntax 1

LongVal = DDE (*Mode*, *ServiceName*, *Topicname*);

The return type is LONG.

Argument	Meaning
ServiceName	The name of the DDE server. Type STR
TopicName	The name of a DDE topic. Type STR

Execution

Mode	Mnemonic	Action
1	INITIATE	Start a DDE conversation with the named DDE server. The return contains a channel number identifying the conversation. Return: 0 if error, else the channel number of the conversation.

Syntax 2

IntVal = DDE (*Mode*, *Channel*);

The return type is INTEGER.

Argument	Meaning
Channel	The channel number of a DDE conversation as returned by DDE INITIATE. Type LONG

Execution

Mode	Mnemonic	Action
2	TERMINATE	Terminate a DDE conversation identified by the channel number. Return: 0 if error, else OK.

Syntax 3

IntVal = DDE (*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
3	TERMINATEALL	Terminate all DDE conversations. Return: 0 if error, else OK.

Syntax 4

IntVal = DDE (*Mode*, *Channel*, *Timeout*);

The return type is INTEGER.

Argument	Meaning
Channel	The channel number of a DDE conversation as returned by DDE initiate. Type LONG
Timeout	A period expressed in milliseconds. Type LONG

Execution

Mode	Mnemonic	Action
4	TIMEOUT	Modify the conversation timeout for the conversation identified by the channel number. Return: 0 if error, else OK.

Syntax 5

IntVal = DDE (*Mode*, *Channel*, *Command*);

The return type is INTEGER.

Argument	Meaning
Channel	The channel number of a DDE conversation as returned by DDE initiate. Type LONG
Command	A string containing an instruction to be sent to the DDE server. Type STR

Execution

Mode	Mnemonic	Action
5	Execute	Request the execution of a command by the DDE server. Return: 0 if error, else OK.

Syntax 6

IStrVal = DDE (*Mode*, *Channel*, *ItemName*);

The return type is STRING.

Argument	Meaning
Channel	The channel number of a DDE conversation as returned by DDE initiate. Type LONG
ItemName	A string containing the name of an item. Type STR

Execution

Mode	Mnemonic	Action
6	REQUEST	Request the value of a specific element of a conversation from a DDE server. Return: The value of the element.

Syntax 7

IStrVal = DDE (*Mode*, *Channel*, *ItemName, Data*);

The return type is INTEGER.

Argument	Meaning
Channel	The channel number of a DDE conversation as returned by DDE initiate. Type LONG
ItemName	A string containing the name of an item. Type STR
Data	A string containing a new value for the item. Type STR

Execution

Mode	Mnemonic	Action
7	POKE	Write the value of a specific element of a conversation from a DDE server. Return: 0 if error, else OK.

Example

For an example, select the Example link above.

DDECONV

See Also Example

Manage DDE exchanges with a DDE server application.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	START	1
1	STOP	1
2*	STATUS	1
3*	ON	2
4*	OFF	2

* These modes are now obsolete and have been replaced by system variables:

SYSTEM.DDE.*Conversation Name*.ON is a bit variable with a value of 1 when the conversation is running and 0 when it is stopped.

SYSTEM.DDE.*Conversation Name*.STATUS is a register variable containing the status of the conversation.

Syntax 1

Channel = DDECONV (*Mode*, *ConvName*);

The return type is INTEGER.

Argument	Meaning
ConvName	The name of the DDE conversation. Type STR

Execution

Mode	Mnemonic	Action
0	START	Start the DDE conversation. Return: 0 if successful 1 if error -1 if the conversation does not exist.
1	STOP	Stop the DDE conversation. Return: 0 if successful 1 if error -1 if the conversation does not exist.
2	STATUS	Test the status of the DDE conversation. Return: 0 if running 1 if stopped -1 if the conversation does not exist.

Syntax 2

IntVal = DDECONV (*Mode*);

The return type is INTEGER

Execution

Mode	Mnemonic	Action
3	ON	Start all DDE conversations. Return: 1 if successful, else 0.
4	OFF	Stop all DDE conversations. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

DECLARE FUNCTION / DECLARE SUB

See Also Example

Declaration of an external function or subroutine. This instruction is used whenever a DLL call is to be made.

WebVue support - Yes.

Mode	Mnemonic	Syntax
-	FUNCTION	1
-	SUB	2

Functions or subroutines must always be declared in the header block of the program.

Syntax 1

DECLARE FUNCTION *GlobalName* LIB *Libname* [ALIAS *AliasName*] ([*Arguments*]) AS Type;

Argument	Meaning
GlobalName	The name by which the SUB or FUNCTION will be known. If different to the real name of the SUB or FUNCTION in the DLL then the AliasNAme must also be specified.
LibName	The name of the DLL where the declared function is to be found. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end. The DLL must be present on the computer that hosts the web back end.
AliasName	The name of the function in the DLL if different to the name that it will be known by in the program (as specified by the GlobalName parameter). Type STR.
Type	The type returned by the function (INTEGER, LONG, SINGLE, DOUBLE or STR).
Arguments	An argument list with the following syntax: [BYVAL] Variable AS Type, [BYVAL] Variable AS Type, ...
Variable	The name of the variable passed as an argument.
Type	The type of the variable passed as an argument. INTEGER, LONG, SINGLE, DOUBLE.

Execution

The external program behaves as a function. It always returns a value.

Syntax 2

DECLARE SUB *GlobalName* LIB *LibName* [ALIAS *AliasName*] ([*Arguments*]);

Execution

The external program behaves as a subroutine. It does not return a value.

The declared sub-program is visible to all sub-programs of the current program. If it is declared in the GLOBAL program it is visible to all programs.

The folder in which the DLL resides must have a path set to it, or else the full path may be included in the declaration.

Example

```
DECLARE SUB DosBeep lib "DOSCALLS" (X as integer);
DECLARE FUNCTION WinExec LIB "KERNEL" (BYVAL File AS STR,BYVAL SH AS INTEGER) AS INTEGER;
```

DELAY

See Also

Suspends execution of the current program for the given period.

WebVue support - Yes.

Syntax

DELAY(*Period*);

There is no return type.

Argument	Meaning
Period	The number of seconds for the delay in the range 0.1 to 3277. Any numeric type.

Execution

The current program relinquishes control and requests execution after the given delay.

If the DELAY instruction is used in the GLOBAL program, it is the program that called GLOBAL which is suspended.

The use of the instruction DELAY is forbidden in functions that may run concurrently, in particular in:

- Functions called by CYCLIC, EVENT, CRONTAB, KEY or SELECTOR
- Send Program animations,
- Actions associated to alarms.

A DELAY is global to a program and suspends the execution of all its functions.

Example

```
SUB MainDelay()
DIM dblFrequency as double; 'in Hz
DIM dblPeriod as double;  ' in ms
DIM intSeconds as integer; ' seconds

dblFrequency = 1000;
dblPeriod = 100;
intSeconds = 4;
BEEP (dblFrequency,dblPeriod);
DELAY (intSeconds);
BEEP (dblFrequency,dblPeriod);
END SUB
```

DGET_BUFFER

See Also

Read a variable of type DOUBLE located in memory.

WebVue support - Yes.

Syntax

DblVal = DGET_BUFFER(*Handle*, *Offset*);

The return type is DOUBLE.

Argument	Meaning
Handle	The start address of the memory location. Type LONG.
Offset	The offset in bytes which is added to the Handle to provide the memory location for the variable to be retrieved. Any numeric type.

Execution

Recovers a variable of type DOUBLE located in memory. The start address of the location is normally provided by a previously executed ALLOC_BUFFER.

Example

```
i1 = DGET_BUFFER(handle,2);
```

DIM

See Also Example

Declaration of variables and variable arrays.

WebVue support - Yes.

Mode	Usage	Syntax
-	Variable	1
-	Array	2

Syntax 1

DIM *VarName As Type* [, *VarName2 As Type*, ...];

There is no return type.

Argument	Meaning
VarName	The name by which the variable is to be known.
Type	The type of variable (INTEGER, LONG, SINGLE, DOUBLE or STR).

Execution

A variable may be declared in the header block of either a function or program (outside the functions). If it is declared in a program header block it will be available to all functions. If it is declared within a function it will only be available to that function.

If a variable is to be substituted, the declaration must be made inside a Sub function, not globally in the program header nor passed as an argument from a Run Program animation. See example 1 below.

Syntax 2

DIM *VarName* [*Size1*] [*Size2*] *As Type* [, *VarName2 As Type*, ...];

There is no return type.

Argument	Meaning
Size1, Size2	The number of elements for each dimension.

Execution

An array variable may be declared in the header block of either a function or program (outside the functions). If it is declared in a program header block it will be available to all functions. If it is declared within a function it will only be available to that function.

The maximum number of dimensions for an array variable is 10. For example, TAB[i][j][k] is an array with three dimensions.
The maximum number of array elements in a particular function is 600 less the number of functions in the program. The array index starts at 0.

During iteration of an array the mimic display is frozen. Iterating a large array is very time consuming and should be avoided whenever possible.

Example 1

```
'Substitution - so the variable Vartext cannot be global

Sub main ()
Dim VarText as str;
Vartexte = "API_PP.RACK0_5.1.I0";
@INPUTTEXT=?Vartext;
```

```
End Sub
```

Example 2

```
SUB Main()
DIM i As integer;
DIM j As integer;
DIM strString1 As str;
DIM intArray1 [3][5] As integer;

'The table always starts at value 0;
strString1 ="1";
'Display strString1 value
PRINT (strString1);

FOR (i=0;i<3;i=i+1)
  FOR (j=0;j<5;j=j+1)
    intArray1[i][j]=1;
    'Display the table value
    PRINT (intArray1[i][j]);
  NEXT
NEXT
END SUB
```

For more examples, select the Example link above.

DVAL

See Also

Return the numeric value of a character string.

WebVue support - Yes.

Syntax

DblVal = DVAL(*string*);

The return type is DOUBLE.

Execution

If the string contains a character or character sequence that is recognised as non-numeric, zero is returned.

Example

```
SUB Main()
DIM dblResult as double;
DIM strString1 as Str;
strString1 = "125.35TEST";
dblResult = DVAL( strString1 );
PRINT("Result:", dblResult );
SUB Main
```

E

EMAIL

See Also Example Specifics

Manages simple email functions to send messages, with or without attachments.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	SEND	1
1	ADDPROFILE	2
2	DELPROFILE	3
3	DELALLPROFILES	4

Syntax 1

EMAIL(*Mode*, *EmailProfile, To, Cc, Bcc, Subject, Message* [, *FileAttachments*[, *Priority* [, *MessageFormat*[, *MessageEncoding*]]]])

Return type: INTEGER.

Argument	Meaning
EmailProfile	The name of an email profile as configured in Actions.Email.Profiles. The profile provides information such as the outgoing (SMTP) server name. If no profile is specified then the default profile is used. Type STR.
To	The address of the recipients as a list of semicolon delimited e-mail addresses. Type STR.
Cc	The address of any carbon copy recipients as a list of semicolon delimited e-mail addresses. Type STR.
Bcc	The address of the blind carbon copy recipients as a list of semicolon delimited e-mail addresses. Type STR.
Subject	The e-mail subject line. Type STR.
Message	The body of the e-mail message. Type STR.
FileAttachments	A list of semicolon delimited file attachments. Files in the list must be specified with absolute paths. Type Str. The maximum size of an email attachment is 10MB.
Priority	The priority of the message. Any numeric type. 0 = Normal 1 = Low 2 = High
MessageFormat	The format of the body of the e-mail message. Any numeric type. 0 = Plain text 1 = HTML
MessageEncoding	The encoding for the body of the e-mail topic. See the Message Encoding topic.

Execution

Mode	Mnemonic	Action
0	SEND	Send an e-mail. Returns 0 if successful, else 1. The sending status is recorded in the system variable.

Syntax 2

EMAIL(MODE, ProfileName, ProfileDescription, AccountName, AccountDescription, SenderAddress, SenderDisplayName, ReplyToAddress, ServerHost, ServerPort, DeliveryMethod, EncryptionMethod, AuthenticationMode, UserName, UserPassword);

Return type: INTEGER.

Argument	Meaning
ProfileName	The name of the profile. Type STR.
ProfileDescription	The description of the profile. Type STR.
AccountName	The name for the account. Type STR.
AccountDescription	The description of the account. Type STR.
SenderAddress	The E-mail address that will appear in the From field of the recipient's e-mail client. Type STR.
SenderDisplayName	If configured, this will be displayed in the From field of the recipient's message client instead of the source e-mail address. Type STR.
ReplyToAddress	E-mail address which will be used by the recipients email client if he chooses to reply to the message. Type STR.
ServerHost	The name of the outgoing server. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end. The outgoing server does not need to be reachable from the web clients.
ServerPort	The port number for the outgoing server. Any numeric type.
DeliveryMethod	A flag indicating the delivery method for the email. 1 via IIS, 0 via network.
EncryptionMethod	A flag indicating the encryption method. 1 for SSL, else 0.
AuthenticationMode	A flag indication if authentication is required. 1 for authentication, otherwise 0.
UserName	The user name if authentication is used. Type STR.
UserPassword	The use password if authentication is used. Type STR.

Execution

Mode	Mnemonic	Action
1	ADDPROFILE	Add a temporary email profile and account. Returns 1 if successful, else 0 (parameter error).

Syntax 3

EMAIL(MODE, ProfileName);

Return type: INTEGER.

Argument	Meaning
ProfileName	The name of the profile. Type STR.

Execution

Mode	Mnemonic	Action
2	DELPROFILE	Delete a temporary email profile and account. Only a profile and account that has been programmatically created can be deleted. Returns 1 if successful, else 0 (parameter error).

Syntax 4

EMAIL(MODE);

Return type: INTEGER.

Execution

Mode	Mnemonic	Action

3	DELALLPROFILES	Delete all temporary email profiles and accounts. Only profiles and accounts that have been programmatically created are deleted. Return always 1.

ERROR

See Also

Return the error code, program name, function name, or line number of the last error.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	ERRNO	1
2	PROGRAM	2
3	FUNCTION	2
4	LINE	1
5	TOFILE	3

Syntax 1

IntVal = ERROR(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	ERRNO	Return the last error code that occurred. Return: 0 = No error. 1 = Wrong string argument. 2 = The value is below the lower limit. 3 = The value is above the higher limit. 5 = The file is already open (FOPEN). 6 = The file is already closed (FCLOSE, FSEEK, FPUTS). 7 = The file failed to open (FOPEN).
4	LINE	Return the line number where the error last occurred.

Syntax 2

IntVal = ERROR(*Mode*);

The return type is STR.

Execution

Mode	Mnemonic	Action
2	PROGRAM	Return the name of the program where the error last occurred.
3	FUNCTION	Return the name of the function where the error last occurred.

Syntax 3

IntVal = ERROR(*Mode*, *Filename*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action

5	TOFILE	Redirects all the errors, which are displayed in the debug window to the specified filename. To disable redirection enter "" for the filename. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file is created on that computer. Return: The name of the program where the error last occurred.


Use \\ to specify the path to be used, so as to generate single \'s in the path string. If the path is not specified, the file is created by default in the TP folder of the project.

Example

```
SUB Main()
DIM intResult as integer;
'open the file
intResult = Error("TOFILE","errorfile.txt");
If (FOPEN("MESSAGE.TXT","R")==0) Then
  'If error in opening:
  PRINT ("code=\t",ERROR("ERRNO"));
  PRINT ("prog=\t",ERROR("PROGRAM"));
  PRINT ("fonc=\t",ERROR("FUNCTION"));
  PRINT ("line=\t",ERROR("LINE"));
Else
  'If no error in opening MESSAGE.TXT:
  '...
End If
END SUB
```

EVENT

See Also Example Further information

Program the execution of a function on the change of a variable.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	ADD or ADDPROG	1
2	ADDPROGEVT	7
3	DELPROGEVT	8
5	DEL	2
6	DELALL	3
7	IMPORTBYFILE	5
8	IMPORTBYHANDLE	6
9	ADDEMAIL	9
10	DELEMAIL	8
11	ADDPROGS	1
12	ADDSMS	10
13	DELSMS	8
21	DELPROG	4

Any program from which functions are triggered must be preloaded.

The use of DELAY within a function called periodically may produce unpredictable results and should be avoided.

Variable names should not be preceded by the @ character.

Syntax 1

IntVal = EVENT(*Mode*, *VarName*, *Sense*, *Prog*, *Branch*, *Func*[, *Farg* [, *ActivBit*]]) ;

The return type is INTEGER.

Argument	Meaning
VarName	The name of a variable from the Variables Tree. Type STR.
Sense	A string or string variable defining the trigger condition. See below.
Prog	The name of the program that contains the function to be executed. Type STR.
Branch	The branch for the above program. Type STR.
Func	The name of the function that is to be executed. Type STR.
Farg	Optional. A string of 2,047 characters maximum used to provide up to 8 arguments (separated by comma ",") to the function. The arguments are retrieved in the function using the verb GETARG.
ActivBit	Optional. The name of the bit or alarm variable which enables the event.

Execution

Mode	Mnemonic	Action
1	ADD or ADDPROG	Create an event. The conditions under which the event is triggered are defined by the Sense expression. If a function name is not supplied the MAIN function

		will be triggered. Any existing event attached to the variable will be replaced.
11	ADDPROGS	Add an event. The conditions under which the event is triggered are defined by the Sense expression. You can attach multiple events with triggering different functions with different senses to the same variable.

Syntax 2

IntVal = EVENT(*Mode*, *VarName*[, *Sense*, *Prog*, *Branch*, *Func*]);

The return type is INTEGER.

Argument	Meaning
VarName	The name of a variable from the Variables Tree. Type STR.
Sense	A string or string variable defining the trigger condition. See below.
Prog	The name of the program that contains the function to be executed. Type STR.
Branch	The branch for the above program. Type STR.
Func	The name of the function that is to be executed. Type STR.

Execution

Mode	Mnemonic	Action
5	DEL	Delete an existing event. The sense must be the same as the original event. If the *Sense*, *Prog*, *Branch*and *Identity* parameters are missing, then all events attached to the named variable are deleted. Return: 1 if successful, else 0.

Syntax 3

IntVal = EVENT(Mode, [, *Type*]);

The return type is INTEGER.

Argument	Meaning
Type	The type of events to be deleted. Type INTEGER. 0 = Delete all events that were created by a program. This is the default if Type is not used. 1 = Delete all program events that were created by a program. 2 = Delete all e-mail events that were created by a program.

Execution

Mode	Mnemonic	Action
6	DELALL	Delete events that were created by a program. Return: Always 1.

Syntax 4

IntVal = EVENT(*Mode*, Var*Name*, *Sense*, *Program*, *Branch*, *Function*);

Argument	Meaning
VarName	The name of a variable from the Variables Tree. Type STR.
Sense	A string or string variable defining the trigger condition. See below.
Prog	The name of the program that contains the function to be executed. Type STR.
Branch	The branch for the above program. Type STR.

Func	The name of the function that is to be executed. Type STR.

Execution

Mode	Mnemonic	Action
21	DELPROG	The identified single event is deleted. All the arguments are required. Return: 1 if successful, else 0.

Sense

Sense is an expression that defines the conditions under which the event is triggered. The available options depend on the type of variable with which the event is associated.

Conditions of variables

See the topic Conditions of Variables.

Syntax 5

IntVal = EVENT(*Mode*, *FileName*);

The return type is INTEGER.

Argument	Range
FileName	Name of the text file in the TP folder of the project or in a folder selected by an absolute path. Type STR.

The formats of event files, and of the means for adding and deleting them, are specified in the topic Events Maintenance by Program.

Execution

Mode	Mnemonic	Action
7	IMPORTBYFILE	Obtain events from the file in the specified folder.

Returns: 1 if OK, else 0.

Example

```
EVENT(7, "addEvent.dat");
'OR
EVENT("IMPORTBYFILE", "addEvent.dat");
```

Syntax 6

IntVal = EVENT(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Range
Handle	Address of the buffer memory area as allocated by ALLOC_BUFFER. Type LONG.

Execution

Mode	Mnemonic	Action
8	IMPORTBYHANDLE	Obtain events from a buffer.

Returns: 1 if OK, else 0.

Example

```
DIM hbuf AS LONG;
hbuf = FILETOBUF("addevent.dat");
EVENT(8, hbuf);
'OR
```

```
EVENT("IMPORTBYHANDLE", hbuf);
```

Syntax 7

IntVal = EVENT(*Mode*, *Name, Description, ServerListName, VarName, Sense, Prog, Branch, Func,*[, *Farg* [, *ActiveBit*]];

The return type is INTEGER.

Argument	Meaning
Name	The name of the event. The string you enter here will appear in the Name field of the Event Configuration dialog and is also used as the identifier when deleting the event programmatically.
Description	A description for the event. The string you enter here will appear in the Description field of the Event Configuration dialog
ServerListName	The name of a server list. On multi-station applications the event will only be active on a server station that appears in the list.
VarName	The name of a variable from the Variables Tree. Type STR.
Sense	A string or string variable defining the trigger condition. See below.
Prog	The name of the program that contains the function to be executed. Type STR.
Branch	The branch for the above program. Type STR.
Func	The name of the function that is to be executed. Type STR.
Farg	Optional. A string of 2,047 characters maximum used to provide up to 8 arguments (separated by comma ",") to the function. The arguments are retrieved in the function using the verb GETARG.
ActiveBit	Optional. The name of the bit or alarm variable which enables the event.

Execution

Mode	Mnemonic	Action
2	ADDPROGEVT	Create an event. The conditions under which the event is triggered are defined by the Sense expression. If a function name is not supplied the MAIN function will be triggered. An existing event with the same name will be replaced.

Returns: 1 if OK, else 0.

Syntax 8

IntVal = EVENT (*Mode*, *Name*);

The return type is INTEGER.

Argument	Range
Name	The name of the event. The string you enter here will appear in the Name field of the Event Configuration dialog and is also used as the identifier when deleting the event programmatically.

Execution

Mode	Mnemonic	Action
3	DELPROGEVT	Delete the named event.
10	DELEMAIL	Delete the named e-mail event.
13	DELSMS	Delete the named SMS event.

Returns: 1 if OK, else 0.

Syntax 9

IntVal = EVENT (*Mode*, *Mode*, *Name, Description, ServerListName, VarName, Sense, EmailTemplate,* [, *ActiveBit* [*, EmailProfile* [*, To* [*, Cc* [*, Bcc* [*, Subject* [*, Message* [, *FileAttachments* [, *Priority*[, *MessageFormat* [, *MessageEncoding*]]]]]]]]]])

The return type is INTEGER.

Argument	Meaning
Name	The name of the event. The string you enter here will appear in the Name field of the Event Configuration dialog and is also used as the identifier when deleting the event programmatically.
Description	A description for the event. The string you enter here will appear in the Description field of the Event Configuration dialog
ServerListName	The name of a server list. On multi-station applications the event will only be active on a server station that appears in the list.
VarName	The name of a variable from the Variables Tree. Type STR.
Sense	A string or string variable defining the trigger condition.
EmailTemplate	The name of an email template as configured in Actions.Email.Templates. The template field my be left blank. Type STR.
ActiveBit	Optional. The name of the bit or alarm variable which enables the event.
EmailProfile	Optional. The name of an email profile as configured in Actions.Email.Profiles. The profile provides information such as the outgoing (SMTP) server name. If no profile is specified then the default profile is used. Type STR.
To	Optional. The address of the recipients as a list of semicolon delimited e-mail addresses. Type STR.
Cc	Optional. The address of any carbon copy recipients as a list of semicolon delimited e-mail addresses. Type STR.
Bcc	Optional. The address of the blind carbon copy recipients as a list of semicolon delimited e-mail addresses. Type STR.
Subject	Optional. The e-mail subject line. Type STR.
Message	Optional. The body of the e-mail message. Type STR.
FileAttachments	Optional. A list of semicolon delimited file attachments. Files in the list must be specified with absolute paths. Type Str.
Priority	Optional. The priority of the message. Any numeric type. 0 = Normal 1 = Low 2 = High
MessageFormat	Optional. The format of the body of the e-mail message. Any numeric type. 0 = Plain text 1 = HTML
MessageEncoding	Optional. The encoding for the body of the e-mail topic. See the Message Encodingtopic

If a template is specified and, in the template, the behaviour of a property was configured as Imposed By The Template, the properties in the template will overwrite any that you configure in the instruction. The properties affected are EmailProfile, To, Cc, Bcc, Subject, Message, Attachments, Priority, MessageFormat and MessageEncoding.

Execution

Mode	Mnemonic	Action
9	ADDEMAIL	Create an event triggering an e-mail. The conditions under which the event is triggered are defined by the Sense expression. If an event of

the same name already exists it will be replaced.

Returns: 1 if OK, else 0.

Syntax 10

IntVal = EVENT(*Mode*, *Name*, *Description*, *ServerListName*, *VarName*, *Sense*, *MsgTemplate*, [, *ActiveBit* [, *SmsProfile* [, *PhoneNumbers*[,*Subject* [, *Message* [, *Format*]]]]]]) ;

The return type is INTEGER.

Argument	Meaning
Name	The event ID. Type STR.
Description	The event description. Type STR.
ServerListName	The name of a server list when used in a multi-station application. Type STR.
VarName	The name of a variable, from the Variables Tree, used to trigger the SMS event. Type STR.
Sense	A string or the name of a string variable defining the trigger condition. Type STR.
MsgTemplate	The name of a message template as configured in Actions.Messages.Message templates. Type STR.
ActiveBit	The name of the bit or alarm variable which enables the event. Optional. Type STR
SmsProfile	The name of an SMS profile as configured in Actions.Messages.SMS profiles. The profile provides information about the connection port and the simcard of the modem. Optional. Type STR.
PhoneNumbers	The telephone number(s) of the SMS recipient(s). Multiple numbers must be separated by semicolons. Optional. Type STR.
Subject	The SMS subject. Optional. Type STR.
Message	The SMS message. Optional. Type STR.
Format	The message format. Optional. 0 = Auto (Default value) 1 = Text 2 = PDU 7-bit 3 = Unicode

Execution

Mode	Mnemonic	Action
12	ADDSMS	Create an SMS event. The change in value of the variable under which the event is triggered is defined by the Sense expression.

For further examples, follow the Example link at the top of this page.

EXCELTOBUF

See Also Example

Create a memory buffer from an Excel file in XLSX format.

WebVue support - Yes.

Syntax

LongVal = EXCELTOBUF (*WorkbookPath, SheetName, LineSeparator, ColumnSeparator [, RangeFirstRow, RangeFirstColumn, RangeLastRow, RangeLastColumn]*);

Argument	Meaning
WorkbookPath	The path and/or the name of the workbook. If you don't supply a path the Supervisor will assume the workbook is in the project's TP folder. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file must exist on that computer.
SheetName	The name of the worksheet. Maximum 30 characters. Type STR
LineSeperator	The character to be used as a line separator in the buffer (one character such as ';' or '\n'). Type STR.
ColumnSeperator	The character to ne used as the column separator in the buffer (one character such as ',' or '\t'). Type STR.
RangeFirstRow	First row of read range in Excel sheet (starting from 1).
RangeFirstColumn	First column of read range in Excel sheet (Starting from 1)
RangeLastRow	Last row of read range in Excel sheet (Starting from 1)
RangeLastColumn	Last column of read range in Excel sheet (Starting from 1)

If the optional range parameters are used, all four must be included.

Execution

Action

Create a memory buffer from an Excel file in XLSX format. Maximum buffer size is 128KB.
Return: Handle of memory buffer if OK, otherwise 0.

It is unnecessary to use the ALLOC_BUFFER instruction prior to using EXCELTOBUF as the buffer space is automatically allocated, but FREE_BUFFER must be used to free the buffer space once it is no longer required to keep it in memory.

EXP

See Also

Exponential function (to base e).

WebVue support - Yes.

Syntax

DblVal = EXP(*Val*);

Argument	Meaning
Val	The number of which the exponential is to be taken. Any numeric type.

The return type is DOUBLE.

Execution

Execution

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
SUB Main()
'Declare variables
DIM dblExponential as double;
DIM sngValue as single;

sngValue = 2.23456;
dblExponential = Exp ( sngValue );
PRINT("Exp( ",sngValue," ) = ",dblExponentielle);
'Display "Exp( 2.23456 ) = 9.34237"
END SUB
```

EXPORT

See Also Example

Run a previously configured (Application Explorer) export.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	GENERATE	1
2	GENERATE_DATES	2
3	GENERATE_PERIOD	3

Syntax 1

IntVal = EXPORT (*Mode*, *ExportName*, [*ExportPageName*], [*OutputName*], [*ReferenceTime*], [*StatusVariable*]);

Argument	Meaning
ExportName	The name of the export to run (as configured in the Application Explorer). Type STR.
ExportPageName	The name of the page to be exported.If not specified all pages will be exported. Type STR.
OutputName	The output destination for the export. If the output format is csv then it is the name of the folder in which the output files are generated. If the output format is Excel it is the name of the workbook. If not specified the default output name configured in the Application Explorer will be used. Type STR If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the output is created on that computer.
ReferenceTime	The timestamp used as a reference to calculate the start and end times. If not specified the actual time when the function is run will be used. Type DOUBLE (DateTimeValue format)
StatusVariable	The name of a register variable in which the status of the export is saved. If not specified the status variable for the export, as configured in the Application Explorer, will be used. If specified the status variable for the export, as configured in the Application Explorer, is not used. Type STR.

Execution

Mode	Mnemonic	Action
1	GENERATE	Generate the export. Return: 1 if OK, 0 or negative value if NOK (ExportName does not match configured export, ExportPage does not match configured page, invalid file name, cannot create output file invalid timestamp or invalid export configuration.)

Syntax 2

IntVal = EXPORT (*Mode*, *ExportName*, [*ExportPageName*], [*FileName*], [*StartTime*], [*EndTime*], [*StatusVariable*]);

Argument	Meaning
ExportName	The name of the export to run (as configured in the Application Explorer). Type STR.
ExportPageName	The name of the page to be exported.If not specified all pages will be exported. Type STR.
FileName	The output destination for the export. If the output format is csv then it is the name of the folder in which the output files are generated. If the output format is Excel it is the name of the workbook. If not specified the default output name

	configured in the Application Explorer will be used. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the output is created on that computer.
StartTime	The timestamp to be used as the start time. Type DOUBLE (DateTimeValue format)
EndTime	The timestamp to be used as the end time. Type DOUBLE (DateTimeValue format)
StatusVariable	The name of a register variable in which the status of the export is saved. If not specified the status variable for the export, as configured in the Application Explorer, will be used. If specified the status variable for the export, as configured in the Application Explorer, is not used. Type STR.

Execution

Mode	Mnemonic	Action
2	GENERATE_DATES	Generate the export using the given start and end time stamps. Return: 1 if OK, 0 or negative value if NOK (ExportName does not match configured export, ExportPage does not match configured page, invalid file name, cannot create output file, invalid timestamps or invalid configuration in export.)

Syntax 3

IntVal = EXPORT (*Mode*, *ExportName*, [*ExportPageName*], [*FileName*], [*ReferenceTime*], [*PeriodType*], [*PeriodValue*], [*PeriodInterval*], [*StatusVariable*]);

Argument	Meaning
ExportName	The name of the export to run (as configured in the Application Explorer). Type STR.
ExportPageName	The name of the page to be exported.If not specified all pages will be exported. Type STR.
FileName	The output destination for the export. If the output format is csv then it is the name of the folder in which the output files are generated. If the output format is Excel it is the name of the workbook. If not specified the default output name configured in the Application Explorer will be used. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the output is created on that computer.
ReferenceTime	The timestamp used as a reference to calculate the start and end times. If not specified the actual time when the function is run will be used. Type DOUBLE (DateTimeValue format)
PeriodType	The period type. Completed or current. Type INTEGER. 0 = Current 1 = Completed -1 = Ignore the argument.
PeriodValue	Period value. Type INTEGER. 1 to 65535. -1 = Ignore the argument.
PeriodInterval	Interval type. Year, month, day..... Type INTEGER. 0 = Minutes 1 = Hours 2 = Days 3 = Weeks 4 = Months 5 = Years -1 = Ignore the argument.
StatusVariable	The name of a register variable in which the status of the export is saved. If not specified the status variable for the export, as configured in the Application Explorer, will be used. If specified the status variable for the export, as configured in the Application Explorer, is not used. Type STR.

Execution

Mode	Mnemonic	Action
3	GENERATE_PERIOD	Generate the export using the given start timestamp and the period. Return: 1 if OK, 0 or negative value if NOK (ExportName does not match configured export, ExportPage does not match configured page, invalid file name, cannot create output file, invalid timestamps or invalid configuration in export.)

EXPORT_LOG

See Also Example

Generate historic log data using the functionality provided by Data Export. See the main Data Export help for details (The Application Explorer.Data Export). See also the important Operational Notes below.

WebVue support - Yes.

The Data Export module is protected by a license option in the Supervisor's protection key. If you do not have this option then Data Export will only operate in demonstration mode.

Mode	Mnemonic	Syntax
1	GETRECORD	1
2	GETSTATISTIC	2
3	READBUFFER	3
4	CANCEL	4
5	DISPOSE	4

Common Arguments

Argument	Meaning
LogListName	The name of the log list from which to get the data. Type STR.
StartTime	The start time for the historic request. The start time must be earlier than the end time. Type DOUBLE.
EndTime	The end time for the historic request. The end time must be later than the start time. Type DOUBLE.
Events	A number, calculated using binary weighted values, representing the events to retreive. Type DOUBLE. 0x0000 0001 = Alarm on 0x0000 0002 = Alarm off 0x0000 0004 = Alarm on nack 0x0000 0008 = Alarm off nack 0x0000 0010 = Alarm on ack 0x0000 0020 = Alarm off ack 0x0000 0040 = Alarm not accessible 0x0000 0080 = Alarm inhibited 0x0000 0100 = Alarm program masked 0x0000 0200 = Alarm variable masked 0x0000 0400 = Alarm user masked 0x0000 0800 = Alarm expression masked 0x0000 1000 = Alarm taken into account 0x0000 2000 = Bit to 0 0x0000 4000 = Bit to 1 0x0000 8000 = Bit to NS 0x0001 0000 = User action command 0x0002 0000 = User action acknowledge 0x0004 0000 = User action logon/logoff 0x0008 0000 = User action execute program 0x0010 0000 = User action mask alarm
AlarmLevelMin	Minimum alarm level (0 to 29). Type INTEGER.
AlarmLevelMax	Maximum alarm level (0 to 29). Type INTEGER.
ExpressionFilter	Attribues expression filter. See the topic: The Application Explorer.Archives.Configuring archive units.Proprietary archive unit.Configuring log data recording.Using an attributes filter expression.
StatusVariable	The name of the register variable in which the status of the historic query is reported. Type STR.
LineSeparator	A single character that is to be used as a line separator in the request output.

ColumnSeparator	A single character that is to be used as a column separator in the request output.
MaxRetLines	The maximum number of lines that the request returns. 0 = no maximum. Type INTEGER.

Syntax 1

IntVal = EXPORT_LOG(*Mode*, *LogListName*, *StartTime*, *EndTime*, *Events*, *AlarmLevelMin*, *AlarmLevelMax*, *ExpressionFilter*, *StatusVariable*, *LineSeparator*, *ColumnSeparator*, *MaxRetLines*, *DataFormat*, *DataHeader*)

Argument	Meaning
DataFormat	Data format for columns. The format for each column is separated by a pipe symbol. Type STR. Example: #D/#M/#Y|#h:#m:#s|#E|#T
DataHeader	Header text. Each column header is separated by a pipe symbol. Type STR. Example: Date|Time|Event|Title

Execution

Mode	Mnemonic	Action
1	GETRECORD	Generate a historic request to return the log records from the Log List. Return: See the Operational Notes below.

Syntax 2

IntVal = EXPORT_LOG(*Mode*, *LogListName*, *StartTime*, *EndTime*, *Events*, *AlarmLevelMin*, *AlarmLevelMax*, *ExpressionFilter*, *StatusVariable*, *LineSeparator*, *ColumnSeparator*, *MaxValuesToProcess*, *MaxRetLines*, *StatFlag*, *StatSortOrder*, *StatCountersFormat*, *StatRounded*)

Argument	Meaning
StatFlag	A number, calculated using binary weighted values, representing the statistics to generate. Type LONGLONG. 0x0000 0000 0001 = Total transition count 0x0000 0000 0002 = Bit to 0 (Occurrences) 0x0000 0000 0004 = Bit to 1 (Occurrences) 0x0000 0000 0008 = Bit NS (Occurrences) 0x0000 0000 0010 = Bit to 0 (Duration) 0x0000 0000 0020 = Bit to 1 (Duration) 0x0000 0000 0040 = Bit NS (Duration) 0x0000 0000 0080 = Alarm off (Occurrences) 0x0000 0000 0100 = Alarm on (Occurrences) 0x0000 0000 0200 = Alarm off ack (Occurrences) 0x0000 0000 0400 = Alarm off nack (Occurrences) 0x0000 0000 0800 = Alarm on ack (Occurrences) 0x0000 0000 1000 = Alarm on nack (Occurrences) 0x0000 0000 2000 = Alarm unavailable (Occurrences) 0x0000 0000 4000 = Alarm masked by expression (Occurrences) 0x0000 0000 8000 = Alarm inhibited (Occurrences) 0x0000 0001 0000 = Alarm not accessible (Occurrences) 0x0000 0002 0000 = Alarm masked by program (Occurrences) 0x0000 0004 0000 = Alarm masked by user (Occurrences) 0x0000 0008 0000 = Alarm masked by variable (Occurrences) 0x0000 0010 0000 = Alarm off (Duration) 0x0000 0020 0000 = Alarm on (Duration) 0x0000 0040 0000 = Alarm off ack (Duration) 0x0000 0080 0000 = Alarm off noack (Duration) 0x0000 0100 0000 = Alarm on ack (Duration) 0x0000 0200 0000 = Alarm on nack (Duration) 0x0000 0400 0000 = Alarm unavailable (Duration)

	0x0000 0800 0000 = Alarm masked by expression (Duration) 0x0000 1000 0000 = Alarm inhibited (Duration) 0x0000 2000 0000 = Alarm not accessible (Duration) 0x0000 4000 0000 = Alarm masked by program (Duration) 0x0000 8000 0000 = Alarm masked by user (Duration) 0x0001 0000 0000 = Alarm masked by variable (Duration)
StatSortOrder	The statistic used to sort the results. Selected using one of the values used for *StatFlag*. 0 = no sorting. Type LONGLONG
StatCountersFormat	A number indicating the units for the counters. Type INTEGER. 0 = None 1 = Seconds 2 = Minutes 3 = Hours 4 = Days
StatRounded	Type INTEGER. 0 = Not rounded 1 = Rounded

Execution

Mode	Mnemonic	Action
2	GETSTATISTIC	Generate a historic request to return statistical values from the log list. Return: See the Operational Notes below.

Syntax 3

IntVal = EXPORT_LOG(*Mode*, *BufferHandle*);

Argument	Meaning
BufferHandle	The handle of a buffer previously created using ALLOC_BUFFER. Type LONG.

Execution

Mode	Mnemonic	Action
5	READBUFFER	Read a buffer of data after either a GETRECORD or GETSTATISTIC. The status variable must be either 0 (Completed), 4 (Completed but the maximum number of points has been reached) or 5 (Completed but the maximum number of processed values has been reached). Return: The number of lines retrieved or if less than 1: 0 = End of buffer -1 = Invalid buffer handle -2 = No previous historic request has been sent. -3 = No answer received. -4 = Nothing to read. -5 = Buffer too small.

Syntax 4

IntVal = EXPORT_LOG(*Mode*);

Execution

Mode	Mnemonic	Action
6	CANCEL	Cancel the current historic request. A CANCEL should always be followed by a DISPOSE. Return: Always 0
7	DISPOSE	Release internal buffer used in a historical request. Must always be called before making a further historic request or after a CANCEL. Return: Always 0.

Operational Notes

The EXPORT_LOG instruction operates asynchronously. That is once a historical request has been generated by either modes GETRECORD or GETSTATISTIC, the script continues without waiting for the historical data to be generated. The instruction's return value indicates if the historic request was successfully sent, or not.

When the historical request is complete (successfully or otherwise) the value of the Status Variable is changed accordingly. On this change, if the Status Variable's value indicates a success, READBUFFER mode must be used to retrieve the data into a memory buffer from where further processing can be done (for example exporting to Excel using BUFTOEXCEL).

It will normally be necessary to use READBUFFER more than once until the number of lines it returns is 0 indicating that all data has been processed.

If you use a loop including READBUFFER you must respect the 30 seconds limit that is allowed for a continuous SCADA Basic thread. If you exceed this (which is possible if you are processing large amounts of data) the thread will abort with an error.

After completion a DISPOSE mode must be executed to release the internal buffer used in the historical request.

Return values

All return values other than zero indicate an error. Of particular note is a return of -30 which indicates a previous request is still running and should be accommodated in the script.

Value	Meaning
0	Request sent
-1	LogListName is not a STR
-2	LogList doesn’t exist
-3	Start time undefined or invalid
-4	End time undefined or invalid
-5	Event code cannot be read
-6	Event code smaller or equal to 0
-7	AlarmLevelMin cannot be read
-8	AlarmLevelMin between 0 and 29
-9	AlarmLevelMax cannot be read
-10	AlarmLevelMax between 0 and 29
-11	AlarmLevelMin smaller than AlarmLevelMax
-12	ExpressionFilter is not a STR
-13	StatusVariable is not a STR
-14	StatusVariable is not a register
-15	LineSeparator cannot be read
-16	ColumnSeparator cannot be read
-20	DataFormat is not a STR
-21	DataHeader is not a STR
-22	Format and header have different column count
-23	StatFlag cannot be read
-24	StatFlag smaller or equal to 0
-25	StatSortingOrder cannot be read
-26	StatSortingOrder is smaller than 0
-27	StatCountersFormat cannot be read
-28	StatCountersFormat between 0 and 4
-29	StatRoundedToRequest cannot be read
-30	A previous request is still running

Status variable values

The status of the latest historical request.

Value	Meaning
0	Completed successfully.
1	Running.
3	Cancelled.
4	Completed but the maximum number of points has been reached.
5	Completed but the maximum number of processed values has been reached.

10	Failed.
11	Failed due to bad configuration.
12	Failed due to a failed historical request.

Examples

See the Examples link at the top of the topic.

EXPORT_TREND

See Also Example

Generate historic trend data using the functionality provided by Data Export. See the main Data Export help for details (The Application Explorer.Data Export). See also the important Operational Notes below.

WebVue support - Yes.

The Data Export module is protected by a license option in the Supervisor's protection key. If you do not have this option then Data Export will only operate in demonstration mode.

Mode	Mnemonic	Syntax
1	GETRAW	1
2	GETSAMPLE	2
3	GETSTATISTIC	3
4	GETAGGREGATED	4
5	READBUFFER	5
6	CANCEL	6
7	DISPOSE	6

Common Arguments

Argument	Meaning
ListVariables	The list of variables from which to collect historic data as either a string or a pointer to a buffer containing the string. Type STR or DOUBLE.
Branch	A branch that will be prefixed to all variable names. Type STR
Unit	The name of the archive unit that contains the historic data. If left empty then the primary archive unit for each variable is used. Type STR.
StartTime	The start time for the historic request. The start time must be earlier than the end time. Type DOUBLE.
EndTime	The end time for the historic request. The end time must be later than the start time. Type DOUBLE.
StatusVariable	The name of the register variable in which the status of the historic query is reported. Type STR.
LineSeparator	A single character that is to be used as a line separator in the buffer output.
ColumnSeparator	A single character that is to be used as a column separator in the buffer output.
MaxRetLines	The maximum number of lines that the request returns. 0 = no maximum. Type INTEGER.
MaxValuesToProcess	The maximum number of raw values that are to be processed. 0 = no maximum. Type INTEGER.
TimeStampsFormat	The format of the timestamp. 0 = date and time, 1 = date only, 2 = time only. Optional with 0 as the default.

Syntax 1

IntVal = EXPORT_TREND(*Mode*, *VariableName*, *Branch*, *Unit*, *StartTime*, *EndTime*, *StatusVariable*, *LineSeparator*, *ColumnSeparator*, *MaxRetLines*[, TimeStampsFormat]);

Argument	Meaning
VariableName	The variable from which to collect historic data. Type STR.

Execution

Mode	Mnemonic	Action
1	GETRAW	Generate a historic request to return the raw values for a single variable.

Return: See the Operational Notes below.

Syntax 2

IntVal = EXPORT_TREND(*Mode*, *ListVariables*, *Branch*, *Unit*, *StartTime*, *EndTime*, *StatusVariable*, *LineSeparator*, *ColumnSeparator*, *MaxValuesToProcess*, *MaxRetLines*, *SamplingRateInterval*, *SamplingRateValue* [,*SynchroSecondValue*, *SynchroMinuteValue*, *SynchroHourValue*, *SynchroDayValue*, *TimeStampsFormat*]);

Argument	Meaning
SamplingRateInterval	The sample rate units as a number. Type INTEGER. 0 = Seconds 1 = Minutes 2 = Hours 3 = Days 4 = Weeks 5 = Months 6 = Years
SamplingRateValue	The sampling rate value. Type INTEGER.
SynchroSecondValue *SynchroMinuteValue* *SynchroHourValue* *SynchroDayValue*	Equivalent to the Synchronisation Time and Day properties in the Data Export Sampling tab. See the main Data Export help for details (The Application Explorer.Data Export). Type INTEGER.

Execution

Mode	Mnemonic	Action
2	GETSAMPLE	Generate a historic request to return sampled value for the variables list. Return: See the Operational Notes below.

Syntax 3

IntVal = EXPORT_TREND (*Mode*, *ListVariables*, *Branch*, *Unit*, *StartTime*, *EndTime*, *StatusVariable*, *LineSeparator*, *ColumnSeparator*, *MaxValuesToProcess*, *MaxRetLines*, *StatFlag*);

Argument	Meaning
StatFlag	A number, calculated using binary weighted values, representing the statistics to generate. Type DOUBLE. 0x0001 = Min 0x0002 = Max 0x0004 = Average 0x0008 = Sum 0x0010 = Standard deviation 0x0020 = First value 0x0040 = Last value 0x0080 = Counter 0x0100 = Min date 0x0200 = Max date 0x0400 = First value date 0x0800 = Last value date 0x1000 = Weighted average

Execution

Mode	Mnemonic	Action
3	GETSTATISTIC	Generate a historic request to return statistical values for the variables list. Return: See the Operational Notes below.

Syntax 4

IntVal = EXPORT_TREND(*Mode*, *ListVariables*, *Branch*, *Unit*, *StartTime*, *EndTime*, *StatusVariable*, *LineSeparator*, *ColumnSeparator*, *MaxValuesToProcess*, *MaxRetLines*, *StatFlag*, *SamplingRateInterval*, *SamplingRateValue* [,*SynchroSecondValue*, *SynchroMinuteValue*, *SynchroHourValue*, *SynchroDayValue*, *TimeStampsFormat, TimeStampsFields*]);

Argument	Meaning
TimeStampsFields	A number indicating the timestamp fields to use. Type INTEGER. 0 = Beginning timestamp. The default if TimeStampFields is not used. 1 = End timestamp only. 2 = Both beginning and end timestamps.

See Syntax 2 and Syntax 3 for a description of the arguments StatFlag, SamplingRateInterval, SamplingRateValue, SynchroSecondValue, SynchroMinuteValue, SynchroHourValue and SynchroDayValue.

Execution

Mode	Mnemonic	Action
4	GETAGGREGATED	Generate a historic request to return aggregated statistical values for the variables list. Return: See the Operational Notes below.

Syntax 5

IntVal = EXPORT_TREND(*Mode*, *BufferHandle*);

Argument	Meaning
BufferHandle	The handle of a buffer previously created using ALLOC_BUFFER. Type LONG.

Execution

Mode	Mnemonic	Action
5	READBUFFER	Read a buffer of data after either a GETRAW, GETSAMPLED, GETSTATISTIC or GETAGGREGATED. The status variable must be either 0 (Completed), 4 (Completed but the maximum number of points has been reached) or 5 (Completed but the maximum number of processed values has been reached). Return: The number of lines retrieved or if less than 1: 0 = End of buffer -1 = Invalid buffer handle -2 = No previous historic request has been sent -3 = No answer received -4 = Nothing to read -5 = Buffer too small

Syntax 6

IntVal = EXPORT_TREND(*Mode*);

Execution

Mode	Mnemonic	Action
6	CANCEL	Cancel the current historic request. A CANCEL should always be followed by a DISPOSE. Return: Always 0
7	DISPOSE	Release internal buffer used in a historical request. Must always be called before making a further historic request or after a CANCEL. Return: Always 0

Operational Notes

The EXPORT_TREND instruction operates asynchronously. That is once a historical request has been generated by either modes GETRAW, GETSAMPLED, GETSTATISTIC or GETAGGREGATED, the script continues without waiting for the historical data to be generated. The instruction's return value indicates if the historic request was successfully sent, or not.

When the historical request is complete (successfully or otherwise) the value of the Status Variable is changed accordingly. On this change, if the Status Variable's value indicates a success, READBUFFER mode must be used to retrieve the data into a memory buffer from where further processing can be done (for example exporting to Excel using BUFTOEXCEL).

It will normally be necessary to use READBUFFER more than once until the number of lines it returns is 0 indicating that all data has been processed.

If you use a loop including READBUFFER you must respect the 30 seconds limit that is allowed for a continuous SCADA Basic thread. If you exceed this (which is possible if you are processing large amounts of data) the thread will abort with an error.

After completion a DISPOSE mode must be executed to release the internal buffer used in the historical request.

Return values

All return values other than zero indicate an error. Of particular note is a return of -30 which indicates a previous request is still running and should be accommodated in the script.

Value	Meaning
0	The historic request has been successfully sent.
-1	VariableName or ListVariable is of the wrong type or cannot be read.
-2	Branch is of the wrong type or cannot be read.
-3	UnitName is of the wrong type or cannot be read.
-4	Branch puls VariableName does not exist.
-5	The variable, or one of the variables is not configured for trend recording.
-6	The StartTime is invalid or undefined.
-7	The EndTime is invalid or undefined.
-8	StatusVariable is not a string or the variable it names is not a register.
-9	LineSeparator cannot be read.
-10	ColumnSeparator cannot be read.
-20	StatFlag is not a double.
-21	SamplingRateInterval is not an integer.
-22	SamplingRateValue is not an integer.
-23	Synchronise parameter is not valid.
-30	A previous historic request is still running.

Status variable values

The status of the latest historical request.

Value	Meaning
0	Completed successfully.
1	Running.
3	Cancelled.
4	Completed but the maximum number of points has been reached.
5	Completed but the maximum number of processed values has been reached.
10	Failed.

11 Failed due to bad configuration.
12 Failed due to a failed historical request.

Examples

See the Examples link at the top of the topic.

EXPRESSION

See Also Example

Dynamically import expression models or expressions on variables from a file or buffer.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	IMPORT_FILE_TEMP	2
1	IMPORT_BUFFER_TEMP	1
2	IMPORT_FILE_ONVAR	2
3	IMPORT_BUFFER_ONVAR	1
4	RECALCULATION	3

Syntax 1

IntVal = EXPRESSION(*Mode*, *Handle*[, *Overwrite*]);

Return type: INTEGER

Argument	Meaning
Handle	Allocation of the memory buffer, as returned by ALLOC_BUFFER or FILETOBUF. Type LONG
Overwrite	Replace the previous buffer. Optional. 0: NO (default value) 1: YES

Execution

Mode	Mnemonic	Action
1	IMPORT_BUFFER_TEMP	Import expression models from the specified buffer. Return: 1 if OK, else 0.
3	IMPORT_BUFFER_ONVAR	Import expressions on variables from the specified buffer. Return: >0 if OK, indicating the total number of items 0 if the format is not recognised, or the buffer is empty. -1 if the result variable of an expression already exists from another expression; the expression is not imported. -2 if the result variable does not exist. -3 if the expression is illegal

Syntax 2

IntVal = EXPRESSION(*Mode*, *FileName*[, *Overwrite*]);

Return type: INTEGER

Argument	Meaning
FileName	The name of the text file to be read, either from the TP folder of the project or using an absolute path. (Type STR)
Overwrite	Replace the previous file. Optional. 0: NO (default value) 1: YES

Execution

Mode	Mnemonic	Action
0	IMPORT_FILE_TEMP	Import expression models as specified, from the specified file. Return: 1 if OK, else 0.
2	IMPORT_FILE_ONVAR	Import expressions on variables as specified, from the specified file. Return: >0 if OK, indicating the number of items imported. 0 if the format is not recognised, or the file is empty. -1 if the result variable of an expression already exists from another expression; the expression is not imported. -2 if the result variable does not exist. -3 if the expression is illegal.

Syntax 3

IntVal = EXPRESSION(*Mode* [, *ExpressionName*]);

Return type: INTEGER

Argument	Meaning
ExpressionName	The name of an expression. (Type STR)

Execution

Mode	Mnemonic	Action
4	RECALCULATION	Force a recalculation of the named expression on variable. If the expression name is not supplied, all expressions are recalculated. Return: 1 if OK, -1 if expression not found.

Can only be used for expressions whose result is a variable. Cannot be used for expression templates.

Example

For an example, select the Example link above.

F

FCLOSE

See Also Example

Close the specified file.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

IntVal = FCLOSE(*Filename*);

The return type is INTEGER.

Argument	Meaning
Filename	The name, including any path, of the file to be closed. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.

Execution

Close the specified file.
Return: 1 if successful, else 0.

Example

```
i1 = FCLOSE("histo.fil");
```

For further examples, select the Example link above.

FCOPY

See Also Example

Copy a file from source to destination.

WebVue support - Yes.

Syntax

IntVal = FCOPY (*ExistingFileName*, *NewFileName*[, *AllowOverwrite*]);

Argument	Meaning
ExistingFileName	The name and optional path of the file to be copied. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
NewFileName	The name and optional path of the file to be created. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
AllowOverwrite	A flag to allow overwrite of a file of the same name. 0 = Do not allow. 1 = Allow.

Execution

Copy the specified file. The optional *AllowOverwrite* flag can be used to permit an existing file of the same name to be overwritten.

Return:
1 - Successful.
0 - Failed. The source file does not exist or it is opened and locked by another application.
-1 - Failed. The destination file already exists and *AllowOverwrite* is set to 0.
-2 - Failed. The source file is open (Fopen instruction).

Further Information

The *ExistingFileName* and *NewFileName* can be either just a file name or include a full or a relative path. If a path is not included the operation takes place within the project's TP folder. If you include a path in the *NewFileName*, and the folder does not exist, it will be created.

Examples

FileName	Example
File name	MyFile.dat
With full path	C:\\temp\\MyFile.dat
With relative path	..\\..\\Files\\MyFile.dat

Note the requirement to use a double slash as the folder separator.

FEOF

See Also

Check whether the end of a file has been reached.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

IntVal = FEOF(*Filename*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file to be checked. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.

Execution

FEOF returns zero if the reading functions (like FGETS or FGETC) have not reached the end of the named file.

The function FEOF does not move the file pointer on reading. Consequently, the following loop will travel through the file one time too many and L will be (NULL) on the last line. The test for the end of file should directly follow the read (FGETS) instruction.

```
WHILE(FEOF(filename) == 0)
  L = FGETS(filename,80);
  PRINT(L);
WEND
```

Example

A control structure of the type DO{.....}WHILE(CONDITION) is not available within SCADA BASIC, however, the following construct provides the same functionality:

```
eof = 0;
WHILE(eof == 0)
  L = FGETS(filename,80);
  eof = FEOF(filename);
  IF(eof == 0) THEN
    PRINT(L);
 END IF
WEND
```

FGETC

See Also

Read a single character from a file.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

StrVal = FGETC(*Filename*);

The return type is STR.

Argument	Meaning
Filename	The file from which the character is to be read. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.

Execution

StrVal Each time a character is read the file pointer is incremented by 1 character.

Example

```
str1 = FGETC("histo.fil");
```

FGETS

See Also

Read a character string from a file.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

StrVal = FGETS(*Filename*, *Num*);

The return type is STR.

Argument	Meaning
Filename	The name of the file from which the string is to be read. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Num	The number of characters to be read. Any numeric type.

Execution

StrVal The specified number of characters is read and returned in a string.

If the end of the file is reached before the number of characters is retrieved, or a Line Feed character is found, the return string will be truncated accordingly. The file pointer is advanced by the number of characters read.

Example

```
str1 = FGETS("HELP.TXT",20);
```

FILETOBUF

See Also

Create a memory buffer using the contents of an ASCII file.

WebVue support - Yes.

Syntax 1

LongVal = FILETOBUF(*Filename*);

The return type is LONG.

Argument	Meaning
Filename	The name of the file from which the buffer is to be created. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file must exist on that computer.

Execution

A buffer is created in memory and the handle (start address) is returned by the function. The buffer size is automatically allocated using the size of the file.

Syntax 2

LongVal = FILETOBUF(*Filename*, *Size*);

Argument	Meaning
Filename	The name of the file from which the buffer is to be created. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file must exist on that computer.
Size	The size of the buffer to be created. Any numeric type.

The return type is LONG.

Execution

A buffer is created in memory and the handle (start address) is returned by the function. The buffer size is fixed at the specified size. If the file is larger than the maximum permitted size of a buffer (see the note below) then the buffer is not created.

Return: The handle of the buffer if successful, else 0.

Any files opened by SCADA BASIC must reside in the project folder TP.
The maximum size of the memory buffer is 32KB
It is unnecessary to use the ALLOC_BUFFER instruction prior to using FILETOBUF as the buffer space is automatically allocated, but FREE_BUFFER must be used to free the buffer space once it is no longer required to keep it in memory.

Example

```
'This program uses the file UTIL.TXT in the project TP folder
SUB Main()
DIM StrFilename as Str;
DIM strLine as Str;
DIM lngBuffer as Long;

StrFilename = "util.txt"; 'File name
lngBuffer = FILETOBUF(StrFilename);     'The test file must exist
strLine = CGET_BUFFER (lngBuffer,0,20); 'Print the first 20 characters
FREE_BUFFER(lngBuffer );
```

```
PRINT(strLine);
END SUB
```

FILETRANSFER

See Also

Manage file transfer from communication objects. See the important Operational Notes below. Currently supported for IEC 61850 devices only.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	DOWNLOAD	1
2	DOWNLOAD_DIRECTORY	2
3	DELETE	3

Syntax 1

Return = FILETRANSFER(*Mode*, *ComObj, SourceFile, DestinationFolder [*, *ResultVar[, CurrentPositionVar[, MaxPositionVar]]]*);

The return type is numeric.

Argument	Meaning
ComObj	The full path of the communication object, as configured in the Application Explorer, to which the file transfer applies. For example Network01.Device01. Type STR.
SourceFile	The name of a file in the communication object. Type STR.
DestinationFolder	The destination folder location. See note below. Type STR
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. See the table below for possible values and their meaning. Type STR.
CurrentPositionVar	The name of a register variable that will be regularly updated with the number of bytes downloaded. Type STR.
MaxPositionVar	The name of a register variable to represent the total file size. Type STR.

Execution

Mode	Mnemonic	Action
1	DOWNLOAD	Download the named file from the communication object to the Supervisor. Return: 0 if successful. A negative value is a parameter error, the value indicating which parameter is bad. For example -3 means that parameter 3 is bad.

Syntax 2

Return = FILETRANSFER*(Mode, ComObj, SourceFolder, DestinationFolder[, ResultVar[, CurrentFileVar[, MaxFileVar[, CurrentPositionVar[, MaxPositionVar]]]]]);*

The return type is numeric.

Argument	Meaning
ComObj	The full path of the communication object, as configured in the Application Explorer, to which the file transfer applies. For example Network01.Device01. Type STR.
SourceFolder	The name of a folder in the communication object. Type STR.
DestinationFolder	The destination folder location. See note below. Type STR
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. See the table below for possible values and their meaning. Type STR.
CurrentFileVar	The name of a register variable that will be regularly updated with the number of bytes downloaded. Type STR.
MaxFileVar	The name of a register variable to represent the total file size. Type STR.
CurrentPositionVar	The name of a register variable to represent the index number of the current

	file being downloaded. Type STR.
MaxPositionVar	The name of a register variable to represent the total number of files to be downloaded. Type STR.

Execution

Mode	Mnemonic	Action
1	DOWNLOAD_DIRECTORY	Download all files in the specified folder of the communication object to the Supervisor. Return: 0 if successful. A negative value is a parameter error, the value indicating which parameter is bad. For example -3 means that parameter 3 is bad.

Syntax 3

Return = FILETRANSFER*(Mode, ComObj, SourceFile [, ResultVar]);*

The return type is numeric.

Argument	Meaning
ComObj	The full path of the communication object, as configured in the Application Explorer, to which the file transfer applies. For example Network01.Device01. Type STR.
SourceFile	The name of a file within the communication object. Type STR.
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. See the table below for possible values and their meaning. Type STR.

Execution

Mode	Mnemonic	Action
1	DELETE	Delete the named file from the communication object. Return: 0 if successful. A negative value is a parameter error, the value indicating which parameter is bad. For example -3 means that parameter 3 is bad.

Operational Notes

The FILETRANSFER instruction operates asynchronously. That is once a request has been generated, the script continues without waiting for the request to be completed. The ResultVar can be used to monitor the status of the request.

Destination folder location

The destination folder is the project's TP folder by default. This will be used if the *DestinationFolder* parameter is empty.

You can also use a full path like "C:\\MyFiles" or a relative path like "..\\..\\MyFiles"

ResultVar values and meaning

0	No Error
1	Unknown
2	Internal Error
3	No CommObj
4	Not Connected
5	Not Supported
6	Access Denied
7	Init Download Failed
8	Download Failed
9	InitUpload Failed
10	Upload Failed
11	Delete File Failed

12 Browsing Failed
13 Routing Message Fail
14 No Remote Server
15 No Transaction
16 No Server
17 No Connection
18 Flow Controlled
19 Max Services Exceeded
20 No Read Data
21 Memory
22 Encoding
23 Bad Transaction
24 Connection Closed
25 Time Out
26 Connection State
27 Application
28 Parameters
29 Confirmed Error
30 Reject
31 Filename Ambiguous
32 File Busy
33 Filename Syntax Error
34 Content Type Invalid
35 Position Invalid
36 File Access Denied
37 File Non Existent
38 Duplicate Filename
39 Insufficient Space In Filestore
40 Device Not Started
41 Device In Error
42 Cannot Access Destination File
43 Definition
44 No Files Or No Folder
45 Move Error No Access
46 Move Error File Not Found
47 Move Error Invalid Name
48 Move Error Already Existing

FMOVE

See Also Example

Move a file from source to destination.

WebVue support - Yes.

Syntax

IntVal = FMOVE (*ExistingFileName*, *NewFileName*[, *AllowOverwrite*]);

Argument	Meaning
ExistingFileName	The name and optional path of the source file. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
NewFileName	The name and optional path of the destination file. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
AllowOverwrite	A flag to allow overwrite of a file of the same name. 0 = Do not allow (default). 1 = Allow.

Execution

Move the specified file. The optional *AllowOverwrite* flag can be used to permit an existing file of the same name to be overwritten.

Return:
1 - Successful.
0 - Failed. The source file does not exist or it is opened and locked by another application.
-1 - Failed. The destination file already exists and *AllowOverwrite* is set to 0.
-2 - Failed. The source file is already opened by SCADA Basic (Fopen).

Further Information

The *ExistingFileName* and *NewFileName* can be either just a file name or include a full or a relative path. If a path is not included the operation takes place within the project's TP folder. If you include a path in the *NewFileName*, and the folder does not exist, it will be created.

Examples

FileName	Example
File name	MyFile.dat
With full path	C:\\temp\\MyFile.dat
With relative path	..\\..\\Files\\MyFile.dat

Note the requirement to use a double slash as the folder separator.

FOPEN

See Also Example

Open the specified file in accordance with the specified access mode.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

IntVal = FOPEN(*Filename*, *Access*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file to be opened. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Access	The access mode for the open function. Type STR.
r	Open a text file for read.
w	Create a text file and open it for write. Overwrites an existing file of the same name.
a	Open an existing file for write, and append to the end.
r+	Open a text file for both read and write.
w+	Create a text file and open it for read and write. Overwrites an existing file of the same name.
a+	Open an existing file for read, write and append to the end.
b	Open a file in a binary mode.

Execution

A file is opened with the given access mode.

Return: 1 if successful, else 0.

Any files opened by SCADA BASIC must reside in the project folder TP.

It is essential to check the return flag each time a file is opened as attempting to read from a file that has not been opened successfully may cause a fatal error.

When the file is in text mode the LF character is transformed into CR LF in the file.

Example

The default folder is Project\TP Folder

```
'write file histo.fil in the TP folder
i1 = FOPEN("histo.fil","w+");
'read file in the C:\Test folder
i1 = FOPEN("C:\\TEST\\histo.fil","r");
```

FOR ... NEXT

See Also

Repeat a block of instructions a specified number of times.

WebVue support - Yes.

Syntax

FOR(*expression1*;*expression2*; [*expression3*])

[block of instructions]

NEXT

There is no return type.

Execution

Expression1 is evaluated when the Execution of the loop starts, to provide the initial state of the loop.

Expression2 is the loop test. The loop will repeat whilst the condition is true.

Expression3 provides the update or increment for the loop.

Example

```
SUB Main()
'Declare variables
DIM i as Integer;

'Integer i increases from 0 to 7 by steps of 2
For (i=0;i<7;i=i+2)
  Print(i);
Next
END SUB
```

FORMAT

See Also Example

Returns a string including an inserted formatted variable at any position within it.

WebVue support - Yes.

Unlike the similar PRINTF instruction found in many programming languages, FORMAT only supports one formatted variable per use.

Syntax

StrVal = FORMAT("*xxxxx*%*FormatString yyyyy*", *Variable*);

The return type is STR.

Argument	Meaning
xxxxx	A text string that will appear before the formatted value in the returned string. Optional.
yyyyy	A text string that will appear after the formatted value in the returned string. Optional.
FormatString	A placeholder string that determines the format and position of the inserted value. It starts with the character % and contains *Width*, *Precision* and *Modifier*.
Width	A number specifying the width of the inserted variable (as a number of characters). If this is preceded by a minus sign the characters will be left justified, otherwise they will be right justified. Optional.
Precision	The maximum number of characters of the inserted variable, the maximum number of digits for an integer or the number of digits to the right of the decimal point for a floating point number. It is always preceded by a full stop. Optional.
Modifier	A character specifying how the variable is to be returned. Type STR.
Variable	The variable that is to be formatted.

The whole of the first argument is enclosed in double quote marks and contains no separators other than the % sign that marks the start of the placeholder string, e.g. "Result = %d approx.".

Execution

A variable's value is returned within a string as a formatted sub-string. The modifier must correspond to the variable type.

Modifier	Variable type	Return format
d, i	Integer	Decimal number.
o	Integer	Octal number (not preceded by a zero).
x, X	Integer	Hexadecimal number, using *abcdef* (*x*) or *ABCDEF*(*X*) for 10 to 15 (not preceded by a 0*x*).
u	Integer	Unsigned decimal number.
c,C	Integer	Single character for the specified ASCII code.
s	Str	Returns a single-byte character string from a string up to the first null or up to the number of characters specified by *Precision*.
f	Double	Decimal notation of the form [-]*m.dddddd*, where the number of *d*'s is given by the precision (6 by default).
e, E	Double	Exponential notation of the form [-]*m.ddddde*±*xx* or [-]*m.ddddd*E±*xx*, where the number of *d* character positions is given by *Precision* (6 by default).
g, G	Double	Equivalent to %*e* or %*E* if the exponent is less than -4 or greater than or equal to *Precision*; otherwise is equivalent to %*f*.  Zeros or the terminating decimal point are not returned.
ld	Long	Four byte integer.
lld	LongLong	Eight byte integer.

Example

For an example of how the FORMAT command displays various text strings, select the Example link above.

FORMULA

See Also Example

Activate or modify a formula.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	ADD	1
2	ENABLE	2
3	DISABLE	2
5	DEL	2
6	DELALL	3

Syntax 1

IntVal = FORMULA(*Mode*, *handle*);

Argument	Meaning
handle	Points to the buffer in which all the arguments needed for the formula are specified. Type LONG.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	ADD	Creates the formula named "label". All the arguments are specified in the buffer pointed to by "handle". The contents of the buffer must follow the syntax of the file FORMULA.DAT. Several formulas may be defined in the same buffer. The formula created is temporary. Any formula created by the configuration cannot be deleted or modified in this way. Return: The number of the formula correctly configured and running.

Syntax 2

IntVal = FORMULA(*Mode*, *label*, *branch*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	ENABLE	Activates the formula specified by the label and branch. This may be applied to all configured formula. Return: 1 if the formula exists and the syntax is correct, else 0.
3	DISABLE	Disables the formula specified by the label and branch. This may be applied to all configured formula. Return: 1 if the formula exists and the syntax is correct, else 0.
5	DEL	Deletes the formula specified by the label and the branch. Return: 1 if the formula exists, else 0.

Syntax 3

IntVal = FORMULA(*Mode*[, *Class*]);

Argument	Meaning
Class	Identifies the class of formula.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	DELALL	Deletes all the existing temporary formulae. If the class is given, only formulae of that class will be deleted. Return: 1 if the formulae are correctly deleted, else 0.

Example

```
SUB main()
DIM hdl as LONG;
hdl = filetobuf("ftest.dat"); 'formula file
print(formula("ADD",hdl));
free_buffer (hdl);
END SUB

SUB actif()
formula("ENABLE","formula test1 tempo","");
END SUB

SUB noactif()
formula("DISABLE","formula test1","");
END SUB
```

For further examples, select the Example link above.

FPUTC

See Also Example

Write a single character to a file.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

IntVal = FPUTC(*Filename*, *Char*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file to which the character is to be written. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Char	The character which is to be written. Type STR

Execution

Return: 1 if successful, else 0.

Example

```
err = FPUTC("histo.fil","A");
```

For further examples, select the Example link above.

FPUTS

See Also Example

Write a character string to a file.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

IntVal = FPUTS(*Filename*, *Chars*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file to which the string is to be written. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Chars	The string which is to be written. Type STR.

Execution

Return: 1 if successful, else 0.

Example

```
i1 = FPUTS("histo.fil",str1);
i1 = FPUTS("histo.fil",CHR(10)); 'Line feed
```

For further examples, select the Example link above.

FREAD

See Also

Reads a number of data items of the same type from a file and stores them in a memory buffer.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

LongVal = FREAD(*Filename*, *Handle*, *Size*, *N*);

The return type is LONG.

Argument	Meaning
Filename	The name of the file from which the data is to be read. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Handle	The location of the memory buffer into which the data is read. Type LONG.
Size	The size in bytes for each data item. Any numeric type.
N	The number of items to be read. Any numeric type.

Execution

Data items are read into a memory buffer. The buffer must be previously allocated with ALLOC_BUFFER, which also supplies the handle.

Return: The number of data items read.

Example

```
'read 20 data items of 4 bytes
i1 = FREAD("histo.fil",handle,4,20);
```

FREE_BUFFER

See Also

Release a memory area reserved by an ALLOC_BUFFER.

WebVue support - Yes.

Syntax

FREE_BUFFER(*Handle*);

There is no return type.

Argument	Meaning
Handle	The location of the memory buffer as returned by ALLOC_BUFFER.

Execution

The memory buffer allocated by ALLOC_BUFFER and referenced by *Handle* is released.

Example

```
SUB Main()
'Declare the return code
DIM lngPointer as long;
DIM intValue as integer;

intValue = 50;
'intValue <=> intValue*4 bytes
lngPointer = ALLOC_BUFFER (intvalue);
PRINT("pointer =", lngPointer );

'The memory area from ALLOC_BUFFER must always be released after use
FREE BUFFER (lngPointer);
END SUB
```

FSEEK

See Also Example

Move a file pointer to a new position.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

IntVal = FSEEK(*Filename*, *Offset*, *Origin*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file on which the operation is to be performed. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Offset	The offset in bytes by which the pointer is to be moved. Any numeric type.
Origin	The point from which the offset is taken. 0 specifies the start of the file, 1 the current position of the pointer and 2 the end of the file.

Execution

The file pointer is moved to a new position. The next operation on the file will be made from there.

Return: 1 if successful, else 0.

Example

```
'move the pointer to the 20th byte from the start
i1 = FSEEK("histo.fil",20,0);
```

For further examples, select the Example link above.

FSTAT

See Also

Return the size and last modification date for the specified file.

WebVue support - Yes.

Syntax

IntVal = FSTAT(*Filename*, *Handle*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file on which the operation is to be performed. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Handle	The location of the memory buffer in which the data is to be returned. Type LONG.

Execution

The file statistics are returned in a memory buffer previously allocated with ALLOC_BUFFER or by FILETOBUF. The size of the file is found at an offset of 0 and the date at an offset of 4. The minimum buffer size for the operation is 22. The file may be open or closed.

Return: 1 if successful, else 0.

Example

```
DIM bufh as LONG; 'buffer handle
CONST SIZE=0; 'offset to read the size
CONST MODIF=4; 'offset to read the date
CONST ALLOC=22; '22 is the minimum required
bufh=ALLOC_BUFFER(ALLOC);
IF(FSTAT("file.txt",bufh))==1) THEN
  PRINT("Size in bytes:",LGET_BUFFER(bufh,SIZE));
  PRINT("Date of last modification:",CGET_BUFFER (bufh,MODIF,18));
ENDIF
```

FTP

See Also Example

Copies a file to or from a file-transfer server (FTP site).

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	DOWNLOAD	1
2	UPLOAD	2

Syntax 1

FTP("DOWNLOAD", *FTPfilesource*, *Filedestination* [, *HMIdisplay* [, *UserName*, *Password* [, *VariableName*]]]);

Argument	Meaning
FTPfilesource	The full path of the file to download e.g. "ftp:*server**path*". Type STR.
Filedestination	The path to the local destination for storing the file. It can be an absolute path or a relative path from the TP project directory, but not a path to a FTP site. Type STR.
HMIdisplay	Whether and how to display the progress dialog. 0: None. 1: Dialog without a Cancel button. (Default) 2: Dialog with a Cancel button. Type: INTEGER.
UserName	The user name of the account used to access the FTP server. If this is omitted, an anonymous login will be used. Type STR.
Password	The password of the account used to access the FTP server, if any. Type STR.
VariableName	The name of a register variable used to provide status indication. 0: Pending. 1: Completed. 2: Cancelled by the user. 3: The destination remote server name cannot be resolved. 4: Login or password incorrect. 7: Transfer cancelled by error. Type STR.

If the variable does not exist, the FTP transfer works but no status is returned. An information message is displayed in the Event Viewer.

The status variable name can be in the form of *varname* as well as @*varname*.

Execution

Mode	Mnemonic	Action
1	DOWNLOAD	Copy a file from the FTP server to the local destination. The return value is as follows. 0: No error. -1: First argument is missing. -2: Second argument is missing. -3: Third argument is out of range.

Syntax 2

FTP("UPLOAD", *Filesource*, *FTPfiledestination* [, *HMIdisplay* [, *UserName*, *Password* [, *VariableName*]]);

Argument	Meaning
Filesource	The full path of the file to be uploaded. It can be an absolute path or a relative path from the TP project directory, but not a path to a FTP site. Type STR.
FTPfiledestination	The path to the FTP destination. Type STR.
HMIdisplay	Whether and how to display the dialog of progress: 0: None. 1: Dialog without a Cancel button. (Default) 2: Dialog with a Cancel button. Type INTEGER.
UserName	The user name of the account used to access the FTP server. If this is omitted, an anonymous login will be used. Type STR.
Password	The password of the account used to access the FTP server, if any. Type STR.
VariableName	The name of a register variable used to provide status indication. 0: Pending. 1: Completed. 2: Cancelled by the user. 3: The destination remote server name cannot be resolved. 4: Login or password incorrect. 5: The source file for uploading does not exist. 6: Logged on but permission denied for uploading. 7: Transfer cancelled by error. Type STR.

If the variable does not exist, the FTP transfer works but no status is returned. An information message is displayed in the Event Viewer.

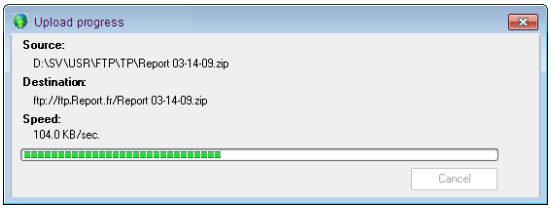
The status variable name can be in the form of *varname* as well as @*varname*.

Execution

Mode	Mnemonic	Action
1	UPLOAD	Copy a file from the local storage to the FTP server. The return value is as follows. 0: No error. -1: First argument is missing. -2: Second argument is missing. -3: Third argument is out of range.

The HMI for file transfer

During transfer, a dialog is optionally displayed with both paths, the speed of data transfer and a progress bar. Show picture



Example

For an example, select the Example link at top of page.

FWRITE

See Also

Write a number of data items of the same type to a file from data stored in a buffer.

WebVue support - Yes.

Before using any of the file management instructions, except FSTAT, RENAME, UNLINK, FILETOBUF or BUFTOFILE, you must first open the file using an FOPEN instruction.

Before any program including the FOPEN instruction ends, it must execute an FCLOSE instruction.

Syntax

LongVal = FWRITE(*Filename*, *Handle*, *Size*, *N*);

The return type is LONG.

Argument	Meaning
Filename	The name of the file to which the data is to be written. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.
Handle	The location of the memory buffer from which the data is read. Type LONG
Size	The size in bytes for each data item. Any numeric type.
N	The number of items to be written. Any numeric type.

Execution

Data items up to the specified number are written to the file. The buffer must be previously allocated with ALLOC_BUFFER, which also supplies the handle, or FILETOBUF.

Return: The number of data items written.

Example

```
'write 20 items of 4 bytes each
i1 = FWRITE("histo.fil",handle,4,20);
```

G

GETARG

See Also Example

Return the calling context of a function.

WebVue support - Yes.

Mode	Mnemonic	Syntax	Return
0	MAINBRANCH	1	STR
1	ARG1	1	STR
2	ARG2	1	STR
3	ARG3	1	STR
4	ARG4	1	STR
5	ARG5	1	STR
6	ARG6	1	STR
7	ARG7	1	STR
8	ARG8	1	STR
10	SOURCE	1	STR
11	PROGRAM	1	STR
12	BRANCH	1	STR
13	FUNCTION	1	STR
14	WINDOW	1	STR
15	WBRANCH	1	STR
16	IDENTIFIER	1	STR
17	VARNAME	1	STR
18	VARVALUE	1	SINGLE
19	VARSTATUS	1	INTEGER
20	KEYTYPE	1	STR
21	KEYCODE	1	INTEGER
22	CRONTYPE	1	INTEGER
23	CRONDATE	1	STR
24	CRONTIME	1	STR
25	ARG9	1	STR
26	ARG10	1	STR
27	ARG11	1	STR
28	ARG12	1	STR
29	WEB	1	INTEGER
30	TS_VALUE	1	DOUBLE
31	TS_TYPE	1	INTEGER
32	VARVALUE_REGISTER	1	DOUBLE
33	VARVALUE_BIT	1	INTEGER
34	VARVALUE_ALARM	1	INTEGER
35	VARVALUE_TEXT	1	STR

Modes ARG1 to ARG16 support the use of substitution characters. See the topic Using Substitution Strings in an Animation in the book Developing the HMI.animation.What you can

display in an animation.

Syntax 1

RtnVal = GETARG(*Mode*);

See table above for return type.

Execution

Mode	Mnemonic	Action
0	MAINBRANCH	Returns the branch name of the program, that contains the call to GETARG.
1-8	ARG1 to ARG8	Returns the 1st to the 8th argument from the calling function argument field.
25-28	ARG9 to ARG12	Returns the 9th to 12th arguments from the calling function argument field. The argument field may be specified when using : CYCLIC EVENT CRONTAB KEY EDITOR.Animation.Send.Program Action linked to an alarm. Selection or validation of a table animation via SELECTOR.
10	SOURCE	Returns a character string giving the source of the call: CYCLIC EVENT CRONTAB KEY MIMIC ACTIONS ARRAY_SELEC ARRAY_INPUT SCRIPT
11	PROGRAM	Returns the name of the current program.
12	BRANCH	Returns the branch name passed as an argument to the current function.
13	FUNCTION	Returns the name of the current function.
14	WINDOW	Returns the window name from where the call was made. This is used in the case of EDITOR.Animation.Link.Program, action linked to an alarm or EDITOR.Animation.Array.
15	WBRANCH	Returns the branch name of the window as in Mode 14.
16	IDENTIFIER	Returns the identifier of the animation concerned as in Mode 14.
17	VARNAME	Returns the name of the variable in the case of an EVENT.
18	VARVALUE	Returns the new value of a bit, alarm or register variable in the case of an EVENT. Cannot be used for text variables.
19	VARSTATUS	Returns If the SOURCE is either MIMIC or ARRAY_SELECT, the region in which the animation resides. If the SOURCE is EVENT, the status of the triggering variable. 1 if the variable is valid, 0 if the variable is invalid. The return type is INTEGER.
20	KEYTYPE	Returns the shift type ("Down/Shift/Control") when using the KEY

		verb.
21	KEYCODE	Returns the keycode when using the KEY verb.
22	CRONTYPE	Returns the agenda type used in CRONTAB: ONCE EACH_HOUR EACH_DAY EACH_WEEK EACH_MONTH
23	CRONDATE	Returns the date or the day of the time trigger according to the type used in CRONTAB.
24	CRONTIME	Returns the hour or minute of the time trigger according to the type used in CRONTAB.
29	WEB	Returns a flag to indicate the manner of access: 0 = local user non-zero = WebVue client (using a Run Program animation).
30	TS_VALUE	Returns the UTC timestamp of the triggering event represented by the number of milliseconds since 1980 in case of an EVENT. The instruction DATETIMESTRING can be used to convert it to a character string.
31	TS_TYPE	Returns the type of the timestamp in the case of an EVENT. 0 = Time stamped by the Supervisor. 1 = Time stamped at the source (using a time stamped protocol). 2 = Time stamped at the source but the timestamp is invalid or uncertain.
32	VARVALUE_REGISTER	Returns the value of the register variable in the case of an EVENT. Note that the value is returned as a DOUBLE.
33	VARVALUE_BIT	Returns the value of the bit variable in the case of an EVENT. 1 = On 0 = Off
34	VARVALUE_ALARM	Returns the value of the alarm variable in the case of an EVENT. 0 = Off ack 1 = On not ack 2 = Off not ack 3 = On ack 12 = Operator mask 13 = Program mask 14 = Variable mask 15 = Expression mask
35	VARVALUE_TEXT	Returns the value of the text variable in the case of an EVENT.

Example

For an example, select the Example link above.

GETPROJECTDIR

See Also

Returns the folder in which the project is located.

WebVue support - Yes.

Syntax

StrVal = GETPROJECTDIR();

The return type is STR.

Execution

The folder in which the project is located is returned as a string.

Example

```
SUB Main()
'Declare variables
DIM strPath as Str;

strPath = GETPROJECTDIR();
PRINT("The project path is: ",strPath);
END SUB
```

GETTREE

See Also

Returns the current branch of the program.

WebVue support - Yes.

Syntax

StrVal = GETTREE();

The return type is STR.

Execution

The branch used is the one passed as an argument to the function when it was called.

```
'A function that can only be used after TREE
SUB Main()
'Declare variables
DIM strBranch as Str;
DIM strTree as Str;

strBranche = "BRANCHE01";
TREE (strBranch);
strTree = GETTREE();
PRINT("The branch is: ",strTree);
propagationGetTree();
END SUB

SUB propagationGetTree()
DIM strTree as Str;

strTree = GETTREE();
PRINT("The branch is: ",strTree);
END SUB
```

GROUPALARM

See Also

Start or stop a selected group or all groups of alarms.

WebVue support - Yes.

The instruction is executed on the station that provides the group of alarms.

Mode	Mnemonic	Syntax
1	START	1
2	STOP	1

Syntax

IntVal = GROUPALARM(*Mode*, *Name*);

The return type is INTEGER.

Argument	Meaning
Name	Name of the alarm group. Type STR. Use '*' to indicate all groups.

Execution

Mode	Mnemonic	Action
1	START	Start all alarms in the group(s). Return: 1 if loaded, else 0.
2	STOP	Stop all alarms in the group(s). Return: 1 if successful, else 0.

H

HARDCOPY

See Also Example

Initiates a hard copy of the desktop client screen on the Windows default printer or clears its print jobs.

WebVue support - Not applicable. Returns the unsuccessful code if used in this context.

Mode	Mnemonic	Syntax
1	SCREEN	1
2	OPTION	2
4	PREVIOUSWINDOW	1
6	DELALL	1
7	DESKTOP	1

Syntax 1

IVAL = HARDCOPY (*Mode*);

Execution

Mode	Mnemonic	Action
1	SCREEN	Print a hard copy of the entire screen on the Windows default printer. Return: 1 if successful, else 0.
4	PREVIOUSWINDOW	Print a hard copy of the window that previously had focus, on the default printer. The window must still be open. Return: 1 if successful, else 0.
6	DELALL	Cancel and clear all print jobs in the default printer's spooler. Return: 1 if successful, else 0.
7	DESKTOP	Print the entire desktop when using a multi-region / multi-screen system. Return: 1 if successful, else 0.

The behavior of HARDCOPY can be affected by the operating system. If you experience difficulties contact Technical Support.

Syntax 2

IVAL = HARDCOPY (*Mode*, *Sub-mode*, *Value*);

Argument	Meaning
Sub-mode	Print according to *Value*. 1 Size: 1 Fit to page. 2 Use whole page. 3 Use scaling factors. 2 Apply scaling to the X axis. 3 Apply scaling to the Y axis. 4 Print via the named spooler. 5 Apply name to the printer dialog. 6 Message. 7 Exit. 8 Text for percentage, from 0 to 100. 9 Text for the Cancel button.
Value	As used in the printing actions above.

Execution

Mode	Mnemonic	Action
2	OPTION	Print a hard copy of the entire screen on the Windows default printer according to the sub-mode. Return: 1 if successful, else 0.

The embedded Hardcopy program

The Supervisor also has an in-built program known as Hardcopy. It is called from the Run.Program animation. The program does not appear in the Program Management window.

The embedded Hardcopy program has two functions:

- SCREEN – prints the entire screen.
- PREVIOUSWINDOW – prints only the most recently active window.

Running Hardcopy from the Macro animation

The Hardcopy instruction is also supported by the Macro animation.

Example

```
HARDCOPY("OPTION",1,3); 'scaling factors
HARDCOPY("OPTION",2,2); 'reduce to 50% on X axis
HARDCOPY("OPTION",3,2); 'reduce to 50% on Y axis
HARDCOPY("OPTION",5,"Color copy"); 'the dialog name
HARDCOPY("OPTION",6,"Copy in progress"); 'a message
HARDCOPY("OPTION",9,"Stop"); 'text for button
HARDCOPY("SCREEN"); 'print current screen
HARDCOPY("DELALL"); 'cancel print jobs
```

For a further example, select the Example link above.

HEX

See Also

Returns a string that represents, in hex, a number passed in base 10.

WebVue support - Yes.

Syntax

StrVal = HEX(*Num*);

The return type is STR.

Argument	Meaning
Num	The number to be converted. Any numeric type.

Execution

Return: A string that represents, in hex, the number passed in base 10.

Example

```
SUB Main()
DIM intNumBase10 as Integer;
DIM strNumberHex as Str;

intNumBase10 = 14;
strNumberHex = Hex (intNumBase10);
PRINT(intNumBase10, " to base 10 is equivalent to ",strNumberHex, " to base 16");
'Displays "14 to base 10 is equivalent to E to base 16"
END SUB
```

HISTORY

See Also Example

Export and import of historic trend data. Selection of primary and secondary archive units.

WebVue support - No.

Mode	Mnemonic	Syntax
1	GETTREND	1
2	IMPORTTREND	2
3	UNIT	3, 4, 5, 11, 12
4	TREND	6, 7, 8, 9, 10, 13

Syntax 1

LVAL = HISTORY (*Mode*, *VarName*, *TimeStart*, *TimeEnd*, *FileName* [,*FileMode*[, *Format*[, *StateVar*[, *NbVar*]]]]);

Argument	Meaning
VarName	The name of a variable for which the historic data is to be exported. Type STR.
TimeStart	The time and date for selection of the earliest data to be exported. Type DOUBLE. See instruction DATETIMEVALUEfor the conversion of time and date.
TimeEnd	The time and date for selection of the latest data to be exported. Type DOUBLE. See instruction DATETIMEVALUEfor the conversion of time and date.
FileName	The name of the file into which the data is exported. Type STR.
FileMode	The mode in which the file will be used. 0 creates the file, overwriting any previous versions (default value). 1 opens the file, and appends data to the end.
Format	A string defining the format in which the historic data will be exported. The following format elements may be used :

Element	Description
###Y	The year as 4 digits (example : 1994)
#Y	The year as 2 digits (example : 94)
#M	The month (01 to 12)
#D	The day (01 to 31)
#h	The hours (00 to 23)
#m	The minutes (00 to 59)
#s	The seconds (00 to 59)
##I	The milliseconds (000 to 999)
#V	The value of the variable

If *Format*is omitted or is a null string, the default format of *#Y#M#D#h#m#s##I, #V* will be used.

StateVar	The name of a database bit variable. When the export is started the bit is initialised to 0. At the end of the export the bit is set to 1. This allows a function to be activated at end of an export (see EVENT). If this facility is not required, the name of the variable may be replaced with a null string. Type STR.
NbVar	The name of a database register variable. At the end of an export it will contain the number of values exported. If this facility is not required the name of the variable may be replaced with a null string. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
1	GETTREND	Export of the historic record of a variable's value. Return: 0 if error, any other number OK.

Syntax 2

LVAL = HISTORY (*Mode*, *FileName*[, *StateVar* [, *NbVar*[, *FORMAT* [,*VARNAME*]]]]);

Argument	Meaning
FileName	The name of the file containing the data to be imported. Type STR.
StateVar	The name of a database bit variable. When the import is started the bit is initialised to 0. At the end of the import the bit is set to 1. This allows a function to be activated at the end of an import (see EVENT). If this facility is not required the name of the variable may be replaced with a null string. Type STR.
NbVar	The name of a database register variable. At the end of an import it will contain the number of values imported. If this facility is not required the name of the variable may be replaced with a null string. Type STR.
FORMAT	A string describing the expected format of data in the import file. If not supplied then the format must be specified in the import file itself. Type STR.
VARNAME	A string defining the name of the variable for which the historic data is to be imported. If not supplied the variable names must be specified in the import file itself. Type STR.

The return type is LONG.

Syntax of the Format String

The format string may be used either as part of the History instruction, or as a line in the import file itself. The syntax is as follows:

FORMAT,*Msflag*,*VarFlag*,*TSFormat*

Format String	Meaning
MsFlag	A flag indicating if milliseconds are used in the time stamp: 0 Milliseconds not used. 1 Milliseconds used.
VarFlag	A flag that specifying if the name of the variable is present on each line of the data in the import file: 0 No variable identifier. 1 The variable identifier is the variable name. 2 The variable identifier is the variable's internal ID.
TSFormat	The format of the time stamp in the import file. 0 YYMMDDhhmmss[ms] 1 MM/DD/YY:hh:mm:ss[:ms] 2 DD/MM/YY:hh:mm:ss[:ms] 3 YY/MM/DD:hh:mm:ss[:ms] 4 MM/DD/YYYY:hh:mm:ss[:ms] 5 DD/MM/YYYY:hh:mm:ss[:ms] 6 YYYY/MM/DD:hh:mm:ss[:ms] YYYY is the year as four digits YY is the year as two digits DD is the day (1 to 31) MM is the month (1 to 12)

hh is the hour (0 to 23)
mm is the minutes (0 to 59)
ss is the seconds (0 to 59)
ms is the milliseconds (0 to 999).

Syntax of the VARNAME String

The VARNAME string may be used either as part of the History instruction, or as a line in the import file itself. The syntax is as follows:

VARNAME,*Varname*

Argument	Meaning
VarName	The name of the variable for which the historic data is to be imported.

The Import File Format

The import file must be in ASCII format with each line being a comment, keyword or data.

Any line preceded by a single quotation mark (') or a hash (#) is treated as a comment and is not processed.

Any line starting with FORMAT is treated as a format string (see above).

Any line starting with VARNAME is treated as variable definition (see above).

All other lines are treated as data and must conform to the following syntax:

TimeStamp,*Value*[,*Varname*]

Data	Meaning
TimeStamp	The time stamp in the selected format.
Value	The value of the variable at that time.
VarName	The name of the variable for this line.

Execution

Mode	Mnemonic	Action
2	IMPORTTREND	Import of historic values for a variable.  To enable this, the check box Import Data File must be selected in the variable's configuration. Return: 0 if error, any other number OK.

Syntax 3

LVAL = HISTORY (*Mode*, *UnitName*, *Operation*);

Argument	Meaning
UnitName	The name of an archive unit. Type STR.
Operation	A string to select the type of operation. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
3	UNIT	Manipulation of the named archive unit. The string contained in the parameter *Operation* determines the

action:

Flush	Write (flush) the contents of all memory buffers to the archive unit files. Return: 0 if error, any other number OK.
Next	Continue recording in the next file in the sequence (Proprietary and free format archive units). Return: 0 if error, any other number OK.
Enable	Enable recording in the archive unit. (Only free format archive units). Return: 0 if error, any other number OK.
Disable	Disable recording in the archive unit. (Only free format archive units). Return: 0 if error, any other number OK.
Isreadonly	Check the read only property of the archive unit. (Proprietary and free format archive units). Return: -1 if error, 1 if read only, else 0.
Lock	Temporarily inhibit the archive unit Return: 0 if error, any other number OK.
Unlock	End a temporary inhibit applied to an archive unit using LOCK' Return: 0 if error, any other number OK.

Syntax 4

LVAL = HISTORY (*Mode*, *UnitName*, "READONLY", *Flag*);

Argument	Meaning
UnitName	The name of an archive unit. Type STR.
Flag	A flag to change the read-only property of the archive unit. Any numeric type.

The return type is LONG.

Execution

Mode	Mnemonic	Action
3	UNIT	Change the read-only property of the archive unit. If the flag is 0 then the read-only property is disabled (you can write to the archive unit). If the flag is 1 then the read-only property is enabled.

Syntax 5

LVAL = HISTORY (*Mode*, *UnitName*, *Operation* [, *StateVar*, *Flag*]);

Argument	Meaning
UnitName	The name of an archive unit. Type STR.
Operation	A string to select the type of operation. Type STR.
StateVar	The name of a bit variable. Type STR.
Flag	A flag (0 or 1)

The return type is LONG.

Execution

Mode	Mnemonic	Action
3	UNIT	Manipulation of the named archive unit. The string contained in the parameter "Operation" determines the action. See syntax 3 for details. The bit variable *StateVar*will be set to the value of the flag when the operation is complete.

Syntax 6

LVAL = HISTORY (*Mode*, "SETPRIMARY", *VarName*, *UnitName*);

Argument	Meaning
VarName	The name of a variable that has been configured to record trend data using two or more archive units. Type STR.
UnitName	The name of one of the secondary archive units for the named variable. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	TREND	Selects one of the secondary archive units for a variable configured to record data in two or more archive units. When the Supervisor extracts data for this variable (for use in a Trend Viewer for example) the data is taken from the secondary archive unit. Return: 0 if error, else OK.

This is a temporary change and is not saved when the Supervisor is shut down.

Syntax 7

LVAL = HISTORY (Mode, "CLEARPRIMARY", VarName;

Argument	Meaning
VarName	The name of a variable that has been configured to record trend data using two or more archive units. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	TREND	Selects the primary unit from the variable configuration for a variable configured to record data in two or more archive units. When the Supervisor extracts data for this variable (for use in a trend display for example) the data is taken from the primary archive unit. Return: 0 if error, else OK.

Syntax 8

LVAL = HISTORY (*Mode*, "SETPRIMARY_UNIT", *PrimaryUnit*, *SecondaryUnit*);

Argument	Meaning
PrimaryUnit	The name of the primary archive unit. Type STR.
SecondaryUnit	The name of a secondary archive units. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	TREND	Selects one of the secondary archive units for all variables having the named primary and secondary archive units. When the Supervisor extracts data for any of these variables (e.g. for use in a Trend Viewer) the data is taken from the secondary archive unit. Return: 0 if error, else OK.

This is a temporary change and is not saved when the Supervisor is shut down.

Syntax 9

LVAL = HISTORY (*Mode*, "CLEARPRIMARY_UNIT", *PrimaryUnit*);

Argument	Meaning
PrimaryUnit	The name of the primary archive unit. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	TREND	Selects the primary unit for all variables having the named primary unit. When the Supervisor extracts data for this variable (for use in a Trend Viewer for example) the data is taken from the primary archive unit. Return: 0 if error, else OK.

Syntax 10

LVAL = HISTORY (*Mode*, "SAMPLING", *VarName*, *UnitName*, *SampleRate*);

Argument	Meaning
VarName	The name of a variable that has been configured to record trend data. Type STR.
UnitName	The name of one of the archive units for the named variable. Type STR.
SampleRate	A period from 0 to 32000 seconds.

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	TREND	Allows the sample rate for recording data for the named variable and archive unit to be dynamically changed. Return: 0 if error, else OK.

This is a temporary change and is not saved when the Supervisor is shut down.

Syntax 11

LVAL = HISTORY (*Mode*, *UnitName*, *Sub-mode*, *ServerName*);

Argument	Meaning
UnitName	The name of the archive unit that is to be redirected. Type STR.
ServerName	Name of the station to which the local station's access requests are to be directed. The station must be part of a historical server association.

The return type is LONG.

Execution

Mode	Mnemonic	Sub-mode	Mnemonic	Action
3	UNIT	10	SETREADSERVER	Dynamically configure which of the two stations in an historic association a historic client reads from. Return: 0 if error, 1 if OK.

Changes are not saved when the Supervisor closes. On starting again, it uses the original configuration.

If a client station makes a request to read from server station 1 which itself is redirected to server station 2, the client station's request will be handled by server station 2.

It is vital to understand the mechanism for redirecting read requests so as to avoid configurations that loop.

For example where server 1 is redirected to server 2 which itself is redirected to server 1.

Syntax 12

LVAL = HISTORY (*Mode*, *UnitName*, *Sub-mode*, *RestoreReadServer*);

Argument	Meaning
UnitName	The name of the archive unit that is to be restored. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Sub-mode	Mnemonic	Action
3	UNIT	11	RESTOREREADSERVER	Restore the archive unit's original configuration. Return: 0 if error, 1 if OK.

Syntax 13

LVAL = HISTORY (*Mode*, "FORCEPRIMARY_UNIT", *PrimaryUnit*);

Argument	Meaning
PrimaryUnit	The name of the archive unit to which the historical data is recorded. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Sub-mode	Mnemonic	Action
4	TREND	-	FORCEPRIMARY_UNIT	Specifies the primary archive unit for reading. Return: 0 if error, 1 if OK.

Changes are not saved when the Supervisor closes. On starting again, it uses the original configuration.

Using SETPRIMARY_UNIT/SETPRIMARY and FORCEPRIMARY

These enable choice from a client station of the unit that is to be used on a server.

With Supervisor version 8.10 onwards, when there is a Trend request between stations the primary unit's identity is transmitted to the producer of the historical data.

Example

For an example, select the Example link above.

I

IF...THEN...ELSE...END IF

See Also

Conditional execution of instructions according to the result of a logical expression.

WebVue support - Yes.

Mnemonic	Syntax
IF...THEN...	1
IF...THEN...ELSE...END IF	2

Syntax 1

IF(*Condition*)THEN

[instruction block 1]

END IF

There is no return type.

Argument	Meaning
Condition	A logical expression, for example: TankLevel > 25.

Execution

If the condition is true then instruction block 1 is executed after which the program continues at the line after the END IF.

Syntax 2

IF(*Condition*)THEN

[instruction block 1]

ELSE

[instruction block 2]

END IF

There is no return type.

Execution

If the condition is true then instruction block 1 is executed else instruction block 2 is executed after which the program continues at the line after the END IF.

Example

```
SUB Main()
DIM i as Integer;

i=0;
'or: i=1;
If ( i==1 )Then
  Print("i=1");
Else
  Print("i is not 1");
End If
END SUB
```

IGET_BUFFER

See Also

Read an integer located in a memory area.

WebVue support - Yes.

Syntax

IntVal = IGET_BUFFER(*Handle*, *Offset*);

The return type is INTEGER.

Argument	Meaning
Handle	The location of the memory buffer as returned by ALLOC_BUFFER. Type LONG.
Offset	The offset in bytes at which the integer is to be found. Any numeric type.

Execution

Return: An integer located in a memory area reserved by ALLOC_BUFFER or FILETOBUF.

Example

```
i1 = IGET_BUFFER(handle,2);
```

IRAND

See Also

Return a random number.

WebVue support - Yes.

Syntax

IntVal = IRAND([*MinVal*, *MaxVal*]);

The return type is INTEGER.

Argument	Meaning
MinVal	The minimum value for the random number to be returned. Any numeric type.
MaxVal	The maximum value for the random number to be returned. Any numeric type.

Execution

Return: A random integer value. If *MinVal* and *MaxVal* are not supplied, the value will lie in the range 0 to 32767.

Example

```
SUB Main()
DIM intRandValue as integer;

intRandValue = IRAND();
PRINT(intRandValue);
intRandValue = IRAND(0,10);
PRINT(intRandValue);
END SUB
```

IVAL

See Also

Convert a string to an integer.

WebVue support - Yes.

Syntax

IntVal = IVAL(*string*);

The return type is INTEGER.

Execution

The string is converted into the equivalent integer.

If any non-numeric characters exist in the string, the numeric characters before them are converted. If the string starts with a non-numeric character, the return is 0.

Example

```
SUB Main()
DIM intResult as Integer;
DIM strString1 as Str;

strString1 = "125.35TEST";
intResult = IVAL( strString1 );
PRINT("The result is:", intResult );
END SUB
```

J

(No topics)

There are no topics in this book.

K

KEY

See Also Key codes

Program the keyboard function keys on the desktop client.

WebVue support - Not applicable. Returns the unsuccessful code if used in this context.

Mode	Mnemonic	Syntax
1	ADDCLK	1
2	ADDSTD	2
3	ADDPROG	3
5	DEL	4
6	DELALL	5
11	ADDCLKS	1
12	ADDSTDS	2
13	ADDPROGS	3
21	DELSTD	6
22	DELCLK	7
23	DELPROG	8

Changes in function key management from version 12 onwards

Prior to version 12 the default configuration of the function keys with pre-configured standard actions was created automatically each time the Supervisor started. Any changes made as part of the project configuration were overwritten. This meant that if you wanted to delete or change the function of any of these keys permanently, it had to be done using a SCADA Basic program arranged to run each time the Supervisor was started.

From version 12 onwards the default configuration of the function keys with pre-configured standard actions is created once only when the project is first created. Therefore any changes made subsequently, as part of the project configuration, are permanent.

If you start a project created with a version prior to 12 the behavior is the same as if you were creating a new project. That is the default configuration of the function keys with pre-configured standard actions is created once only. A SCADA Basic program to delete or change the function of these keys is no longer necessary.

Syntax 1

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*, *Window*, *Branch*, *Sequence* [, *Activbit*]);

The return type is INTEGER

Argument	Meaning
Activbit	Optional. The name of a bit or alarm variable that enables the event. Type STR.

Execution

Mode	Mnemonic	Action
1	ADDCLK	Program a function key to operate a control zone on a window. The sequence number corresponds to the sequence of the control zone in the window.
11	ADDCLKS	Add another function to a key, without deleting any previous functions. Return: 0 if successful, else 1.

Syntax 2

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*, *Action*

[, *Activbit*]);

The return type is INTEGER.

Argument	Meaning
Activbit	The name of a bit or alarm variable which enables the event.

Execution

Mode	Mnemonic	Action
2	ADDSTD	Program a standard action. Cancel and replace the previous control on the same key. Return: 1 if successful, else 0.
12	ADDSTDS	Add another function to a key, without deleting any previous functions. Return: 0 if successful, else 1.

Syntax 3

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*, *Program*, *Branch*, *Function*, [*farg* [, *activbit*]]);

The return type is INTEGER.

Argument	Meaning
Farg	Optional. String of 2,047 characters maximum. Used to pass from 1 to 8 arguments (separated by a comma) to the function.
Activbit	Optional. The name of a bit or alarm variable which enables the event.

Execution

Mode	Mnemonic	Action
3	ADDPROG	Execute a function from a program. If the name of the function is not given then the MAIN routine will run. Return: 1 if successful, else 0.
13	ADDPROGS	Add another function to a key, without deleting any previous functions. Return: 0 if successful, else 1.

Syntax 4

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
5	DEL	Remove all actions from the specified key. Return: 1 if successful, else 0.

Syntax 5

IntVal = KEY(*Mode*) ;

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	DELALL	Remove all actions from all keys. The standard actions will return after a key action is removed. Return: 0 if successful, else 1.

Syntax 6

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*, *Window*, *Branch*, *Sequence*);

Argument	Meaning
ShiftType	Description of the shift type.
KeyCode	Code of the key.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
21	DELCLK	Remove control specified by the key of the control zone. Return: 0 if successful, else 1.

Syntax 7

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*, *action*);

Argument	Meaning
ShiftType	Description of the shift type.
KeyCode	Code of the key.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
22	DELSTD	Remove the action specified. Return: 0 if successful, else 1.

Syntax 8

IntVal = KEY(*Mode*, *ShiftType*, *KeyCode*, *Program*, *Branch*, *Function*);

Argument	Meaning
ShiftType	Description of the shift type.
KeyCode	Code of the key.
Program	The name of the program where the function is.
Branch	The name of the branch given as an argument to the function when called.
Function	The name of the function to be removed.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
23	DELPROG	Remove the specified action. Return: 0 if successful, else 1.

To call a function, the program must be pre-loaded.

If the MAIN function is used then no DELAY instructions should be used or the results may be unpredictable.

Coding for the *ShiftType* argument

The parameter *ShiftType* is a string describing the key behaviour as follows:

Value	Behaviour
D	The action is executed when the key is pressed (otherwise releasing the key executes the action).
S	The action is executed when the key and the Shift key are pressed together.
C	The action is executed when the key and the Control key are pressed together.

Characters may be combined, for example SC would mean pressing the control and shift key together.

The standard behaviors of keys are listed in the topic Key Codes.

Example

```
KEY(2,"D" ,2, 2); ' F2
KEY(2,"D" ,3, 3); ' F3
KEY(2,"D" ,4, 4); ' F4
KEY(2,"D" ,5, 5); ' F5
KEY(2,"D" ,6, 6); ' F6
KEY(2,"D" ,7, 7); ' F7
KEY(2,"D" ,8, 8); ' F8
KEY(2,"D" ,9, 9); ' F9
KEY(2,"DS",10, 10); ' F10
KEY(2,"D" ,11, 11); ' F11
KEY(2,"" ,27, 21); ' PageDn
KEY(2,"" ,26, 22); ' PageUp
KEY(2,"" ,32, 23); ' ArrowRt
KEY(2,"" ,33, 23); ' ArrowDn
KEY(2,"" ,30, 24); ' ArrowLt
KEY(2,"" ,31, 24); ' ArrowUp
KEY(2,"" ,29, 25); ' Home
KEY(2,"" ,28, 26); ' End
```

L

LAN

See Also Example

See also the Help book on Networked Applications.

The SCADA BASIC instruction LAN is used to control the network status of a station in distributed and redundant applications.

If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end which must be a Supervisor station within the distributed application.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	CONNECT	1
2	PRODUCT_MODE	2
3	GET_LISTS	3
4	GET_STATIONS_INLIST	4
5	SET_SERVER_MODE	5
6	GET_STATION_NUMBER	6
7	GET_STATION_NAME	7
10	SET_AVAILABLE_RATE	9
11	SET_READ_SERVER	10
12	RESET_READ_SERVER	11
13	START_CONNECTIONS	12
14	STOP_CONNECTIONS	12
15	START_CONNECTION	13
16	STOP_CONNECTION	13
17	SET_ACTIVE_NODE	14

From version 7.20a onwards, Mode 2 - PRODUCT_MODE should no longer be used. Mode 5 - SET_SERVER_MODE should be used instead.

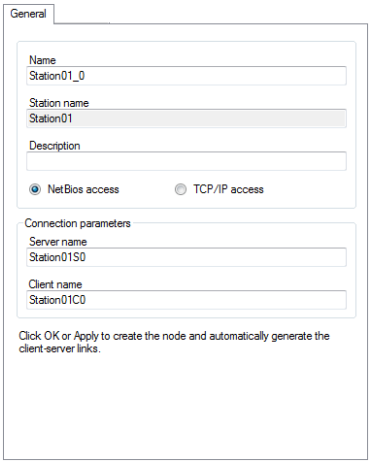
Syntax 1

IntVal = LAN (*Mode*, *RmtSvrCx*);

The return type is INTEGER.

Argument	Meaning
RmtSvrCx	The name of the connection to a remote server in the distributed application as in the Advanced Properties of the network configuration. In the picture below, the server connection name is Station0150. Type STR.

Show picture



Execution

Mode	Mnemonic	Action
1	CONNECT	The station that runs the program containing the LAN instruction is connected to the specified server. The station running the program must be configured as a client. Return: 1 if OK, else 0.

The CONNECT mode can only be used if the station is in a network association.

Syntax 2

IntVal = LAN (*Mode*, *AssocNo*, *Flag*);

The return type is INTEGER.

Argument	Meaning
AssocNo	The association number as specified in the network configuration.
Flag	A flag indicating the new status of the server. 1 The server becomes active. 0 The server becomes passive.

Execution

Mode	Mnemonic	Action
2	PRODUCT_MODE	The status of the station running the program is switched to the requested state (active or passive). When a server station is switched to passive, any Equipment, DDE or Internal variables are temporarily changed to External, and their value comes from the active server. When a server station is switched to active any variables that were temporarily changed to External return to their original state. Return: 1 if OK, else 0.

The database bit variable SYSTEM.SERVER.Association Name.Station Name is set to 1 when the local station is an active server, and 0 when it is passive

Syntax 3

BuffList = LAN (*Mode*, *ListType*, *StationName*);

Argument	Meaning
ListType	Type INTEGER: 1 = Server list 2 = Client list 3 = Other list
StationName	The station name for selection. (Type STR)

The return type is Long.

Execution

Mode	Mnemonic	Action
3	GET_LISTS	Fill *BuffList*with all the lists of type *ListType*containing *StationName*. Return: The handle of the buffer.

Format of *BuffList*: list1,list2,list3....

The buffer *BuffList*will be allocated automatically. Remember to release the buffer via FREE_BUFFER.

Syntax 4

BuffList = LAN(*Mode*, *ListType*, *ListName*);

Argument	Meaning
ListType	Type INTEGER: 1 = Server list 2 = Client list 3 = Other list
ListName	The name of the list to be selected. Type STR.

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	GET_STATIONS_INLIST	Fills the buffer identified by *BuffList*with the station identifiers in the list *List Name*.

Format of *BuffList*: list1,list2,list3....

The buffer *Bufflist*will be allocated automatically Remember to release the buffer via FREE_BUFFER.

Syntax 5

IntVal = LAN(*Mode*, *AssocName*, *Servername*, *ServerMode*);

Argument	Meaning
AssocName	Association name.
Servername	Name of the server station to be switched between active and passive states.
ServerMode	Required state: SET_SERVER_ACTIF = active SET_SERVER_PASSIF = passive

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
5	SET_SERVER_MODE	This mode makes it possible to switch the server station to the active state only if that station belongs to a single active server association. When the station switches out of the passive server state, the variables of the types internal, equipment, DDE, OPC and LON produced by the association temporarily become external and subscribe on the active, remote server station. When the station switches out of the active server state, those temporarily external variables stop subscribing and revert their original type.

Response 1 if OK, 0 if not.

The specialized bit variable SYSTEM.Association_Name.Station_Name has the value 1 when the server station is active, 0 when it is passive.

This only works if the Availability Rate is the same in both historic servers.

For compatibility, the variables that do not have a list of server stations also change.

The internal variables that start with SYSTEM do not switch.

When the station that is active is forced passive, the station in the association (apart from the station that is forced) which has the lowest number will be forced active.

Syntax 6

IntVal = LAN(*Mode*, *StationName* [, *AssocFlag*]);

Argument	Meaning
StationName	Name of the station.
AssocFlag	What to return if the station name corresponds to an association name: (Optional) 0 Return the station number, else 1 Return the association number if the station number is not found.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	GET_STATION_NUMBER	Provides the station number from its name. If the name does not exist, the result is 0.

Response: station number if OK, else 0.

Syntax 7

StrVal = LAN(*Mode*, *StationNumber, AssocFlag*);

Argument	Meaning
StationNumber	Number of the station.
AssocFlag	What to return if the station number corresponds to an association number: (Optional) 0 Return the station name, else 1 Return the association name if the station name is not found.

The return type is STR.

Execution

Mode	Mnemonic	Action
7	GET_STATION_NAME	Provides the station name from its number. If the number does not exist, the result is an empty string.

Response: station name if OK, else an empty string.

Syntax 9

IntVal = LAN (*Mode*, *AssocName, ServerName*, *AvailableRate*);

The return type is INTEGER.

Argument	Meaning
AssocName	Name of the association.
ServerName	Name of the server station in the association.

AvailableRate Availability rate (between 0 and 100). Type INTEGER

Execution

Mode	Mnemonic	Action
10	SET_AVAILABLE_RATE	The availability rate of a server station that is part of an association can be modified dynamically.  The change does not persist since it does not involve any change of configuration. For a historical association, this value may be changed when it is anticipated that a historical server will be slowed down by a maintenance procedure (e.g. purge). For a real time association it may be used for example to set a server in main mode. This mode provides for the situation where two stations of an association with a single active producer have been disconnected. Both are active but when they are re-connected, only the priority station remains active. If all stations are at the same availability rate, the one with the lower station number remains active. Otherwise it's the station with the biggest availability rate that remains active. This mode can be run on any station. Return: 1 if OK, else 0.

Syntax 10

IntVal = LAN (*Mode*, *AssocName, StationName* [, *ClientStationName*]);

The return type is INTEGER.

Argument	Meaning
AssocName	The name of a historic association.
StationName	The name of one of the servers in historic association.
ClientStationName	The name of a client for the historical data produced by the association *AssocName*.

Execution

Mode	Mnemonic	Action
11	SET_READ_SERVER	The server station in an historic association can be selected dynamically to process read requests for a client station. This mode can be used to redistribute the workload. The active server station for the client station *ClientStationName* in the association *AssocName* will then be *StationName* regardless of the availability rate of servers in the association. Return: 1 if OK, else 0.

Syntax 11

IntVal = LAN (*Mode*, *AssocName, StationName*[, *ClientStationName*]);

The return type is INTEGER.

Argument	Meaning
AssocName	Name of the association.
StationName	The name of one of the servers in historic association.
ClientStationName	The name of a client for the historical data produced by the association *AssocName*. If *ClientStationName* is not given, the local station that runs the instruction is used.

Execution

Mode	Mnemonic	Action
12	RESET_READ_SERVER	The server station *StationName*for the client station *ClientStationName*in the association *AssocName* is no longer forced to be the active server. Whichever server has the higher availability rate will be active. Return: 1 if OK, else 0.

Syntax 12

IntVal = LAN (*Mode*, *RemoteStation*);

The return type is INTEGER.

Argument	Meaning
RemoteStation	Name of the remote station in the distributed application (as specified in the multi-station configurator). Type STR

Execution

Mode	Mnemonic	Action
13	START_CONNECTIONS	If the station running the program has a connection configured with the specified remote station, it connects to that station. If the station running the program is configured as a server, it accepts connections from the specified remote station. Return: 1 if OK, else 0.
14	STOP_CONNECTIONS	If the station running the program has a connection configured with the specified remote station, it disconnects from that station. If the station running the program is configured as a server, it no longer accepts connections from the specified remote station. Return: 1 if OK, else 0.

Syntax 13

IntVal = LAN (*Mode*, *RemoteStation, RemoteCnt*);

The return type is INTEGER.

Argument	Meaning
RemoteStation	Name of the remote station in the distributed application (as specified in the multi-station configurator). Type STR.
RemoteCnt	Name of the Server or Client connection of the remote station *RemoteStation* (as specified in the multi-station configurator). Type STR.

Execution

Mode	Mnemonic	Action
15	START_CONNECTION	If *RemoteCnt* is a server connection and the station running the program has a link configured with the specified remote station, it connects to that station using the *RemoteStation* connection. If *RemoteCnt* is a client connection and the station running the program is configured as a server, it connects to that station via the *RemoteStation* connection. Return: 1 if OK, else 0.
16	STOP_CONNECTION	If *RemoteCnt* is a server connection and the station running the program has a connection configured with the specified remote station, it disconnects from the *RemoteStation* connection. If *RemoteCnt* is a client connection and the station running the program is configured as a server, it no longer accepts connections from the specified

remote station via the *RemoteStation* connection.
Return: 1 if OK, else 0.

Syntax 14

IntVal = LAN (*Mode SourceNodeName*[, *DestinationNodeName*]);

The return type is INTEGER.

Argument	Meaning
SourceNodeName	Name of the source node. Type STR.
DestinationNodeName	Name of the destination node, unless blank in which case all stations are affected. Type STR.

Execution

Mode	Mnemonic	Action
17	SET_ACTIVE_NODE	Enables selection of the active node for a particular station or all stations to which it is connected. This mode can be run from any station. The following system variables are updated: SYSTEM.*ClientConnectionNodeName*.ACTIVECXT SYSTEM.*ServerConnectionNodeName*.ACTIVECXT SYSTEM.*NodeName*.ACTIVECXT SYSTEM.*ClientConnectionNodeName*.ACTIVE_NUMBER SYSTEM.*ServerConnectionNodeName*.ACTIVE_NUMBER SYSTEM.*NodeName*. ACTIVE_NUMBER Return: 1 if OK, else 0.

Example

There are three stations with each two nodes.

- SERVER1 with nodes SERVER1_0 and SERVER1_1
- SERVER2 with nodes SERVER2_0 and SERVER2_1
- CLIENT1 with nodes CLIENT1_0 and CLIENT1_1

The instruction to make node 0 of station SERVER1 active for all stations is:

```
LAN( "SET_ACTIVE_NODE", "SERVER1_0");
```

The instruction to make node 0 of station SERVER1 active only for node 0 of station SERVER2 is:

```
LAN( "SET_ACTIVE_NODE", "SERVER1_0", "SERVER2_0");
```

LANGUAGE

See Also Example

Change the operating language.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	WebVue
0	GET	1	Yes
1	SET_L1	2	No
2	SET_L2	2	No
3	TOGGLE	2	No

Syntax 1

IntVal = LANGUAGE(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	GET	Returns the language code currently used. Return: 1 for the primary language, 2 for the alternative language.

Syntax 2

IntVal = LANGUAGE(*Mode* [, *Refresh*]);

Refresh Optional argument:
1 - refresh all visible windows.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	SET_L1	Switch to primary language. Return: 1 if successful, else 0.
2	SET_L2	Switch to alternative language. Return: 1 if successful, else 0.
3	TOGGLE	Toggle between languages. Return: 1 if successful, else 0.

Example

```
SUB L2()
LANGUAGE("SET_L2", 1); 'switch language and refresh windows
END SUB
```

For a larger example, select the Example link above.

LCASE

See Also

Convert the characters of a string to lower case.

WebVue support - Yes.

Syntax

StrVal = LCASE(*Input*);

The return type is STR.

Argument	Meaning
Input	The string to be converted. Type STR.

Example

```
SUB Main()
DIM strResult as Str;
DIM strString as Str;

strString = "STRING IN CAPITALS";
strResult = LCase( strString );
PRINT("Result is: ", strResult );
'displays "Result is: string in capitals"
END SUB
```

LEFT

See Also

Return a number of characters from the beginning (left hand end) of a string.

WebVue support - Yes.

Syntax

StrVal = LEFT(*Input*, *N*);

The return type is STR.

Argument	Meaning
Input	The string to be manipulated. Type STR.
N	The number of characters to return. Any numeric type.

Execution

If N is greater than the length of the string then the entire string is returned.

Example

```
SUB Main()
DIM strResult as Str;
DIM strString as Str;

strString = "Hello, World!";
strResult = LEFT( strString ,5); 'takes the first 5 characters
PRINT("Result is: ", strResult );
'Displays "Result is: Hello"
END SUB
```

LEN

See Also

Return the length of a string.

WebVue support - Yes.

Syntax

IntVal = LEN(*InputString*);

The return type is INTEGER.

Argument	Meaning
InputString	The string whose length is to be returned. Type STR.

Example

```
SUB Main()
DIM intResult as INTEGER;
DIM strString as STR;

strString = "Hello, World!";
intResult = LEN(strString);
PRINT("Result: ", intResult );
'Displays "Result: 13"
END SUB
```

LGET_BUFFER

See Also

Read a LONG located in a memory area.

WebVue support - Yes.

Syntax

LongVal = LGET_BUFFER(*handle*, *Offset*);

The return type is LONG.

Argument	Meaning
Handle	The location of the memory buffer as returned by ALLOC_BUFFER. Type LONG.
Offset	The offset in bytes at which the LONG is to be found. Any numeric type.

Execution

Return: A LONG located in a memory buffer reserved by ALLOC_BUFFER.

Example

```
i1 = LGET_BUFFER(handle, 2);
```

LISTBOX

See Also Example

Access some properties of the Supervisor's List-box form control.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	WebVue
1	COUNT	1	Yes
2	GETSELECTEDINDEX	1	Yes
3	SETSELECTEDINDEX	2	Yes
4	GETTEXT	3	Yes
5	GETUSERDATA	3	Yes
6	LOAD	4	Yes
7	INSERT	5	No
8	REMOVE	6	No
9	SORT	7	No
10	CLEAR	1	No

All syntaxes

Argument	Meaning
Window	The name of the window that contains the form control. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR.
Identity	The identity of the form control within the specified window. Type STR.

Syntax 1

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*);

The return type is LONG.

Execution

Mode	Mnemonic	Action
1	COUNT	Returns the number of items. Type LONG.
2	GETSELECTEDINDEX	Returns the index of the item currently selected. Type LONG.
10	CLEAR	Clears the list contents and any selection. Return: 1 if successful, else 0.

Syntax 2

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Index*, *Notification*);

Argument	Meaning
Index	The index of the item to select. Type LONG. If the index is -1 then no item is selected.
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control. Type INTEGER. The function will scroll the control if necessary to display the item.

The return type is type LONG.

Execution

Mode	Mnemonic	Action

3	SETSELECTEDINDEX	Select an item according to its index. Type LONG. Return: 1 if successful, else 0.

Syntax 3

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of the item whose text is to be retrieved.

The return type is STR.

Execution

Mode	Mnemonic	Action
4	GETTEXT	Returns the Text of the item as defined for the language currently in use. Type ST
5	GETUSERDATA	Returns the User Data associated with the item. Type STR.

Syntax 4

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*, *FileName*);

Argument	Meaning
FileName	The name of the file that contains the form control's data.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	LOAD	Loads the file's content into the form control's list of items. Type INTEGER. Return: 1 if successful (file loaded), else 0.

Syntax 5

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Text*, *Userdata*, [*Index*]);

Argument	Meaning
Text	The text to insert. Type STR.
Userdata	The user data to insert. Type STR.
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
7	INSERT	Inserts the Text and optional Userdata in the form control list. If Index is omitted text is inserted at the start. If Index is -1 the text is inserted at the end. Otherwis text is inserted at the position indicated by Index. Return: 1 if successful, else 0.

Syntax 6

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
8	REMOVE	Removes the selected item from the list. Return: 1 if successful, else 0.

Syntax 7

Return = LISTBOX(*Mode*, *Window*, *Branch*, *Identity*[, *Sortorder*, *Orderby*]);

Argument	Meaning
Sortorder	A flag indication the sort order. 0 = ascending, 1 = descending
Orderby	A flag selecting which data to sort by. 0 = Tex, 1 = Userdata.

The return type is LONG.

Execution

Mode	Mnemonic	Action
9	SORT	Sort the form control list using the specified criteria. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

LOG

See Also

Natural logarithm function (base e).

WebVue support - Yes.

Syntax

DblVal = LOG(*Value*);

The return type is DOUBLE.

Argument	Meaning
Value	The value to be converted. Any numeric type.

Execution

Return: The natural logarithm of the value.

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
SUB Main()
'Declare variables
DIM dblExponential as double;
DIM sngValue as single;
DIM dblResult as double;

sngValue = 2.23456;
dblExponential = Exp ( sngValue );
dblResult = Log(dblExponential);
PRINT("la valeur Log(",dblExponential,") = ",dblResult);
'Displays "The value of Log(9.34237) = 2.23456"
END SUB
```

LOGDISPLAY

See Also Example Further information

Log Viewer management including list navigation and filtering.

Partial WebVue support - see mode table.

If using a project with multiple regions, you must set the region before executing any instructions that interacts with the HMI. For further information see the REGION topic.

Mode	Mnemonic	Syntax	WebVue
0	BEGIN	1	Yes
1	BEFORE	1	Yes
3	AFTER	1	Yes
4	END	1	Yes
9	DOMAIN	2	Yes
10	NATURE	2	Yes
11	PRINTALL	1, 8	No
12	SETDATETIME	3, 7	No
13	FILTER	4	Yes
14	PRINTSELECTED	1, 8	No
15	PRINTDISPLAY	1, 8	No
16	FIRST	1	Yes
17	LAST	1	Yes
18	EVENTMASK	6	Yes
19	MINMAX	5	Yes
20	EXECUTE	1	Yes
21	LOGLIST	2	Yes
22	EVENTMASKEX	12	Yes
23	LINESELECT	9	Yes
24	SETSORT	10	Yes
25	GETSORT	11	
26	AUTOREFRESH_SET	17	Yes
27	AUTOREFRESH_PAUSE	1	Yes
28	AUTOREFRESH_RESUME	1	Yes
29	ISLINESELECTED	13	No
30	ISLINEVISIBLE	13	No
31	GETLINECOUNT	1	No
32	COPY_CLIPBOARD	14	No
33	GETLINES	15	No
34	GETSELECTEDLINES	15	No
35	GETCELL	16	No
36	AUTOREFRESH_STATUS	1	No
37	AUTOREFRESH_PERIOD	1	No

* When used with WebVue there is no difference between Logon and Logoff when filtering using EVENTMASK or EVENTMASKEX.

Common arguments

Argument	Meaning
Window	The name of the Window in which the animation is embedded. Type STR
Branch	The window branch. Use "*" for the branch of the current program. Type STR.
Identity	Identity of the Log Viewer (from its configuration). Type STR
Context	A flag indicating if the instruction is to be executed immediately (0) or whether it is to wait for an Execute instruction (1).

Using the Context argument and the Execute mode

Many of the modes support an optional context flag. If the context flag is used, and set to 1, the changes that are to be made to the Log Viewer filter using the instruction are delayed until an EXECUTE mode is encountered. This allows you to make several changes to a Log Viewer filter which are then all executed at the same time. As each time a Log Viewer filter is changed a new historical request is made, this avoids making multiple historical requests where one would have been sufficient.

The following modes support the context flag.

DOMAIN, NATURE, LOGLIST, SETDATETIME, FILTER, MINMAX, EVTMASK and EVENTMASKEX.

Syntax 1

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	BEGIN	Request and display the previous buffer of data. Equivalent to the << butt the Log Viewer.
1	BEFORE	Scroll the list backwards by one page. Equivalent to the < button on the Lo Viewer.
3	AFTER	Scroll list forwards by one page. Equivalent to the > button on the Log Vie
4	END	Request and display the next buffer of data. Equivalent to the >> button o Log Viewer.
11	PRINTALL	Print the entire contents of the current buffer on the printer configured in t Viewer.
14	PRINTSELECTED	Print the selected line on the printer configured in the Log Viewer.
15	PRINTDISPLAY	Print all the lines currently visible in the Log Viewer.
16	FIRST	Display the first (oldest) record from the archive file. Equivalent to the |< on the Log Viewer.
17	LAST	Display the last (newest) record from the archive file. Equivalent to the >| on the Log Viewer.
20	EXECUTE	Execute the run-time context set up by commands of type MINMAX, EVENT
27	AUTOREFRESH_PAUSE	Pause the auto-refresh mechanism.
28	AUTOREFRESH_RESUME	Resume the auto-refresh mechanism.
31	GETLINECOUNT	Return the number of lines in the Log Viewer.
36	AUTOREFRESH_STATUS	Return the status of the auto-refresh mechanism. 1 = Running 2 = Paused 3 = Not set (period is 0) else 0 (window does not exist etc.).
37	AUTOREFRESH_PERIOD	Return the auto-refresh period in seconds.
		Return: 1 if OK, else 0 (window does not exist etc.). Except for modes 31, 37.

Syntax 2

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Argument* [,*Context*]);

Argument	Meaning
Argument	The name of the new domain, nature or log list. To specify all the domains and natures, *Argument*must be a null string. Type STR.

The return type is INTEGER.

Execution

Selects the new domain, nature or log list.

Mode	Mnemonic	Action
9	DOMAIN	New domain.

10	NATURE	New nature. Return: 1 if OK, 0 if not.
21	LOGLIST	New log list. Return: 1 if OK, 0 if not

If the filter animation of the Log Viewer is configured with domain and/or nature, the FILTER mode of LOGDISPLAY will be ignored.

Syntax 3

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*[,StartTime [,*EndTime* [,*Context*]]]);

Argument	Meaning
StartTime	The new start time for the Log Viewer, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE. If omitted then the current time is used.
EndTime	The new end time for the Log Viewer, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE. If omitted then the current time is used.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
12	SETDATETIME	Selects a new time for the Log Viewer Return: 1 if successful, else 0.

The maximum number of lines for a Log Viewer is 32,000.

Syntax 4

ret = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Filter [,Context]*);

Argument	Meaning
Filter	A filter expression. Type STR.

The return type is INTEGER.

See the topic Native Filter Expressions for information on native filter expressions.

Execution

Mode	Mnemonic	Action
13	FILTER	Selects information displayed in the log viewer by applying a filter expression. • For a proprietary archive unit the filter must be a native filter expression. • For a database archive unit the filter can either be a native filter expression or an SQL Expression. For example TextAttr11 IS NOT NULL. Return: 1 if successful, else 0 (the window does not exist, etc.).

If the filter of the log list has been configured with a domain or nature, the LOGDISPLAY mode FILTER will not be taken into account.

If the filter is on #T and the log list is in a database archive unit, you must include the column Description in the log table otherwise the result is an empty viewer.

Syntax 5

ret = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Min*, *Max*, *[Context]*);

Argument	Meaning
Min	The minimum priority level of alarms. Type INTEGER.
Max	The maximum priority level of alarms. Type INTEGER.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
19	MINMAX	Selects information displayed in the log window by applying a minimum and maximum priority level for alarms. Return: 1 if successful, else 0

Syntax 6

ret = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *EventMask* [,*Context*]);

Argument	Meaning
EventMask	A mask of events. See the topic Event masks for more information on how masks are constructed. Type DOUBLE.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
18	EVENTMASK	Change the mask of events. Return: 1 if successful, else 0.

Syntax 7

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*[,*StartTime* [,*EndTime*]] [,*Context*] [,*ChangeNoOfLines*]);

Argument	Meaning
StartTime	The new start time for the Log Viewer, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE. If omitted then the current time is used.
EndTime	The new end time for the Log Viewer, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE. If omitted then the current time is used.
ChangeNoOfLines	An optional flag that causes a temporary change in the size of the Log Viewer buffer. See below.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
12	SETDATETIME	Selects a new time for the Log Viewer Return: 1 if successful, else 0.

Changing the number of lines

If the number of lines arising from the request is more than the configured size of the buffer, and the flag *ChangeNoOfLines*is set to 1 the Log Display buffer sizes is temporarily increased to the maximum of 32,000 lines. The display is positioned on first line of the start time.

If you make a further request, other than SETDATETIME with *ChangeNoOfLines* set to 1, or if you close the Log Viewer animation, the size of the Log Viewer buffer reverts to its original value.

Syntax 8

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity[, PrintFormat]*);

Argument	Meaning
PrintFormat	Optional parameter for selecting the format for printing: 0: Print the events(s) using the display format defined in the Log Viewer's Display tab. (default). 1: Print the events(s) using the print format defined in the Log Viewer's Execution tab.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
11	PRINTALL	Print the entire contents of the current buffer on the printer configured in the Log Viewer.
14	PRINTSELECTED	Print the selected line on the printer configured in the Log Viewer
15	PRINTDISPLAY	Print all the lines currently visible in the Log Viewer. Return: 1 if OK, else 0 (window does not exist etc.)

Syntax 9

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *ProgramModule, ProgramBranch, ProgramFunction, ProgramParameter*);

Argument	Meaning
ProgramModule	Module of the program to be executed.
ProgramBranch	Branch for the program.
ProgramFunction	Function of the program to be executed.
ProgramParameter	Parameter for the program.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
23	LINESELECT	Specifies a program to be executed when a line select/unselect occurs. Return: 1 if successful, else 0 for a bad parameter.

The XMLPATH instruction is used to get information about the LINESELECT mode. A namespace is created automatically using Branch, Mimic and Identity parameters. See the topic on XMLPATH for its format.

The meanings of the XML paths are as follows.

Path	Meaning
lineselect/variable	Variable name.
lineselect/x	X position
lineselect/y	Y position
lineselect/selected	Selected/unselected
lineselect/time	Date-time
lineselect.element[1-8].value	Column element

Syntax 10

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Column*, *Sort*);

Argument	Meaning
Column	Values 0 to 7 to identify the column, -1 to reset the sort.
Sort	Values 1 for ascending, 0 for descending.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
24	SETSORT	Dynamically sort a log list by a column. Return: 1 if successful, else 0 for a bad parameter

Syntax 11

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Column*);

Argument	Meaning
Column	Values 0 to 7 to identify the column, -1 to reset the sort.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
25	GETSORT	Obtain the status of the sort. Return: 1 if successful, else 0 for a bad parameter

The XMLPATH instruction is used to get information about the GETSORT mode. A namespace is created automatically using Branch, Mimic and Identity parameters. See the topic on XMLPATH for its format.

The meanings of the XML paths are as follows.

Path	Meaning
getsort/column	Values 0 to 7 to identify the column, -1 to reset the sort.
getsort/sort	Values 1 for ascending, 0 for descending.

Syntax 12

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*,*lLowPart*, *lHighPart*, [,*Context*]);

Argument	Meaning
lLowPart	Lower part of the mask. See the topic Event masks for more information on how masks are constructed. Type LONG.
lHighPart	Upper part of the mask. See the topic Event masks for more information on how masks are constructed. Type LONG.
Context	An optional flag. If set to 1 the action required by the instruction is delayed until the next EXECUTE mode instruction is used.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
22	EVENTMASKEX	Change the events filter of the log viewer. Return: 1 if successful, else 0 for a bad parameter.

Syntax 13

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity*, *Line*);

Argument	Meaning
Line	A 1 based line number. Type INTEGER.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
29	ISLINESELECTED	Checks if the line is selected. Return: 1 if selected, else 0
30	ISLINEVISIBLE	Checks if the line is visible. Return: 1 if visible, else 0

Syntax 14

IntVal = LOGDISPLAY(*Mode*, *Window*, *Branch*, *Identity, Separator*);

Argument	Meaning
Separator	The character or characters used to delimit the contents of each column. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
32	COPY_CLIPBOARD	Copy the selected lines to the Windows clipboard. Return: 1 if successful, else 0

Syntax 15

StrVal = LOGDISPLAY(*Mode, Window, Branch, Identity, LineStart[, LineEnd, ColSeparator, LineSeparator]*);

Argument	Meaning
Linestart	The 1 based numeric ID of a line in the Log Viewer buffer. Type INTEGER.
LineEnd	The 1 based numeric ID of a line in the Log Viewer buffer. Type INTEGER.
ColSeparator	The character or characters used to separate the columns in the returned string. The default is the comma ",". Type STR.
LineSeparator	The character or characters used to separate the lines in the returned string. The default is new line "\n". Type STR.

The return type is STR.

Execution

Mode	Mnemonic	Action
33	GETLINES	Return the lines from LineStart to LineEnd (inclusive) as a single string delimited using the ColSeperator and LineSeperator characters. If LineEnd is omitted only a single line is returned with the columns delimited using the default characters. Return: Selected lines of the Alarm Viewer.
34	GETSELECTEDLINES	Same as GETLINES except that only selected lines are returned.

Syntax 16

StrVal = LOGDISPLAY(*Mode, Window, Branch, Identity, Line, Column*);

Argument	Meaning
Line	The 1 based numeric ID of a line in the Log Viewer buffer. Type INTEGER.
Column	The 1 based numeric ID of a column in the Log Viewer buffer. Type INTEGER.

The return type is STR.

Execution

Mode	Mnemonic	Action
35	GETCELL	Return the contents of the specified cell.

Syntax 17

StrVal = LOGDISPLAY(*Mode, Window, Branch, Identity, AutoRefreshPeriod, PauseWhenNavigating*);

Argument	Meaning
AutoRefreshPeriod	The auto refresh period in seconds. If executed in a WebVue session context , the value c be less than 5 seconds. Type INTEGER.
PauseWhenNavigating	A flag indicating if the Log Viewer refresh should be paused whilst using the navigation bu 0 = Do not pause, 1 = pause. Type INTEGER.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
26	AUTOREFRESH_SET	Set the auto-refresh period and action whilst navigating. Return: 1 if successful, else 0.

Example

For examples, select the Example link above.

LOGICAL

See Also Example

Logical bitwise operation on 32-bit integers.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	NOT	1
1	AND	2
2	XOR	2
3	OR	2
4	SHIFT_LEFT	2
5	SHIFT_RIGHT	2
6	MODULO	2

Syntax 1

LongVal = LOGICAL(*Mode*, *x*);

The argument *x*can be of any numerical format.

Argument	Format
x	Any numerical format. It is converted to type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
0	NOT	Logical NOT.

Syntax 2

LongVal = LOGICAL(*Mode*, *x*, *y*);

The arguments *x*and *y* can be of any numerical format.

Argument	Format
x	Any numerical format. It is converted to type LONG.
y	Any numerical format. It is converted to type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
1	AND	Logical AND.
2	XOR	Logical XOR.
3	OR	Logical OR.
4	SHIFT LEFT	Bit shift to the left.
5	SHIFT RIGHT	Bit shift to the right.
6	MODULO	Modulus.

Example

For an example, select the Example link above.

LOGICAL64

See Also Example

Logical bitwise operation on 64-bit integers.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	NOT	1
1	AND	2
2	XOR	2
3	OR	2
4	SHIFT_LEFT	2
5	SHIFT_RIGHT	2
6	MODULO	2

Syntax 1

LongLongVal = LOGICAL64(*Mode*, *x*);

The argument *x*can be of any numerical format.

Argument	Format
x	Any numerical format. It is converted to type LONGLONG.

The return type is LONGLONG.

Execution

Mode	Mnemonic	Action
0	NOT	Logical NOT.

Syntax 2

LongLongVal = LOGICAL64(*Mode*, *x*, *y*);

The arguments *x*and *y* can be of any numerical format.

Argument	Format
x	Any numerical format. It is converted to type LONGLONG.
y	Any numerical format. It is converted to type LONGLONG.

The return type is LONGLONG.

Execution

Mode	Mnemonic	Action
1	AND	Logical AND.
2	XOR	Logical XOR.
3	OR	Logical OR.
4	SHIFT LEFT	Bit shift to the left.
5	SHIFT RIGHT	Bit shift to the right.
6	MODULO	Modulus.

Example

```
Sub main ()
'64-bit variable
Dim ll1 as longlong;
Dim ll2 as longlong;
Dim llRes as longlong;
ll1=2147491969;
Print(ll1); '2147491969
ll2=1;
llRes = logical64("AND", ll1, ll2);
Print(llRes); '1
```

```
End Sub
```

LONWORKS

See Also

Management and control of a LonWorks network configured in the Supervisor.

WebVue support - Yes.

Mode	Mnemonic	Syntax
2	READCONFIGPROPERTY	1
3	STARTNETWORK	2
4	STOPNETWORK	2
5	STARTNODE	3
6	STOPNODE	3
9	SENDMESSAGE	6

Syntax 1

IntVal = LONWORKS(*Mode, NetworkAlias, NodeAlias*);

The return type is INTEGER.

Argument	Meaning
NetworkAlias	The alias of a LonWorks network configured in the Supervisor. Type STR
NodeAlias	The alias of a LonWorks node configured in the Supervisor. Type STR

Execution

Mode	Mnemonic	Action
2	READCONFIGPROPERTY	Refresh the collection of variables in the Supervisor related to the ConfigProperties of the specified LonWorks network and node. Return: 1 if OK, else 0

Syntax 2

IntVal = LONWORKS(*Mode, NetworkAlias*);

The return type is INTEGER.

Argument	Meaning
NetworkAlias	The alias of a LonWorks network configured in the Supervisor. Type STR

Execution

Mode	Mnemonic	Action
3	STARTNETWORK	Start communication with the specified LonWorks network. The status of the network is available in the variable SYSTEM.LNS.*NetworkAlias*.ON.
4	STOPNETWORK	Stop communication with the specified LonWorks network. The status of the network is available in the variable SYSTEM.LNS.*NetworkAlias*.ON. Return: 1 if OK, else 0

Syntax 3

IntVal = LONWORKS(*Mode, szURL*);

The return type is INTEGER.

Argument	Meaning

szURL	The character string formed by *NetworkAlias/NodeAlias* or *NetworkAlias*.
NetworkAlias	The network alias as specified in the menu Configuration.Communication.LonWorks.
NodeAlias	The node alias as specified in the menu Configuration.Communication.LonWorks.

The application can detect when network start-up has completed by checking the variable:
SYSTEM.LNS.*NetworkAlias.NodeAlias*.ON

Execution

Mode	Mnemonic	Action
5	STARTNODE	Start communication with the specified LonWorks node. The status of the network is available in the variable SYSTEM.LNS.*NetworkAlias*.ON.
6	STOPNODE	Stop communication with the specified LonWorks node. The status of the network is available in the variable SYSTEM.LNS.*NetworkAlias*.ON. Return: 1 if OK, else 0

Syntax 6

IntVal = LONWORKS(*Mode*, *NetworkAlias/NodeAlias*, *MsgCode*, *MsgData* [, *ResultVar*]);

The return type is INTEGER.

Argument	Meaning
NetworkAlias	Identifier of the LonWorks network as defined in the menu Configuration/Communication/LonWorks.
NodeAlias	Identifier of the LonWorks network as defined in the menu Configuration/Communication/LonWorks.
MsgCode	A code for the message to be sent. It is a character string in hexadecimal format, with a value between 00 and FF inclusive.
MsgData	Data to be sent in the message. It is a character string in hexadecimal format, in which each pair of characters corresponds to one byte (values 00 to FF). Maximum length 1024 characters = 500 bytes.
ResultVar	Result variable (optional): 0: Message being sent. 1: Message sent successfully. 2: Sending failed.

Execution

Mode	Mnemonic	Action
9	SENDMESSAGE	This mode enables a message to be sent to the node specified by *NetworkAlias/NodeAlias*. Return: 0 Message sent. -1 Wrong syntax of *NetworkAlias/NodeAlias*. -2 Wrong syntax of *MsgCode.* -3 Wrong syntax of *MsgData.* -4 *ResultVar* does not exist or is not a register variable.

LPRINT

See Also

Send a message to a printer.

If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end which will be able to print on a configured printer connected to it.

WebVue support - Yes.

Syntax

IntVal = LPRINT(*Num*, *Mess1*, *Mess2*.....*Mess9*);

The return type is INTEGER.

Argument	Meaning
Num	The number of the printer from the configuration menu. Any numeric type.
Mess1 - 9	The messages to be printed. (Maximum 9) All variable types are supported.

Execution

Return: 1 if successful, else 0.

When printing the value of a database register variable, the output format is defined by the format property in the variable configuration.

Example

```
SUB main()
'Declare variables
DIM intReturn as Integer;
DIM sngRegister as Single;

sngMesure=12;
intReturn = LPRINT (1,"Example of using LPRINT: value = ",sngRegister ,34);
'Print this text on printer 1: "Example of using LPRINT: value = 12"
END SUB
```

LTRIM

See Also

Return a copy of a string without any leading spaces.

WebVue support - Yes.

Syntax

StrVal = LTRIM(*string*);

The return type is STR.

Argument	Meaning
string	The string whose leading spaces are to be removed. Type STR.

Example

```
SUB Main()
'Declare strings
DIM strResult as Str;
DIM strValue as Str;

strValue = "            Hello, World!"; 'preceding spaces
strResult = LTrim(strValue);
PRINT("LTrim(",strValue,") = ",strResult);
'Display "LTrim(            Hello, World!) = Hello, World!"
END SUB
```

LVAL

See Also

Convert a string to a LONG.

WebVue support - Yes.

Syntax

LongVal = LVAL(*string*);

The return type is LONG.

Argument	Meaning
string	The string to be converted. Type STR.

Execution

If any non-numeric characters exist in the string, the numeric characters before them are converted. If the string starts with a non-numeric character, the return is 0.

Example

```
SUB Main()
DIM lngResult as long;
DIM strString1 as Str;

strString1 = "125.35TEST";
lngResult = LVAL( strString1 );
PRINT("Result: ", lngResult );
'Display "Result: 125"
END SUB
```

M

M104

See Also

Starts, stops or resets communication via the IEC 60870-5-104 protocol.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	NETWORK_START	1
1	NETWORK_STOP	1
2	DEVICE_START	2
3	DEVICE_STOP	2
4	SECTOR_START	3
5	SECTOR_STOP	3
6	SECTOR_CLOCKSYNCHRO	3
7	SECTOR_RESET	3
8	SECTOR_GI	3
9	SECTOR_CI	3
10	STPV_COMMAND	4
11	SBO_SELECT	7
12	SBO_EXECUTE	8
13	SBO_CANCEL	8
16	DEVICE_SET_IPADD	6
17	DEVICE_SWITCHOVER	5

The instruction sends a message.

Syntax 1

M104(*Mode*, *NetworkID*);

Argument	Meaning
NetworkID	The network's identifier. Type STR.

Execution

Mode	Mnemonic	Action
0	NETWORK_START	To start the network.
1	NETWORK_STOP	To stop the network.

Syntax 2

Return= M104(*Mode*, *NetworkID*, *DeviceID*);

Argument	Meaning
NetworkID	The network's identifier. Type STR.
DeviceID	The device's identifier. Type STR.

Execution

Mode	Mnemonic	Action
2	DEVICE_START	To start the device.
3	DEVICE_STOP	To stop the device.

Syntax 3

Return= M104(*Mode*, *NetworkID*, *DeviceID*, *SectorID*);

Argument	Meaning

NetworkID	The network's identifier. Type STR.
DeviceID	The device's identifier. Type STR.
SectorID	The sector's identifier. Type STR.

Execution

Mode	Mnemonic	Action
4	SECTOR_START	To start the sector.
5	SECTOR_STOP	To stop the sector.
6	SECTOR_CLOCKSYNCHRO	To synchronize the PCs' clocks.
7	SECTOR_RESET	To reset.
8	SECTOR_GI	A global interrogation (GI) command.
9	SECTOR_CI	A counter interrogation (CI) command.

The command SECTOR_CLOCKSYNCHRO sends a message to set the time of a sector to the PC's time when the message was composed.

The command SECTOR_GI sends a request for the values of non-cyclic variables.

Syntax 4

Return= M104(*Mode*, *VariableName*, *Direction*);

Argument	Meaning
VariableName	The network's identifier. Type STR.
Direction	Direction of movement: "UP" or "DOWN". Type STR.

Execution

Mode	Mnemonic	Action
10	STPV_COMMAND	To step the variable's value up or down.

Syntax 5

Return= M104(*Mode*, *NetworkID*, *DeviceID, Device_to_activate_ID*;

Argument	Meaning
NetworkID	The network's identifier. Type STR.
DeviceID	The main device's identifier. Type STR.
Device _to_activate_ID	The identifier of the device to be activated. Type STR.

Execution

Mode	Mnemonic	Action
17	DEVICE_SWITCHOVER	Force activation of a particular device in a configuration where there are one or more standby devices configured.

Syntax 6

Return= M104(*Mode*, *NetworkID*, *DeviceID, IP_Address*;

Argument	Meaning
NetworkID	The network's identifier. Type STR.
DeviceID	The main device's identifier. Type STR.
IP_Address	New IP address for the device in the format x.x.x.x. Example 192.168.0.99. Type STR.

Execution

Mode	Mnemonic	Action
16	DEVICE_SET_IPADD	Changes the IP address of the device. The instruction also stops and starts the device which is required to complete the change of address.

Syntax 7

Return= M104(*Mode*, *VariableName, Value [*, *ResultVar])*;

The return type is integer.

Argument	Meaning
VariableName	The name of a variable that is mapped to an M104 information object. Type STR.
Value	The value to be sent to the M104 information object. Type INTEGER.
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. Type STR. 0 - Initiate request 1 - Success 2 - Command failed (see Event Viewer).

Execution

Mode	Mnemonic	Action
11	SBO_SELECT	Send a Select command to an M104 information object as part of a select before operate sequence. Return: 0 if successful. Any other value indicates an error in the instruction syntax. see below

Syntax 8

Return= M104(*Mode*, *VariableName [*, *ResultVar])*;

The return type is integer.

Argument	Meaning
VariableName	The name of a variable that is mapped to an M104 information object. Type STR.
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. Type STR. 0 - Initiate request 1 - Success 2 - Command failed (see Event Viewer).

Execution

Mode	Mnemonic	Action
12	SBO_EXECUTE	Send an Execute command to an M104 information object. Completes a select before operate sequence after a Select command has been previously sent. Return: 0 if successful. Any other value indicates an error in the instruction syntax.
13	SBO_CANCEL	Send a Cancel command to an M104 information object. Cancels a select before operate sequence after a Select command has been sent. Return: 0 if successful. Any other value indicates an error in the instruction syntax. see below

Return values from the Select, Execute and Cancel modes in case of error

-2 = Bad parameter.
-5 = Bad command sequence. For example Select command already sent.

Examples

```
M104("NETWORK_START", "NETWORK1");
M104("DEVICE START", "NETWORK1","DEVICE1");
M104("SECTOR_START", "NETWORK1","DEVICE1","SECTOR1");
M104("SECTOR_CLOCKSYNCHRO", "NETWORK1","DEVICE1","SECTOR1");
M104("SECTOR_RESET", "NETWORK1","DEVICE1","SECTOR1");
M104("SECTOR_GI", "NETWORK1","DEVICE1","SECTOR1");
M104("SECTOR_CI", "NETWORK1","DEVICE1","SECTOR1");
```

M61850

See Also

Sends commands via the IEC 61850 protocol.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	SBOW_SELECT	1
2	SBOW_OPERATE	2
3	SBOW_CANCEL	2
4	SBO_SELECT	1
5	SBO_OPERATE	2
6	SBO_CANCEL	2
7	START_NETWORK	3
8	STOP_NETWORK	3
9	START_PHYSICALDEVICE	4
10	STOP_PHYSICALDEVICE	4
11	START_REPORTGROUP	5
12	STOP_REPORTGROUP	5
13	START_DATASETGROUP	5
14	STOP_DATASETGROUP	5
15	START_DATAGROUP	5
16	STOP_DATAGROUP	5
17	DE_OPERATE	1
18	DEW_OPERATE	1

Syntax 1

Return= M61850(*Mode*, *VariableName, Value [*, *ResultVar])*;

The return type is integer.

Argument	Meaning
VariableName	The name of a variable that is mapped to an M61850 data object or attribute. Type STR.
Value	The value to be sent to the M61850 data object or attribute. Type INTEGER.
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. Type STR. 0 - Initiate request 1 - Success 2 - Command failed (see Event Viewer).

Execution

Mode	Mnemonic	Action
1	SBOW_SELECT	Send a Select command with enhanced security to an M61850 data object or attribute as part of a select before operate sequence.
4	SBO_SELECT	Send a Select command to an M61850 data object or attribute as part of a select before operate sequence.
17	DE_OPERATE	Generate a Direct command with normal security.
18	DEW_OPERATE	Generate a Direct command with enhanced security. Return: 0 if successful. A negative value is a parameter error, the value indicating which parameter is bad. For example -3 means that parameter 3 is bad.

Syntax 2

Return= M61850(*Mode*, *VariableName [*, *ResultVar])*;

The return type is integer.

Argument	Meaning

VariableName	The name of a variable that is mapped to an M61850 data object or attribute. Type STR.
ResultVar	The name of a register variable in which a value, corresponding to the result of the instruction, is placed. Type STR. 0 - Initiate request 1 - Success 2 - Command failed (see Event Viewer).

Execution

Mode	Mnemonic	Action
2	SBOW_OPERATE	Send an Operate command, with enhanced security, to an M61850 data object or attribute. Completes a select before operate sequence after a Select command has been previously sent.
3	SBOW_CANCEL	Send a Cancel command, with enhanced security, to an M61850 data object or attribute. Cancels a select before operate sequence after a Select command has been previously sent.
5	SBO_OPERATE	Send an Operate command to an M61850 data object or attribute. Completes a select before operate sequence after a Select command has been previously sent.
6	SBO_CANCEL	Send a Cancel command to an M61850 data object or attribute. Cancels a select before operate sequence after a Select command has been previously sent. Return: 0 if successful. A negative value is a parameter error, the value indicating which parameter is bad. For example -2 means that parameter 2 is bad.

Syntax 3

IntVar = M61850(Mode, NetworkID, ResultVar);

The return type is integer.

Argument	Meaning
NetworkID	The name of the network. Type STR.
ResultVar	The name of the variable in which is put the status of the command. Type STR. 0 - Initiate request 1 - Success 2 - Write error (see Event Viewer).

Execution

Mode	Mnemonic	Action
7	START_NETWORK	Start the network. The ResultVar is updated with the status of the request.
8	STOP_NETWORK	Stop the network. The ResultVar is updated with the status of the request. Return: 0 if successful, else not 0.

Syntax 4

IntVar = M61850(Mode, NetworkID, DeviceID, ResultVar);

The return type is integer.

Argument	Meaning
NetworkID	The name of the network. Type STR.
DeviceID	The name of the device. Type STR.
ResultVar	The name of the variable in which is put the status of the command. Type STR. 0 - Initiate request 1 - Success

2 - Write error (see Event Viewer).

Execution

Mode	Mnemonic	Action
9	START_PHYSICALDEVICE	Start the physical device. The ResultVar is updated with the status of the request.
10	STOP_PHYSICALDEVICE	Stop the physical device. The ResultVar is updated with the status of the request. Return: 0 if successful, else not 0.

Syntax 5

IntVar = M61850(Mode, NetworkID, DeviceID, GroupID, ResultVar);

The return type is integer.

Argument	Meaning
NetworkID	The name of the network. Type STR.
DeviceID	The name of the device. Type STR.
GroupID	The name of the group. Type STR.
ResultVar	The name of the variable in which is put the status of the command. Type STR. 0 - Initiate request 1 - Success 2 - Write error (see Event Viewer).

Execution

Mode	Mnemonic	Action
11	START_REPORTGROUP	Start the report group.
12	STOP_REPORTGROUP	Stop the report group.
13	START_DATASETGROUP	Start the dataset group.
14	STOP_DATASETGROUP	Stop the dataset group.
15	START_DATAGROUP	Start the data group.
16	STOP_DATAGROUP	Stop the data group. Return: 0 if successful, else not 0.

MAPDISPLAY

See Also Example

Manage the operation of a Map Control in the HMI.

WebVue support - No.

Mode	Mnemonic	Syntax
1	LOADDEFAULT	1
2	CENTERTOMARKER	2
3	GETZOOM	3
4	SETZOOM	4
5	GETLATITUDE	3
6	SETLATITUDE	4
7	GETLONGITUDE	3
8	SETLONGITUDE	4
9	EXPORT	5
10	LOADMARKERS	6
11	ISLAYERDISPLAYED	7
12	SETLAYERVISIBILITY	8
13	GETMARKERLATITUDE	9
14	SETMARKERLATITUDE	10
15	GETMARKERLONGITUDE	9
16	SETMARKERLONGITUDE	10
17	GETMARKERTOOLTIP	11
18	SETMARKERTOOLTIP	10
19	GETMARKERMIMIC	11
20	SETMARKERMIMIC	12
21	GETMARKERBRANCH	11
22	SETMARKERBRANCH	12
23	GETMARKERTYPE	13
24	GETUSERDATAVARIABLENAME	14
25	GETUSERDATALABEL	14
26	SETUSERDATALABEL	15
27	GETUSERDATAVALUE	14
28	SETUSERDATAVALUE	15
29	ISLOADING	16
30	GETMARKERVISIBILITY	17
31	SETMARKERVISIBILITY	10
32	SETMAPPROVIDER	18

Modes associated with map markers (2, and 11 to 28 inclusive) will generate an error if executed while the Map Markers file is still loading.

Properties Common to all Modes

Argument	Meaning
Window	The name of the window containing the Map Control. Type STR.
Branch	The branch of the window containing the Map Control. Type STR.
MapID	The identifier of the Map Control (within the window). Type STR.
LayerName	The identity of a map markers layer. Type STR.
MarkerName	The identity of a map marker. Type STR.
ViewIndex	The view index (starting at 1) if there is more than one view of the window (mimic) open in the same region. Optional.
Value	New value for the map property being set. Type depends on the context - either INTEGER, DOUBLE or STR.

Syntax 1

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*);

The return type is integer.

Execution

Mode	Mnemonic	Action
1	LOADDEFAULT	Resets the run-time properties of the Map Control to their default values, including reloading the GPX (map markers) file. Return: 1 if successful, else 0.

Syntax 2

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName*, *MarkerName*[, *ViewIndex*]);

The return type is integer.

Execution

Mode	Mnemonic	Action
2	CENTERTOMARKER	Centres a map on the given marker. If the marker is multipoint (track or polygon), then the map relocates on the first point of the object. If *LayerName* is empty, the default layer is used. Return: 1 if successful, else 0.

Syntax 3

DblVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*[, *ViewIndex*]);

The return type is double.

Execution

Mode	Mnemonic	Action
3	GETZOOM	Gets the zoom level of the map. Return: The zoom level.
5	GETLATITUDE	Gets the latitude of the centre of the map using the WGS84 system. Return: The latitude.
7	GETLONGITUDE	Gets the longitude of the centre of the map using the WGS84 system. Return: The longitude.

Syntax 4

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *Value*[, *ViewIndex*]);

The return type is integer.

Execution

Mode	Mnemonic	Action
4	SETZOOM	Set a new zoom level for the map. Return: 1 if successful, else 0.
6	SETLATITUDE	Set the latitude of the centre of the map using the WGS84 system. Return: 1 if successful, else 0.
8	SETLONGITUDE	Set the longitude of the centre of the map using the WGS84 system. Return: 1 if successful, else 0.

Syntax 5

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *FileName*);

The return type is integer.

Argument	Meaning
FileName	The name of the file to be used for output. Type STR.

Execution

Mode	Mnemonic	Action
6	EXPORT	Creates a map markers file from the layers and markers of a Map Control. Generates an error if the map markers file is still loading. Return: 1 if successful, else 0.

Syntax 6

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *FileName*[, *AsyncFlag*[, *StateVar*]])

The return type is integer.

Argument	Meaning
FileName	The name of the file containing the map markers. Type STR.
AsynchFlag	A flag indicating if the file should be loaded synchronously (0) or asynchronously (1).
StateVar	The name of a register variable that indicates the status of the load process. Type STR. 0 = Success 1 = Running 3 = Cancelled 10 = Failed

Execution

Mode	Mnemonic	Action
10	LOADMARKERS	Loads layers and markers from the specified file. Previously loaded markers are deleted. Return: 1 if successful, else 0.

Syntax 7

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName*[, *ViewIndex*]);

The return type is integer.

Execution

Mode	Mnemonic	Action
11	ISLAYERDISPLAYED	Returns the visibility of a layer of the given Map Control, taking in account Show property and zoom limits. Return: 1 if visible, 0 if hidden, else -1.

Syntax 8

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, Value*);

The return type is integer.

Argument	Meaning
Value	A flag to indicate the visibility setting. Type INTEGER. 1 to show, 0 to hide.

Execution

Mode	Mnemonic	Action
12	SETLAYERVISIBILITY	Set layer visibility. Return: 1 if successful, else 0.

Syntax 9

DblVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName*);

The return type is double.

Execution

Mode	Mnemonic	Action
13	GETMARKERLATITUDE	Gets the latitude coordinate (-90 to 90) of the marker in WGS84 system. Does not work for multi-point markers (tracks or polygons). Return: The latitude coordinate.
15	GETMARKERLONGITUDE	Gets the longitude coordinate (-180 to 180) of the marker in WGS84 system. Does not work for multi-point markers (tracks or polygons). Return: The longitude coordinate.

Syntax 10

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName*, *Value*);

The return type is integer.

Execution

Mode	Mnemonic	Action
14	SETMARKERLATITUDE	Sets the latitude coordinate (-90 to 90) of the marker in WGS84 system using the supplied value. Does not work for multi-point markers (tracks or polygons). The Value parameter is a double and is the latitude value. Return: 1 if successful, else 0.
16	SETMARKERLONGITUDE	Sets the longitude coordinate (-180 to 180) of the marker in WGS84 system using the supplied value. Does not work for multi-point markers (tracks or polygons). The Value parameter is a double and is the longitude value. Return: 1 if successful, else 0.
18	SETMARKERTOOLTIP	Sets the marker short description (tooltip). The Value parameter is a string and is the tooltip text. Return: 1 if successful, else 0.
31	SETMARKERVISIBILITY	Sets the marker's visibility. Value = 1 is visible, Value = 0 is invisible. Return: 1 if successful, else 0.

Syntax 11

StrVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName*);

The return type is string.

Execution

Mode	Mnemonic	Action
17	GETMARKERTOOLTIP	Gets the short description (tooltip) of the marker in current language. Return: The marker tooltip.
19	GETMARKERMIMIC	Gets the name of the mimic that should be opened when clicking on the marker. Return: The mimic name. An empty string indicates a failure.
21	GETMARKERBRANCH	Gets the branch of the marker. This branch is used to bind marker properties. If the mimic property is defined, when a user clicks on the marker, the mimic will be opened with the corresponding branch. For symbol markers the branch is also assigned to the instantiated symbols. Return: The marker branch.

Syntax 12

IntVal = MAPDISPLAY *(Mode, Window, Branch, MapID, LayerName, MarkerName, Value[, ChildMode]);*

The return type is integer.

Argument	Meaning
ChildMode	Flag indicating if the mimic is to be opened as a child or an independent window.

Execution

Mode	Mnemonic	Action
20	SETMARKERMIMIC	Sets the name of the mimic that should be opened when clicking on the marker. The parameter Value is a string and is the name of the mimic. Return: 1 if successful, else 0.
22	SETMARKERBRANCH	Sets the branch of the marker. This branch is used to bind marker properties. If the mimic property is defined, when a user clicks on the marker, the mimic will be opened with the corresponding branch. For symbol markers the branch is also assigned to the instantiated symbols. Return: 1 if successful, else 0.

Syntax 13

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName*);

The return type is integer.

Execution

Mode	Mnemonic	Action
23	GETMARKERTYPE	Gets the marker type. Return: 1 = Shape 2 = Text 3 = Image 4 = Symbol 5 = Track 6 = Polygon 0 = Unsuccessful.

Syntax 14

StrVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName, UserDataName*);

The return type is string.

Argument	Meaning
UserDataName	The name given to an item in the User Data array as configured using the Map Markers Editor. Type STR.

Execution

Mode	Mnemonic	Action
24	GETUSERDATAVARIABLENAME	Get the name of the variable, as configured for the supplied UserDataName, in the User Data array. Return: The variable name.
25	GETUSERDATALABEL	Get the label, as configured for the supplied UserDataName, in the User Data array. Return: The label.
27	GETUSERDATAVALUE	Get the value of the variable, as configured for the supplied UserDataName, in the User Data array.Return: The variable value as a string.

Syntax 15

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName, UserDataName, Value*);

The return type is integer.

Argument	Meaning
UserDataName	The name given to an item in the User Data array as configured using the Map Markers Editor. Type STR.

Execution

Mode	Mnemonic	Action
26	SETUSERDATALABEL	Set the label for the supplied UserDataName, in the User Data array. The Value parameter is a string and is the label text. Return: 1 if successful, else 0.
28	SETUSERDATAVALUE	Set the value of the variable for the supplied UserDataName, in the User Data array. The Value parameter is a string and is the data value.Return: 1 if successful, else 0.

Syntax 16

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*);

The return type is integer.

Execution

Mode	Mnemonic	Action
29	ISLOADING	Test if the map markers file is loading. Return: 1 = Loading, 0 = Loaded, -1 indicates an error.

Syntax 17

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *LayerName, MarkerName*);

The return type is integer.

Execution

Mode	Mnemonic	Action
30	GETMARKERVISIBILITY	Gets the marker's visibility. Return: 1 if visible, 0 if hidden, -1 indicates an error.

Syntax 18

IntVal = MAPDISPLAY (*Mode*, *Window*, *Branch*, *MapID*, *ProviderName*, *IsLocal* [, *UseCache*]);

Return type: INTEGER.

Argument	Signification
ProviderName	Map provider name. Type STR.
IsLocal	Local or online map. 1 if local map, 0 if online map.
UseCache	Flag indicating to use the cache. 1 use the cache, 0 do not use the cache.

Execution

Mode	Mnemonic	Action
32	SETMAPPROVIDER	Change the map provider. Return: 1 if successful, else 0.

MDNP3

See Also

Manages communication via the DNP3 protocol.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	NETWORK_START	1
1	NETWORK_STOP	1
2	DEVICE_START	2
3	DEVICE_STOP	2
4	CLOCK_SYNCHRO	3
5	RESET or COLD_RESTART	3
6	REFRESH_ATT	3
7	READ_CLASS	4
8	READ_GROUP	5, 6 or 7
9	FREEZE	8
10	SELECT	9
11	OPERATE	9
12	DIRECT_OPERATE	10
13	SELECT_CROB	11
14	OPERATE_CROB	11
15	DIRECT_OPERATE_CROB	11
16	DIRECT_OPERATE_NO_ACK_CROB	11

About the status variable *ResultVar* used in several of the modes

None of the MDNP3 instruction modes return a value. Instead they use a result variable, the name of which is an optional argument in several of the modes. *ResultVar* must be a register variable and has only three possible values.

0 - The command is being sent to the DNP3 network.
1 - The command was sent successfully.
2 - The command was not sent successfully.

Syntax 1

MDNP3(*Mode*, *NetworkID*);

Argument	Meaning
NetworkID	The network's identifier. Type STR.

Execution

Mode	Mnemonic	Action
0	NETWORK_START	Start the network.
1	NETWORK_STOP	Stop the network.

Syntax 2

MDNP3(*Mode*, *NetworkID*, *DeviceID*);

Argument	Meaning
NetworkID	The network's identifier. Type STR.
DeviceID	The device's identifier. Type STR.

Execution

Mode	Mnemonic	Action
2	DEVICE_START	Start the device.

3	DEVICE_STOP	Stop the device.

Syntax 3

MDNP3(*Mode*, *NetworkID*, *DeviceID[, ResultVar]*);

Argument	Meaning
NetworkID	The networks identifier. Type STR.
DeviceID	The device's identifier. Type STR.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
4	CLOCK_SYNCHRO	Synchronise the device clock.
5*	RESET	Instruct the device to perform a cold restart (function code 13).
5*	COLD_RESTART	Instruct the device to perform a cold restart (function code 13).
6	REFRESH_ATT	Refresh the device attributes and timestamp of the device in the Supervisor. Reads object groups 0 and 50.

* Mode 5 has two alternative mnemonics. The actions are identical.

Syntax 4

MDNP3(*Mode*, *NetworkID*, *DeviceID*, *ClassNum [, ResultVar]*);

Argument	Meaning
NetworkID	The network's identifier. Tye STR.
DeviceID	The device's identifier. Type STR.
ClassNum	The number of a DNP3 class. Range 0 to 3.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
7	READ_CLASS	Refresh the data in the Supervisor by requesting the data points associated with one of the DNP3 classes.

Syntax 5

MDNP3(*Mode*, *NetworkID*, *DeviceID*, *GroupNum*, *VariationNum*, "START_STOP", *Start*, *Stop[, ResultVar]*);

Argument	Meaning
NetworkID	The network's identifier. Tye STR.
DeviceID	The device's identifier. Type STR.
GroupNum	The DNP3 group number. Any numeric type.
VariationNum	The variation number within the group which determines the encoding format for the data and the presence (or not) of the flags and event time. Any numeric type.
Start	The number of the first data point to be read. Any numeric type.
Stop	The number of the last data point to be read. Any numeric type.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
8	READ_GROUP	Refresh the data in the Supervisor requesting a range of data points from a group and using a specific variation to format the data.

Syntax 6

MDNP3(*Mode*, *NetworkID*, *DeviceID*, *GroupNum*, *VariationNum*, "QUANTITY", *Quanitity[, ResultVar]*);

Argument	Meaning

NetworkID	The network's identifier. Tye STR.
DeviceID	The device's identifier. Type STR.
GroupNum	The DNP3 group number. Any numeric type.
VariationNum	The variation number within the group which determines the encoding format for the data and the presence (or not) of the flags and event time. Any numeric type.
Quantity	The number of data points to be read. Any numeric type.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
8	READ_GROUP	Refresh the data in the Supervisor requesting a number of data points from a group and using a specific variation to format the data.

Syntax 7

MDNP3(*Mode*, *NetworkID*, *DeviceID*, *GroupNum*, *VariationNum*, "ALL"*[, ResultVar]*);

Argument	Meaning
NetworkID	The network's identifier. Type STR.
DeviceID	The device's identifier. Type STR.
GroupNum	The DNP3 group number. Any numeric type.
VariationNum	The variation number within the group which determines the encoding format for the data and the presence (or not) of the flags and event time. Any numeric type.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
8	READ_GROUP	Refresh the data in the Supervisor requesting all the data points from a group and using a specific variation to format the data.

Syntax 8

MDNP3(*Mode*, *AckMode*, *ClearMode*, *NetworkID*, *DeviceID[, ResultVar]*);

Argument	Meaning
AckMode	A flag requesting acknowledgement or not. Either "NO_ACK" (0), or "ACK" (1).
ClearMode	A flag requesting that the counters be cleared, or not. Either "DO_NOT_CLEAR" (0), or "CLEAR" (1).
NetworkID	The network's identifier. Type STR.
DeviceID	The device's identifier. Type STR.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
9	FREEZE	Freeze the device counters with optional acknowledge and clear.

Syntax 9

MDNP3(*Mode*, *Variable*, *Value[, ResultVar]*);

Argument	Meaning
Variable	The name of a variable linked to a data point of a DNP3 device. The variable can be either a register linked to an AO (analog output) or a bit linked to a BO (binary output). Type STR.
Value	The value to which the variable is to be sent.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
10	SELECT	Send a select message to the DNP3 device. A select should be followed by an operate.
11	OPERATE	Send an operate message to the DNP3 device. An operate must be preceded by a select.

Syntax 10

MDNP3(*Mode*, *Variable*, *Value[, NoAck, [ResultVar]]*);

Argument	Meaning
Variable	The name of a variable linked to a data point of a DNP3 device. The variable can be either a register linked to an AO (analog output) or a bit linked to a BO (binary output). Type STR.
NoAck	An optional flag set to 1 (no acknowledge is required) or 0 (acknowledge is required). The default is 0.
Value	The value to which the variable is to be sent.
ResultVar	The name of a register variable. Type STR.

Execution

Mode	Mnemonic	Action
12	DIRECT_OPERATE	Send a direct operate message to the DNP3 device.

The messages sent to the DNP3 device using modes 10, 11 and 12 incorporate a structured datablock ca an AOB (Analog Output Block) when used for a register, or a CROB (Control Relay Output Block) when u with a bit. Some of the parameters used in the AOB and CROB are supplied by the variable's advanced properties.

Syntax 11

MDNP3(*Mode*, *Variable*, *OpType*, *Queue*, *Clear*, *TripClose*, *Count*, *OnTime*, *OffTime[, ResultVar]*);

Argument	Meaning
Variable	The name of a variable linked to a data point of a DNP3 device. The variable can be either a register linked to an AO (analog output) or a bit linked to a BO (binary output). Type STR.
OpType	See below.
Queue	See below.
Clear	See below.
TripClose	See below.
Count	The number of times that the operation is to be executed.
OnTime	The CROB off time in milliseconds. Any numeric type.
OffTime	The CROB on time in milliseconds. Any numeric type.
ResultVar	The name of a register variable. Type STR.

Coding of the OpType argument

Mnemonic	Value
"NULL"	0
"PULSE_ON"	1
"PULSE_OFF"	2
"LATCH_ON"	3
"LATCH_OFF"	4

Coding of the Queue argument

Mnemonic	Value
"DO_NOT_QUEUE"	0
"QUEUE"	1

Coding of the Clear argument

Mnemonic	Value
"DO_NOT_CLEAR"	0
"CLEAR"	1

Coding of the TripClose

Mnemonic	Value
"NULL"	0
"TRIP"	1
"CLOSE"	2

Execution

Mode	Mnemonic	Action
13	SELECT_CROB	Send a select message to the DNP3 device. The CROB properties are supplied as arguments rather than by the variable's advanced properties. A select should be followed by an operate.
14	OPERATE_CROB	Send an operate message to the DNP3 device. The CROB properties are supplied as arguments rather than by the variable's advanced properties. An operate must be preceded by a select.
15	DIRECT_OPERATE_CROB	Send a direct operate message to the DNP3 device. The CROB properties are supplied as arguments rather than by the variable's advanced properties.
16	DIRECT_OPERATE_NO_ACK_CROB	Send a direct operate with no acknowledge message to the DNP3 device. The CROB properties are supplied as arguments rather than by the variable's advanced properties.

If the property Different address was enabled in the variable's configuration this is still taken into accour

MID

See Also

Return a sub-string of a character string.

WebVue support - Yes.

Syntax

StrVal = MID(*Input*, *Start* [, *Num*]);

The return type is STR.

Argument	Meaning
Input	The string to be manipulated. Type STR.
Start	The start position for the returned sub-string (0 denotes the first character position). Any numeric type.
Num	The number of characters in the returned sub-string. Any numeric type.

Execution

Returns *Num* characters from *Input* from the position defined by *Start*. If *Num* is omitted, or the remainder of the string is less than *Num* then the remainder of the string is returned.

Example

```
SUB Main()
DIM strChain1 as Str;
DIM strChain2 as Str;
DIM inti1 as Integer;
DIM inti2 as Integer;

inti1 = 7;
inti2 = 5;
strChain1 = "Hello, World!";
strChain2 = MID (strChain1,inti1,inti2); '5 characters from the 7th
Print("Return: ",strChain2);
'Display "Return: World"
END SUB
```

MULTIMEDIA

See Also Example

Provides access to the Window's Media Control Interface (MCI) for the desktop client. To view the MCI Help file click here. (Saved as a zip file - it must be extracted first.)

WebVue support - Not applicable. Returns the unsuccessful code if used in this context.

Mode	Mnemonic	Syntax
0	SEND	1
1	OPENW	2
2	WINDOW	3

Syntax 1

StrVal = MULTIMEDIA(*Mode*, *MCI_String*);

The return type is STR.

Argument	Meaning
MCI_string	The string to be used. Type STR.

Execution

Mode	Mnemonic	Action
0	SEND	Send a MCI command using the string of characters contained in the paramater *MCI_String*.

Return: The status of the MCI command.

Syntax 2

IntVal = MULTIMEDIA(*Mode*, *device_id*, *res_name*[, *window* [, *wbranch* [, *x*, *y*, *w*, *h*]]]);

The return type is INTEGER.

Argument	Meaning
device_id	The alias of the resource.
res_name	The resource name.
window	The window for embedding the resource.
wbranch	The mimic branch for embedding the resource.
x, y	Co-ordinates of its top left corner
w, h	Width and height.

Execution

Mode	Mnemonic	Action
1	OPENW	Open the video resource *res_name* (file .AVI, Overlay video, etc.) with the alias *device_id* embedding it in the window specified by *window* and its branch *wbranch*.

The picture is positioned using the co-ordinates of its top left corner (*x* and *y*). The size is specified by its width and height (*w* and *h*). If no window is specified, a window is created outside the application.

Return: 1 if OK , else 0.

Syntax 3

IntVal = MULTIMEDIA(*Mode*, *device_id*, *window* , *wbranch*[, *x*, *y*, *w*, *h*]);

The return type is INTEGER.

Argument	Meaning
device_id	The alias of the resource.
window	The window for embedding the resource.
wbranch	The mimic branch for embedding the resource.
x, y	Co-ordinates of its top left corner
w, h	Width and height.

Execution

Mode	Mnemonic	Action
2	WINDOW	Embed the video resource specified with the alias *device_id* in the window specified by window and its branch *wbranch*.

The picture is positioned using the co-ordinates of its top left corner (*x* and *y*).

The size is specified by its width and height (*w* and *h*). If no window is specified, a window is created outside the application.

Return: 1 if OK, else 0 (for example if the window is not open).

Example

For a larger example, select the Example link above.

```
SUB Main()
WINDOW("OPEN", "toto", "sample", "");
IF (WINDOW("IS_OPEN","toto","sample")) THEN
 MULTIMEDIA ("OPENW","MOVIE","Clock.avi","toto", "sample",20,10,220,240);
END IF
MULTIMEDIA ("SEND","Play movie");
MULTIMEDIA ("SEND","Set movie time format frames");
END SUB
```

(No topics)

There are no topics in this book.

o

OCT

See Also

Returns a string representing, in octal, a number passed in base 10.

WebVue support - Yes.

Syntax

StrVal = OCT(*N*);

The return type is STR.

Argument	Meaning
N	The number to be converted. Any numeric type.

Execution

The return string can have a maximum of 16 characters.

Example

```
SUB Main()
Declare
DIM IntValue as integer;
DIM strResult as Str;
IntValue = 13;
strResult = OCT(IntValue);
'Euclidean division: 13 = 1*8 + 5
PRINT("OCT(",IntValue,") = ",strResult);
'Displays "OCT(13) = 15"
END SUB
```

OPC

See Also

Manage exchanges with a OPC server configured in the Supervisor.

WebVue support - Yes.

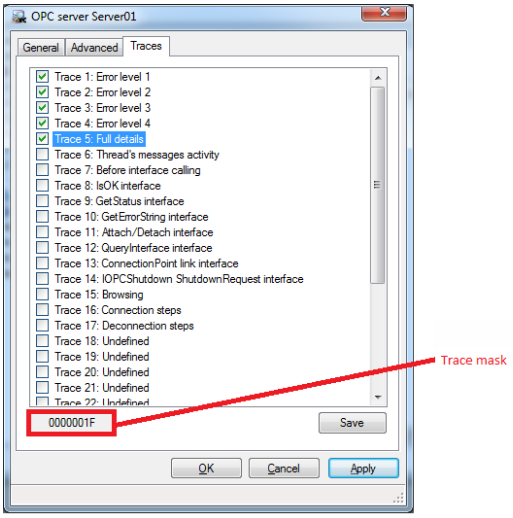
Mode	Mnemonic	Syntax
1	READ	1
5	SETTRACE	2
6	RESETTRACE	2
7	PUTMASKTRACE	2
8	SAVEMASKTRACE	2
9	STARTGROUP	2
10	STOPGROUP	2
11	STARTSERVER	3
12	STOPSERVER	3

The mode READ replaces the previous modes READ_SYNC and READ_ASYNC.

The mode ACTIVATE_ITEMS is no longer supported.

Modes 5 to 8 are for managing traces. With them, you can initiate all actions of the OPC Server dialog's Traces tab. The actions (check to set or uncheck to reset) modify the trace mask as displayed at the foot of the dialog.

Show picture



Syntax 1

OPC (*Mode*, *szURL* [,*ReadFrom*]);

There is no return type.

As configured in the OPC settings of the Supervisor:

Argument	Meaning
szURL	The character string formed by *ServerAlias/GroupName* or *ServerAlias*.
ServerAlias	The alias of the OPC server identified in the menu Configuration.Communication.OPC. Type STR.

GroupName	The OPC Group name as identified in the menu Configuration.Communication.OPC. Type STR.
ReadFrom	Whether to read from: "DEVICE" - the device's memory (default) or "CACHE" - from the OPC server's memory.

The parameters *ServerAlias* and *GroupName* are separated by a single oblique character (/).

Execution

Mode	Mnemonic	Action
1	READ	Read from the equipment for the specified OPC server and group, from beginning to end of the equipment managed by the server. The read parameters are provided by the configuration of the group (synchronous or asynchronous).

Syntax 2

OPC (*Mode*, *szURL*);

There is no return type.

Argument	Meaning
szURL	The character string formed by *ServerAlias/GroupName* or *ServerAlias*.
ServerAlias	The alias of the OPC server identified in the menu Configuration.Communication.OPC. Type STR.
GroupName	The OPC Group name as identified in the menu Configuration.Communication.OPC. Type STR.

Execution

Mode	Mnemonic	Action
5	SETTRACE	To set the trace mask on the OPC server.
6	RESETTRACE	To reset the trace mask on the OPC server.
7	PUTMASKTRACE	To set the full trace mask on the OPC server.
8	SAVEMASKTRACE	To save the full trace mask on the OPC server. At the next startup, the trace mask will be the saved mask.
9	STARTGROUP	To start the identified group.
10	STOPGROUP	To stop the identified group.

The application can detect when network start-up has completed by checking the variable:

SYSTEM.OPC.*ServerAlias.GroupName*.ON

Syntax 3

OPC (*Mode*, *ServerAlias*[/*GroupName*]);

There is no return type.

Argument	Meaning
ServerAlias	The alias of the OPC server identified in Configuration.Communication.OPC.

Execution

Mode	Mnemonic	Action
11	STARTSERVER	To start the server identified by *ServerAlias*.
12	STOPSERVER	To stop the server identified by *ServerAlias*.

For STARTSERVER and STOPSERVER, the application can detect when server starting up or stopping has completed by checking the variable:

```
SYSTEM.OPC.ServerAlias.ON
```

OPTIONLIST

See Also Example

Access to the Supervisor's Option-button list form control.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	WebVue
1	COUNT	1	Yes
2	GETSELECTEDINDEX	1	Yes
3	SETSELECTEDINDEX	2	Yes
4	GETTEXT	4	Yes
5	GETUSERDATA	4	Yes
6	GETSTATE	4	Yes
7	SETSTATE	3	Yes
8	LOAD	5	Yes
9	GETEVENTTYPE	1	Yes
10	GETOPTION	1	Yes
11	SETOPTION	2	Yes
12	INSERT	6	No
13	REMOVE	7	No
14	SORT	8	No
15	CLEAR	1	No

An important note about the terminology used and operation of the Supervisor's Option List Control.

- The item that is SET is the one in which the centre of the corresponding circle is filled.
- The item that is SELECTED is the one whose text is highlighted (by changing its background colour

An item that is SET is not necessarily SELECTED (and vice versa). Programmatically you are normally on interested in which item is SET. The item that is SELECTED has little meaning.
For more information on the behaviour of the Option List Control see the Supervisor's main help.

All syntaxes

Argument	Meaning
Window	The name of the window that contains the form control. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR.
Identity	The identity of the form control within the specified window. Type STR.

Syntax 1

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*);

Execution

Mode	Mnemonic	Action
1	COUNT	Returns the number of items. Type LONG.
2	GETSELECTEDINDEX	Returns the index of the item currently selected. Type LONG.
9	GETEVENTTYPE	Returns the type of the most recent event. Type LONG. The return is binary weighted value. Bit 0: The selection has been changed. Bit 1: An item's state has been changed.
10	GETOPTION	Return the index of the item which is set. If no item is set then -1 is returned. Typ LONG

15	CLEAR	Clears the list contents and any selection. Return: 1 if successful, else 0.

Syntax 2

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*, *Notification*);

Argument	Meaning
Index	The index of the item to select. Type LONG. If the index is -1 then no item is selected.
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control. Type INTEGER. For this to work, it also has to be enabled in the Operations tab of the Option List properties dialog. The function will scroll the control if necessary to display the item.

Execution

Mode	Mnemonic	Action
3	SETSELECTEDINDEX	Selects the item corresponding to the supplied index. Return: 1 if successful, else 0.
11	SETOPTION	Sets the item corresponding to the supplied index. Return: The index of the item set or -1 if no item is checked. Type INTEGER.

Syntax 3

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*, *State*, *Notification*);

Argument	Meaning
Index	The index of the item to select. Type LONG. If the index is -1 then no item is selected.
State	State to be set. The value can be 0 or 1. Type INTEGER
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control. Type INTEGER. For this to work, it also has to be enabled in the Operations tab of the Option List properties dialog. The function will scroll the control if necessary to display the item.

Execution

Mode	Mnemonic	Action
7	SETSTATE	Sets the state of the item corresponding to the supplied index. Type LONG. Return: 1 if successful, else 0.

Syntax 4

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of the item whose text is to be retrieved.

Execution

Mode	Mnemonic	Action
4	GETTEXT	Returns the Text of the item as defined for the language currently in use. Type ST

5	GETUSERDATA	Returns the User Data associated with the item. Type STR.
6	GETSTATE	Returns the state of an item (set or not). Type LONG.

Syntax 5

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*, *FileName*);

Argument	Meaning
FileName	The name of the file that contains the form control's data.

Execution

Mode	Mnemonic	Action
8	LOAD	Loads the file's content into the form control's list of items. Type INTEGER. Return: 1 if successful (file loaded), else 0.

Syntax 6

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*, *Text*, *Userdata*, [*Index*]);

Argument	Meaning
Text	The text to insert. Type STR.
Userdata	The user data to insert. Type STR.
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
12	INSERT	Inserts the Text and optional Userdata in the form control list. If Index is omitted text is inserted at the start. If Index is -1 the text is inserted at the end. Otherwis text is inserted at the position indicated by Index. Return: 1 if successful, else 0.

Syntax 7

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*, *Index*);

Argument	Meaning
Index	The index of an item in the form control. Type LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
13	REMOVE	Removes the selected item from the list. Return: 1 if successful, else 0.

Syntax 8

Return = OPTIONLIST(*Mode*, *Window*, *Branch*, *Identity*[, *Sortorder*, *Orderby*]);

Argument	Meaning
Sortorder	A flag indication the sort order. 0 = ascending, 1 = descending
Orderby	A flag selecting which data to sort by. 0 = Tex, 1 = Userdata.

The return type is LONG.

Execution

Mode	Mnemonic	Action
14	SORT	Sort the form control list using the specified criteria. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

P

PIE

See Also Example

Manage the operation of a Pie Control in the HMI.

WebVue support - No.

Mode	Mnemonic	Syntax
1	ADD_DATAPOINT	1
2	REMOVE_DATAPOINT	2
3	CLEAR	3
4	GET_COUNT	3
5	SET_COLOR	4
6	SET_VALUE	5
7	LOAD	6
8	COLLECTED_LABEL	7
9	COLLECTED_COLOR	8
10	COLLECTED_THRESH	9
11	RESET_COLLECTED	3
12	EXPLODED_SLICE	10
13	RESET_EXPLODED	3
14	PRINT	11
15	COPY_CLIPBOARD	3
16	SAVE_IMAGE	12
17	SET_CHARTTYPE	13
18	SET_3DDISPLAY	14
19	CUSTOM_PROPERTY	15
20	SET_TITLE	16
21	SHOW_LEGEND	17
22	SET_BACKIMAGE	18

Modes 8 to 13 inclusive are used, for a pie or doughnut chart, to manage the collection one or more segments into one. They cannot be used with a pyramid chart.

Properties common to more than one mode

Argument	Meaning
Window	The name of the window containing the pie. Type STR.
Branch	The branch of the window containing the pie. Type STR.
Identifier	The identifier of the pie (within the window). Type STR.
Tooltip	The text for a tooltip which appears when the mouse is over a segment. Type STR.
Value	The value for a segment. Type DOUBLE.
Label	The text for the label. Type STR
LegendText	The text for the legend. Type STR

Syntax 1

LongVal = PIE(*Mode, Window, Branch, Identity, Value, Label, LegendText [, Tooltip])*

The return type is LONG.

Execution

Mode	Mnemonic	Action
1	ADD_DATAPOINT	Add a data point to the end of the collection. Return: 1 if OK, 0 if NOK.

Syntax 2

LongVal = PIE(*Mode, Window, Branch, Identity, Index*)

The return type is LONG.

Execution

Mode	Mnemonic	Action
2	REMOVE_DATAPOINT	Remove a data point from the collection using the given index. Return: The number of data points after the remove.

Syntax 3

LongVal = PIE(*Mode, Window, Branch, Identity*)

The return type is LONG.

Execution

Mode	Mnemonic	Action
3	CLEAR	Clear the entire data points collection. Return: 1 if OK, 0 if NOK.
4	GET_COUNT	Return: The number of data points in the collection.
11	RESET_COLLECTED	Reset a collected slice to its component slices. Return: 1 if OK, 0 if NOK.
13	RESET_EXPLODED	Reset all exploded slices. Return: 1 if OK, 0 if NOK.
15	COPY_CLIPBOARD	Copy an image representing the chart to the clipboard. Return: 1 if OK, 0 if NOK.

Syntax 4

LongVal = PIE(*Mode, Window, Branch, Identity, Index, Red, Green, Blue[, Opacity]*);

The return type is LONG.

Argument	Meaning
Red,Green,Blue	An RGB color as three integers. Type INTEGER.
Opacity	The opacity used for alpha blending. Range 0 to 100%. Type INTEGER.

Execution

Mode	Mnemonic	Action
5	SET_COLOR	Set the color, and optional opacity, for the segment at the given index. Return: 1 if OK, 0 if NOK.

Syntax 5

LongVal =PIE(*Mode, Window, Branch, Identity, Index, Value*);

The return type is LONG.

Execution

Mode	Mnemonic	Action
6	SET_VALUE	Set the datapoint value for the segment at the given index Return: 1 if OK, 0 if NOK.

Syntax 6

LongVal = PIE(*Mode, Window, Branch, Identity, FilePath*);

The return type is LONG.

Argument	Meaning

FilePath	Either the full path name of a file, or just the file name itself if located in the project's TP folder. Type STR.

Execution

Mode	Mnemonic	Action
7	LOAD	Load a pie chart contents from a file. Return: 1 if OK, 0 if NOK.

Syntax 7

LongVal = PIE(*Mode, Window, Branch, Identity, Label, LegendText[, Tooltip]*);

The return type is LONG.

Execution

Mode	Mnemonic	Action
8	COLLECTED_LABEL	Specifies the texts of the label, legend and tooltip of the collected slice. If the ToolTip argument is not set then the tooltip will be the same than label text. Return: 1 if OK, 0 if NOK.

Syntax 8

IntVal = PIE(Mode, Window, Branch, Identity, Red, Green, Blue[, Opacity]);

The return type is INTEGER.

Argument	Meaning
Red,Green,Blue	An RGB color as three integers. Type INTEGER.
Opacity	The opacity used for alpha blending. Range 0 to 100%. Type INTEGER.

Execution

Mode	Mnemonic	Action
9	COLLECTED_COLOR	Sets the color and the opacity of the collected slice Return: 1 if OK, 0 if NOK.

Syntax 9

LongVal = PIE(*Mode, Window, Branch, Identity, ThresholdValue[, UsePercent]*);

The return type is LONG.

Argument	Meaning
ThresholdValue	The threshold value under which segments are grouped. Type LONG.
UsePercent	A flag, either 0 or 1. Type INTEGER.

Execution

Mode	Mnemonic	Action
10	COLLECTED_THRESH	Specifies the threshold value under which the slices will be grouped. If UserPercent = 1 then ThesholdValue is a percent and must be between 0 and 100. If UserPercent = 0 then ThesholdValue is an absolute value and can be any positive value. Return: 1 if OK, 0 if NOK.

Syntax 10

LongVal = PIE(*Mode, Window, Branch, Identity, Index[, Exploded]*)

The return type is LONG.

Argument	Meaning
Exploded	The sub-mode. Type INTEGER. 1 = the slice at the given Index is exploded. 0 = then the slice at the given Index is no longer exploded. -1 = the collected slice is exploded.

Execution

Mode	Mnemonic	Action
12	EXPLODED_SLICE	Manage exploded segments. Return: 1 if OK, 0 if NOK.

Syntax 11

LongVal = PIE(*Mode, Window, Branch, Identity[, Orientation][, Printer]*)

The return type is LONG.

Argument	Meaning
Orientation	The orientation for the print. Type INTEGER. 1 = landscape. 2 = portrait.
Printer	The printer name. Type STR.

Execution

Mode	Mnemonic	Action
13	PRINT	Print the chart on the selected printer. If the printer name is omitted the Windows' default printer is used. Return: 1 if OK, 0 if NOK.

Syntax 12

LongVal = PIE(*Mode, Window, Branch, Identity, ImageFile[, ImageFormat]*)

The return type is LONG.

Argument	Meaning
ImageFile	Either the full path name of a file, or just the file name itself if located in the project's TP folder. Type STR.
ImageFormat	The image format. Type INTEGER. 0 = BMP (default) 1 = GIF 2 = PNG

Execution

Mode	Mnemonic	Action
16	SAVE_IMAGE	Save a copy of the chart as an image in the specified format. Return: 1 if OK, 0 if NOK.

Syntax 13

LongVal = PIE(*Mode, Window, Branch, Identity, ChartType[, Radius]*)

The return type is LONG.

Argument	Meaning
ChartType	The chart type. Type INTEGER. 0 = Pie / Doughnut

	1 = Pyramid
Radius	The inner radius of the doughnut. Default 0. Type INTEGER.

Execution

Mode	Mnemonic	Action
16	SET_CHARTTYPE	Set the appearance of the chart from either pie, doughnut or pyramid. Return: 1 if OK, 0 if NOK.

Syntax 14

LongVal = PIE(*Mode, Window, Branch, Identity, 3dDisplay*)

The return type is LONG.

Argument	Meaning
3dDisplay	A flag indicating the display mode. Type INTEGER. 0 = 2D. 1 = 3D.

Execution

Mode	Mnemonic	Action
16	SET_3DDISPLAY	Set the appearance of the chart from either 2D or 3D. Return: 1 if OK, 0 if NOK.

Syntax 15

LongVal = PIE(*Mode, Window, Branch, Identity, Property, PropertyValue*)

The return type is LONG.

Argument	Meaning
Property	The name of the property to modify. See the table below for the possible values. Type STR.
PropertyValue	The new value for the property. See the table below for the possible values. Type INTEGER.

Name	Value
Palette	0 = Custom 1 to 12 = Standard palette
LabelPosition	0 = Inside 1 = Outside 2 = Disabled 3 = Outside in column (Pyramid)
LabelFormat	0 = text and value 1 = text and value percent 2 = text 3 = value 4 = value percent
LegendFormat	0 = text and value 1 = text and value percent 2 = text 3 = value 4 = value percent
Pie_2dStyle	2d only. 0 = default 1 = convex 2 = concave

Pie_Radius	0 to 99
Pie_Angle	0 to 360
Pyr_ValueType	0 = linear value type 1 = surface value type
Pyr_3dBase	3d only 0 = square base 1 = circular base
Pyr_3dAngle	3d only -10 to 10
Pyr_Gap	0 to 100
Pyr_Label_InsidePos	0 = center 1 = top 2 = bottom
Pyr_Label_OutsidePos	0 – right 1 - left

Execution

Mode	Mnemonic	Action
17	CUSTOM_PROPERTY	Change the chart appearance using a set of properties and values. The properties prefixed PIE are applicable to the pie / doughnut chart type and the ones prefixed PYR are applicable to the pyramid. Return: 1 if OK, 0 if NOK.

Syntax 16

LongVal = PIE(*Mode, Window, Branch, Identity, Title*)

The return type is LONG.

Argument	Meaning
Title	The title for the chart. Type STR.

Execution

Mode	Mnemonic	Action
16	SET_TITLE	Set the chart main title. Return: 1 if OK, 0 if NOK.

Syntax 17

LongVal = PIE(*Mode, Window, Branch, Identity[, Show]*)

The return type is LONG.

Argument	Meaning
Show	A flag. Type INTEGER. 0 = Hide the legend. Default setting. 1 = Show the legend.

Execution

Mode	Mnemonic	Action
21	SHOW_LEGEND	Show or hide the chart legend. Return: 1 if OK, 0 if NOK.

Syntax 18

LongVal = PIE(*Mode, Window, Branch, Identity, Path[, Area[, WrapMode[, Position]]]*)

The return type is LONG.

Argument	Meaning
Path	Full path of an image file. Type STR.
Area	The position of the image. Type INTEGER 0 = Plot area 1 = Legend area 2 = Border area 3 = Entire chart area
WrapMode	How the image is displayed. Type INTEGER 0 = Image is scaled. This is the default. 1 = Image is tiled 2 = Actual size
Position	Position of the image when displayed as actual size. 0 = Center 1 = Center left 2 = Center right 3 = Bottom 4 = Bottom left 5 = Bottom right 6 = Top 7 = Top left 8 = Top right

Execution

Mode	Mnemonic	Action
22	SET_BACKIMAGE	Display an image as the chart background. Return: 1 if OK, 0 if NOK.

POPULATION

See Also Example Defining a population

A Population is a filter which may be applied to the database of a station (using the instruction STATION_FILTER) to control the distribution of variable values and calculation of alarm synthesis.

The POPULATION instruction allows populations to be created using the program language. Populations may also be created interactively from the POPULATION node in the Application Explorer.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	CREATE	1
2	ADDLINE	2
3	DUMP	3

Syntax 1

IntVal = POPULATION(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a memory buffer.

Execution

Mode	Mnemonic	Action
1	CREATE	One or more populations are created using the definition supplied by the contents of the memory buffer. The contents of the buffer can be created using either Seq_Buffer or Population mode ADDLINE. See example. Return: 1 if successful, else 0.

Defining the Population

See the topic Defining a Population.

Syntax 2

IntVal = POPULATION(Mode, *Handle*, *Line*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a memory buffer.
Line	A line of a filter. Type STR.

Execution

Mode	Mnemonic	Action
2	ADDLINE	To add a line to a filter. One or more ADDLINE instructions must be followed by a CREATE instruction to complete the population creation. Return: 1 if successful, else 0.

Syntax 3

IntVal = POPULATION(*Mode*, *Name*);

The return type is INTEGER.

Argument	Meaning

Name See Syntax 1.

Execution

Mode	Mnemonic	Action
3	DUMP	The definition of the named population is displayed in the results area of the Program Management dialog box. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

POW

See Also

Power function.

WebVue support - Yes.

Syntax

DblVal = POW(*x*, *y*);

The return type is DOUBLE.

Argument	Meaning
x andy	Any numeric type.

Execution

Return: *x* to the power of *y*.

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
SUB Main()
DIM dblResult as double;
DIM dblValue1 as double;
DIM dblValue2 as double;
dblValue1 = 2.5;
dblValue2= 2;
dblResult = POW(dblValue1,dblValue2);
PRINT("Result of 2.5^(2) = ",dblResult);
'Display "Result of 2.5^(2) = 6.25"
END SUB
```

PRINT

See Also

Display text strings in the debug output of the Program Management window.

WebVue support - Yes.

Syntax

IntVal = PRINT(*Data1*[, *Data2*....[, *Data10*]]);

The return type is INTEGER.

Argument	Meaning
Data1 to Data10	The information to be printed. Maximum 10 items. All variable types are supported.

Execution

The values of the supplied data items are displayed in the program monitor window.

The display format for different types of data is as follows:

Data	Format
STR	The same as that passed as a parameter.
INTEGER	Signed integer.
LONG	Signed long.
SINGLE	[-]dddd.dddddd. The number of digits before the decimal point depends on the size of the number.
DOUBLE	[-]dddd.dddddd or [-]d;dddE[sign]ddd.
CONST	[-]dddd.dddddd or [-]d;dddE[sign]ddd. Return: Always 1.

Example

```
SUB Main()
DIM dblValue1 as double;
DIM dblValue2 as double;

dblValue1 = 2.5;
dblValue2= 2;

PRINT(dblValue1,dblValue2);
'Displays "2.52"
PRINT("dblValue1",dblValue2);
'Displays "dblValue12"
PRINT("dblValue1 ",dblValue2);
'Displays "dblValue1 2"
PRINT("dblValue1 = ",dblValue1);
'Displays "dblValue1 = 2.5"
PRINT("Hello,"," ","World","!");
'Displays "Hello, World!"
END SUB
```

PRINTER

See Also Example

Management of printers.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	PAUSE	1
1	REPLAY	1
2	CLEAR	1
3	GETNB	1
4	STATUS	1
5	SELECT	2
6	FLUSH	1
7	CLOSE	1
8	OPEN	1

Syntax 1

IntVal = PRINTER(*Mode*, *ID*);

Argument	Meaning
ID	The number of the printer (from 1 to 8).

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	PAUSE	Set the printer out of service. This suspends printing, then restarts printing after the wait period for recovery. The timer starts when PAUSE is run.
1	REPLAY	Set the printer in service, i.e. restart printing without losing data in the print buffer.
2	CLEAR	Clear all messages in the print spooler buffer. All unprinted messages will be lost. Return: always 1.
3	GETNB	Return the number of unprinted lines in the print spooler buffer.
4	STATUS	Return the printer status 1 = in service, else 0
6	FLUSH	If print spooler is activated then all the messages are sent to the printer. If the print spooler is not activated the request is queued and is executed when the spooler is next activated.
7	CLOSE	Take the printer out of use and empty the print buffer. The Supervisor no longer stores in memory the events to be printed.
8	OPEN	Bring the printer into use. Return: 1 if OK, else 0.

Modes 0 and 1: PAUSE replaces OFF and REPLAY replaces ON.

The maximum size of the printer buffer is 2,000 messages.

Syntax 2

IntVal = PRINTER(*Mode*, *Main*, *Standby*);

Argument	Meaning
Main	The number of the printer as in the first field of the printer definition in the file PARAM.DAT.
Standby	The number of the standby printer (as in the 16th field of the printer definition in PARAM.DAT).

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
5	SELECT	Select the address path of the printer on the network (used for redundant printers). Return: 1 if OK, else 0.

Example

```
SUB PRN1HS() 'pause printing on PRN1 and
PRINTER("OFF", 1); 'resume printing on PRN2
PRINTER("ON", 2);
END SUB

SUB PRN1RS() 'resume printing on PRN2 and
PRINTER("OFF", 2); 'pause printing on PRN1
PRINTER("ON", 1);
END SUB
```

For further examples, select the Example link above.

PROGRAM

See Also Example

Executed, load, unload or stop a program or function.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	STOP	1
1	START	1, 3
2	IS_LOADED	1
3	PRELOAD	1, 3
4	UNLOAD	1
5	EXECUTE	1
9	FUNCTION	2
10	IS_FUNCTION	2

Syntax 1

IntVal = PROGRAM(*Mode*, *ProgName*, *Branch*) ;

The return type is INTEGER.

Argument	Meaning
ProgName	The name of a program.
Branch	The branch for the program.

Execution

Mode	Mnemonic	Action
0	STOP	Stop the execution of a program and unload it. Return: 1 if successful, 0 if the program is not loaded.
1	START	Load a program and execute it. Return: 1 if successful, 0 if the program is already loaded.
2	IS_LOADED	Test if a program is loaded. Return: 1 if loaded, else 0.
3	PRELOAD	Load a program. Return: 1 if successful, 0 if the program is already loaded.
4	UNLOAD	Unload a program. Return: 1 if successful, 0 if the program is not loaded.
5	EXECUTE	Execute a program. Return: 1 if successful, 0 if the program is not loaded.

Syntax 2

IntVal = PROGRAM(*Mode*, *ProgName*, *Branch*, *Function*[,*Farg*]) ;

The return type is INTEGER.

Argument	Meaning
ProgName	The name of a program.
Branch	The branch for the program.
Function	The name of a function in the program *ProgName*.
Farg	Optional. From 1 to 8 arguments, comma-separated, up to 2,047 characters. Type STR.

Execution

Mode	Mnemonic	Action
9	FUNCTION	Runs the specified function. If the name of the function is not specified then MAIN is executed. Return: 1 if successful, 0 if the program is not loaded.
10	IS_FUNCTION	Check whether the function exists. Return: 1 if successful, 0 if the function does not exist.

The name of the branch is not accessible in the MAIN sub-program.

The function must be pre-loaded before it can be started. If the function call is to include a branch the function must be first pre-loaded with the same branch.

Syntax 3

IntVal = PROGRAM(*Mode*, *ProgName*, *Branch*, [,Dynamic]) ;

The return type is INTEGER.

Argument	Meaning
ProgName	The name of a program.
Branch	The branch for the program.
Dynamic	Optional. 0 for normal operation, 1 for dynamic load.

Execution

Mode	Mnemonic	Action
1	START	Load a program and execute it. If the optional parameter Dynamic is 1, the program is loaded and started even if a copy of it is already loaded. Return: 1 if successful, else 0
3	PRELOAD	Load a program. If the optional parameter Dynamic is 1, the program is loaded even if a copy of it is already loaded. Return: 1 if successful, else 0.

Example

```
IF(PROGRAM("IS_LOADED", "prog1", "")==0) THEN
PROGRAM("START", "prog1", "");
ENDIF
```

For further examples, select the Example link above.

PUT_BUFFER

See Also

Store data in a memory buffer.

WebVue support - Yes.

Syntax

PUT_BUFFER(*Handle*, *Offset*, *Data*);

There is no return type.

Argument	Meaning
Handle	The location of the memory buffer as returned by ALLOC_BUFFER or FILETOBUF. Type LONG.
Offset	The offset in bytes at which the data is to be placed. Any numeric type.
Data	The data to be put in the buffer. Type STR.

Execution

The data is stored in a memory buffer at the specified offset.

Example

```
SUB Main()
'Declare variables
DIM lngBuffer1 as Long;
DIM strLine as Str;
DIM intLength as Integer;

lngBuffer1 = ALLOC_BUFFER(110);
strLine = "123.34;string_here;345;123456789;,t,\n";
PUT_BUFFER(lngBuffer1, 0, strLine);
intLength = ASCIIFIELD("LEN",lngBuffer1);
PRINT("Length: ",intLength);  'shows: Length: 110
FREE_BUFFER(lngBuffer1);
END SUB
```

Q

(No topics)

There are no topics in this book.

R

RECIPE

See Also Example Further information

Recipe management.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	EXIST	1
2	SEND	1, 5
3	REALTIME	1, 2
4	READ	3
5	CREATE	4
6	SENDEVEN	1, 5
7	REALTIMEEVEN	1, 2
8	STARTWATCH	6, 7
9	STOPWATCH	1, 5, 10
10	REALTIMECREATE	5, 8
11	REALTIMECREATEEVEN	5, 8
12	REFRESH	9
14	NETWORKBROADCAST	11
15	NETWORKBROADCASTALL	12

The values for Recipe access rights are described in the topic Access Rights Weighting.

The maximum size for the recipe file for mode SEND is 128 Kb.
The maximum size for the recipe file for mode NETWORKBROADCAST is 64 Kb.

Syntax 1

IntVal = RECIPE(*Mode*, *Identifier*);

The return type is INTEGER.

Identifier The identity of the recipe. This can be either its number or its title. See the topic 'Configuring Recipes'. Type STR.

Execution

Mode	Mnemonic	Action
1	EXIST	Test whether the recipe exists. Return: 1 if the recipe number exists. 2 if the recipe title exists. 0 if the recipe does not exist.
2	SEND	Send a recipe. The restrictions, which are normally imposed when sending a recipe from an animation, apply, e.g. the user must have the correct rights and the enable bit must be set. Return: 1 if successful, else 0.
3	REALTIME	Modify the recipe by substituting the current (real time) value of each variable within its list. Return: 1 if successful, else 0.
6	SENDEVEN	The same as Mode 2 except that the recipe is sent even if some of the variables are invalid or out of limits. Return: 1 if successful, else 0.
7	REALTIMEEVEN	The same as Mode 3 except that the real time values are substituted even if some of them are invalid or out of limits.

		Return: 1 if successful, else 0.
9	STOPWATCH	Stop the refresh of variables in the recipe (As started with a STARTWATCH)
		Return: 1 if successful, else 0.

Syntax 2

IntVal = RECIPE(*Mode*, *Identifier*, *NewNum*, *NewTitle*, *Flag*, *File*);

The return type is INTEGER.

Argument	Meaning
Identifier	The identity of the recipe. This can be either its number, or its title. See the topic 'Configuring Recipes'. Type STR.
NewNum	The number for the new recipe. Type STR.
NewTitle	The title for the new recipe. Type STR.
Flag	Save the new recipe in a file. (0 or 1).
File	The name of the file in which the recipe is to be saved.

Execution

Mode	Mnemonic	Action
3	REALTIME	A new recipe is created using the variable list from the recipe specified by the identifier, but with the current (real time) value for each of the variables. If a file is specified it will be saved. An existing recipe with the same identifier it will be overwritten.
		Return: 1 if successful, else 0.
7	REALTIMEEVEN	The same as Mode 3 except that the real time values are substituted even if some of them are invalid or out of limits.
		Return: 1 if successful, else 0.

The name of the file in which the recipe is to be saved cannot be renamed by this command, i.e. the parameters*NewNum*, *NewTitle* and *File*should be kept unchanged.

Syntax 3

IntVal = RECIPE(*Mode*, *Identifier*, *Handle*, *Size*);

The return type is INTEGER.

Argument	Meaning
Identifier	The identity of the recipe. This can be either its number, or its title. (See the topic 'Configuring Recipes'.) Type STR.
Handle	The location of a memory buffer as returned by ALLOC_BUFFER. Type LONG.
Size	The size of the recipe file. This must be smaller than the buffer size or a run time error will occur.

Execution

Mode	Mnemonic	Action
4	READ	The contents of a recipe are read into a memory buffer.
		Return: 1 if successful, else 0.

Syntax 4

IntVal = RECIPE(*Mode*, *Handle*, *NewNum*, *NewTitle*, *Flag*, *File*);

The return type is INTEGER.

Argument	Meaning
Handle	The location of a memory buffer as returned by ALLOC_BUFFER. Type LONG.
NewNum	The number for the new recipe. Type STR.
NewTitle	The title for the new recipe. Type STR.
Flag	Save the new recipe in a file. (0 or 1).
File	The name of the file in which the recipe is to be saved (if Flag = 1).

Execution

Mode	Mnemonic	Action
5	CREATE	A new recipe is created using the contents of a memory buffer. If a file is specified it will be saved. Any existing recipe with the same identifier will be overwritten. Return: 1 if successful, else 0.

Syntax 5

IntVal = RECIPE(*Mode*, *Identifier*, *Branch*);

The return type is INTEGER.

Identifier The identity of the recipe. This can be either its number, or its title. (See the topic 'Configuring Recipes'.) Type STR.

Branch A branch that will be prefixed to all variable names in the recipe list. Type STR.

Execution

Mode	Mnemonic	Action
2	SEND	Send a recipe. The restrictions, which are normally imposed when sending a recipe from an animation, apply, e.g. the user must have the correct rights and the enable bit must be set. Return: 1 if successful, else 0.
6	SENDEVEN	Send a recipe. The same as Mode 2 except that the recipe is sent even if some of the variables have an invalid or out of limits value. Return: 1 if successful, else 0.
9	STOPWATCH	Stop the refresh of variables in the recipe (As started with a STARTWATCH) Return: 1 if successful, else 0.
10	REALTIMECREATE	Modify the recipe by substituting the current (real time) value of each variable within its list. Return: 1 if successful, else 0.
11	REALTIMECREATEEVEN	Modify the recipe by substituting the current (real time) value of each variable within its list, even if some of the variables have an invalid or out of limits value. Return: 1 if successful, else 0.

Syntax 6

WatchID = RECIPE(*Mode*, *Identifier*, *Varname*, *Status*);

The return type is INTEGER.

Argument	Meaning
Identifier	The identity of the recipe. This can be either its number, or its title. (See the topic

	'Configuring Recipes'.) Type STR.
Varname	A name of a bit variable in the database. Type STR.
Status	Either 0 or 1.

Execution

Mode	Mnemonic	Action
8	STARTWATCH	This mode is used to refresh any variables in a recipe that do not have the option 'Automatic scan for mimics' enabled in their configuration. The bit 'Varname' is set to the value in 'Status' when the function is executed. Return: The return is a unique number representing the STARTWATCH demand, and must be remembered if using STOPWATCH syntax 10. The return type is LONG.

Syntax 7

WatchID = RECIPE(*Mode*, *Identifier*, *Varname*, *Status*, *Branch*);

The return type is INTEGER.

Argument	Meaning
Identifier	The identity of the recipe. This can be either its number, or its title. Type STR.
Varname	A name of a bit variable in the database. Type STR.
Status	Either 0 or 1.
Branch	A branch that will be prefixed to all variable names in the recipe list. Type STR.

Execution

Mode	Mnemonic	Action
8	STARTWATCH	This mode is used to refresh any variables in a recipe that do not have the option 'Automatic scan for mimics' enabled in their configuration. The bit 'Varname' is set to the value in 'Status' when the function is executed. Return: The return is a unique number representing the start watch demand, and must be remembered for use with the STOPWATCH mode. The return type is long.

Syntax 8

IntVal = RECIPE(*Mode*, *Identifier*, *Branch*, *NewNum*, *NewTitle*, *File*);

The return type is INTEGER.

Identifier	The identity of the recipe. This can be either its number, or its title. (See the topic 'Configuring Recipes'.) Type STR.
Branch	A branch that will be prefixed to all variable names in the recipe list. Type STR.
NewNum	The number for the new recipe. Type STR.
NewTitle	The title for the new recipe. Type STR.
File	The name of the file in which the recipe is to be saved.

Execution

Mode	Mnemonic	Action
10	REALTIMECREATE	A new recipe is created using the variable list from the recipe specified by the identifier, but with the current (real time) value for each of the variables. If a file is specified it will be saved. An existing recipe with the same identifier it will be overwritten.

		Return: 1 if successful, else 0.
11	REALTIMECREATEEVEN	Identical to Mode 10 except that the action will be performed even if some of the variable values are out of limits or invalid. Return: 1 if successful, else 0.

The name of the file in which the recipe is to be saved cannot be renamed by this command, i.e. the parameters*NewNum*, *NewTitle* and *File* should be kept unchanged.

Syntax 9

IntVal = RECIPE(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
12	REFRESH	Force the recipes to be reread from the configuration files in the project's 'R' folder. This is to allow the recipe configuration to be regenerated after the recipe source files have been modified by an external program or application. Return: 1 if successful, else 0.

Syntax 10

IntVal = RECIPE(*Mode*, *WatchID*);

The return type is INTEGER.

WatchID The identity of a previously issued STARTWATCH function. Type LONG.

Execution

Mode	Mnemonic	Action
9	STOPWATCH	Stop the refresh of variables in the recipe started with the STARTWATCH function identified with the WatchID. Return: 1 if successful, else 0.

Syntax 11

IntVal =RECIPE(*Mode*, *RecipeID*, *List*, *HStatus*, *EventVar, LocalRemote*);

Argument	Meaning
RecipeID	The name or number of a recipe. Type STR.
List	The name of the list containing the names of the network stations to receive modifications (as created from the Configure.Communication.Network Stations command). Type STR.
HStatus	The handle of a buffer allocated by ALLOC_BUFFER. After the instruction has been executed the buffer will contain a list of stations and their update status in the format: StaNum,Status,...;.,...;...; where StaNum is the number of the station and Status is 1 if successful, else 0.
EventVar	The name of a database bit variable that will be set to 1 when the instruction has been executed. If an event variable is not required a null string ("") must be used as this argument cannot be omitted. Type STR.
LocalRemote	A flag to determine where the changed recipe configuration is applied: 1: All stations (including the local station). 0: Remote stations only. (Default)

Execution

Mode	Mnemonic	Action
14	NETWORKBROADCAST	Sends the configuration of the selected recipe to all the stations contained in the station list. Return: 1 successful, else 0.

Syntax 12

IntVal =RECIPE(*Mode*, *List*, *HStatus*, *EventVar*, LocalRemote);

Argument	Meaning
List	The name of the list containing the names of the network stations to receive modifications (as created from the Configure.Communication.Network Stations command). Type STR.
HStatus	The handle of a buffer allocated by ALLOC_BUFFER. After the instruction has been executed the buffer will contain a list of stations and their update status in the format of *StaNum*,*Status*, etc.; where *StaNum* is the number of the station and *Status* is 1 if successful, else 0. Type LONG.
EventVar	The name of a database bit variable that will be set to 1 when the instruction has been executed. If an event variable is not required a null string ("") must be used as this argument cannot be omitted.Type STR.
LocalRemote	A flag to determine where the changed recipe configuration is applied: 1: All stations (including the local station). 0: Remote stations only. (Default)

Execution

Mode	Mnemonic	Action
15	NETWORKBROADCASTALL	Sends the configuration of all recipes to all the stations contained in the station list. Return: 1 successful, else 0.

Example

For an example, select the Example link above.

REFRESH_DB

See Also

Asynchronous assignment of variable from an ASCII file.

WebVue support - Yes.

Mode	Mnemonic	Syntax
0	SETUP	1
1	LOAD	2

Syntax 1

IntVal = REFRESH_DB(*Mode*, *AssignTime*, *IdleTime*);

Argument	Meaning
AssignTime	The time period allowed to assign values to the list of variables. Default value 250 ms. Type INTEGER.
IdleTime	The time during which the process is paused. Default value 250 ms. Type INTEGER.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	SETUP	Change the delays used in assigning values to variables.

If the SETUP mode is used without any parameters the assignment is treated in one block. If values are to be assigned to a large number of variables, this could eventually stop the system.

Syntax 2

IntVal = REFRESH_DB(*Mode*, *Filename* [, *Type*]);

Filename Name of the file from which the assignment will be made. The syntax must be as follows:

id1, val1
id2, val2
.....
idn, valn

where:

id is the identification of the variable
val is the assignment value

Type Specifies the type of file:

0 The name of the variable is specified. (Default)
1 The internal ID of the variable is specified.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	LOAD	Start the assignment process. The default path for the file is the TP folder of the current project. Return: 1 if OK, else 0.

Example

```
SUB rdb_set()
  REFRESH DB("SETUP", 800, 1000);
END SUB
SUB rdb_go()
  DIM i as integer;
  i = 0;
  WHILE (i == 0)
    i = REFRESH_DB("LOAD", "C\\DLOAD\\tstvar.txt");
DELAY(2);
WEND
END SUB
```

REGION

See Also Example

Manage regions on a single display or multiple displays on a desktop client.

- A physical display may be divided into a number of virtual screens.
- On systems with a multi-screen graphics card (or multiple graphics cards), each region can be set up to appear on a different physical screen.

When a window is opened on a multi-region system, the position at which it opens is relative to the selected region.

WebVue support - Not applicable. Returns the unsuccessful code if used in this context.

Mode	Mnemonic	Syntax
1	SETSYSREGION	1
2	GETSYSREGION	2
3	SETREGION	1
4	GETREGION	2
5	SETSELECTION	3
6	GETSELECTION	2
7	GETSELECTIONMODE	2

Syntax 1

IntVal = REGION(*Mode*, *Regions*);

Argument	Meaning
Regions	The number of regions (mode 1), or a particular region (mode 3). Type INTEGER.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	SETSYSREGION	Dynamically set the number of regions.
3	SETREGION	Set the region to which subsequent SCADA BASIC operations will be directed. Return: -1 if error else OK.

If using multiple regions, you must set the region before executing any instructions that interacts with the HMI (WINDOW, ALARMDISPLAY etc.).

```
Dim iRegion As Integer;
iRegion = 1;
Region ("SETREGION", iRegion);
Window("OPEN", "Alarmwindow", "");
```

If you run a program from an animation, you can get the region in which the mimic is located using GETARG("VARSTATUS")

```
Dim iRegion As Integer;
iRegion = GETARG("VARSTATUS");
Region ("SETREGION", iRegion);
Alarmdisplay("ACK_SELECTED", "Alarmwindow", "", "Alarm1");
```

Syntax 2

IntVal = REGION(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	GETSYSREGION	Return the number of regions or 0 if unsuccessful.
3	GETREGION	Return the current region or 0 if unsuccessful.
6	GETSELECTION	Return the current selection or 0 if unsuccessful.
7	GETSELECTIONMODE	Return the selection mode.

Syntax 3

IntVal = REGION(*Mode*, *Regions*[, *SubMode*]);

Argument	Meaning
Regions	The number of regions. Type INTEGER.
SubMode	Any numeric type

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
5	SETSELECTION	Set the region in which all future window opening initiated by a Link Open animation will occur. The behavior depends on *SubMode* as follows: 0 The window is opened in the same region as the window in which the Link Open animation resides. 1 The window is always opened in the selected region. The selection is remembered even after the system has been shutdown and restarted. This is the default option. 2 The first Link Open animation selected after the instruction has been executed opens a window in the selected region. Subsequent Link Open animations operate as in mode 0. Return: -1 if error else OK.

Example

For an example, select the Example link above.

REGVAR2D

See Also

Control the use of register variables as type DOUBLE in a program.

WebVue support - Yes.

Syntax

REGVAR2D([*Flag*]);

There is no return type.

Argument	Meaning
Flag	Either 0 or 1

Execution

The action depends on the setting of *Flag*.

Setting	Meaning
1	Allows the use of register variables as type DOUBLE in a program. (Default)
2	Disallow the use of register variables as type DOUBLE in the program.

Example

```
'Variables
'@REGISTER01: Register type
'@REGISTER02: Register type

SUB Main()
REGVAR2D(1);
'The register variable is now type Double
'so this assignment works correctly:
@REGISTER01 = COS(45);

REGVAR2D(0);
'Automatic conversion to Double is disabled.
'The next line causes a runtime error
'because the Register variable is now Single:
@REGISTER02 = COS(45);
END SUB
```

RENAME

See Also

Rename a file.

WebVue support - Yes.

Syntax

IntVal = RENAME(*OldName*, *NewName*);

The return type is INTEGER.

Argument	Meaning
OldName	Current name of the file. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file is renamed on that computer.
NewName	New name of the file. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file is renamed on that computer.

Execution

Return: 1 if successful, else 0.

The file to be renamed must not be open.

Any files used by SCADA BASIC must reside in the project folder TP.

Example

```
'Before you run the program, create a file old.txt in the TP folder.
SUB Main()
If (RENAME("old.txt","new.txt")==0) Then
  'When the Rename command does not work,
  'that is because a file new.txt already exists, so delete it:
  If (UNLINK("new.txt")==1) Then
    RENAME ("old.txt","new.txt");
  Else
    'If Rename still does not work, the file may be open:
    Print("Check that the new file is not already open.");
  End If
End If
END SUB
```

REPLACE

See Also Example

Search for, and replace, a sub-string.

WebVue support - Yes.

Syntax

StrVal = REPLACE (*InputString*, *OldSubString*, *NewSubString*[, *CaseSensitive*[, *OccurrenceCount*]]);

Argument	Meaning
InputString	The input string. Maximum length 32000 characters. Type STR.
OldSubString	The sub-string to search for. Maximum length 2047 characters. Type STR.
NewSubString	The replacement sub-string. Maximum length 2047 characters. Type STR.
CaseSensitive	A flag indicating if the search is to be case sensitive. 0 = Ignore case (default). 1 = Match case.
OccurrenceCount	The maximum number of occurrences of the *OldSubString* to replace. Any numeric type.

Execution

Action

Search the *InputString* for occurrences of *OldSubString* and replace with *NewSubString*. Return type STR. Return: The modified string.

RETURN

See Also Example

Force an exit from a function.

WebVue support - Yes.

Syntax

RETURN(*Value*);

There is no return type.

Argument	Meaning
Value	A value returned by the function. Any variable type.

Execution

Control is returned to the calling function. A value may also be returned.

Example

For an example, select the Example link above.

RIGHT

See Also

Return a number of characters from the end (right hand side) of a string.

WebVue support - Yes.

Syntax

StrVal = RIGHT(*Input*, *N*);

The return type is STR.

Argument	Meaning
Input	The string to be manipulated. Type STR.
N	The number of characters to return. Any numeric type.

Execution

If *N* is greater than the length of the string, the entire string is returned.

Example

```
SUB Main()
DIM strResult as Str;
DIM strString as Str;

strString = "Hello, World!";
strResult = Right( strString ,6); 'take the last 6 characters
PRINT("Result: ", strResult );
'Displays "Result: World!"
END SUB
```

RTRIM

See Also

Return a copy of a string without any trailing spaces.

WebVue support - Yes.

Syntax

StrVal = RTRIM(*string*);

The return type is STR.

Argument	Meaning
string	The string whose trailing spaces are to be removed. Type STR.

Example

```
SUB Main()
'Declare variables
DIM strResult as Str;
DIM strValue as Str;

strValue = "Hello, World!                ";  'with trailing spaces
strResult = RTrim(strValue);
PRINT("RTrim(",strValue,") = ",strResult);
'Displays "RTrim(Hello, World!                ) = Hello, World!"
END SUB
```

S

SELECTOR

See Also Example Further information

Select the data that appears in a grid control animation.

WebVue support - No.

If using a project with multiple regions, you must set the region before executing any instructions that interacts with the HMI. For further information see the REGION topic.

How the Lines and Columns are numbered in a grid control

The data lines and columns are numbered starting at 0.

The header line(s) and column(s) (Fixed Row / Col in the Grid Properties dialog) are numbered starting at -1. Show picture

-1,-1	-1,0	-1,1	-1,2	-1,3	-1,4
0,-1	0,0	0,1	0,2	0,3	0,4
1,-1	1,0	1,1	1,2	1,3	1,4
2,-1	2,0	2,1	2,2	2,3	2,4
3,-1	3,0	3,1	3,2	3,3	3,4
4,-1	4,0	4,1	4,2	4,3	4,4

Modes

Mode	Mnemonic	Syntax
1	GETNBLINE	1
2	GETNBCOL	1
3	GETCELL	2
4	GETLINE	3
5	GETCOL	4
6	GETARRAY	5
7	PUTCELL	6
8	PUTLINE	3
9	PUTCOL	4
10	PUTARRAY	5
11	SCROLL	7
12	GETLISTART	1
13	GETNBLINEMAX	1
14	VARIABLE	9
15	VARMODE	10
16	SELECTLINE	1
17	SELECTCOL	1
18	SELECTMODE	7
19	SFIRSTCELL	1
20	SNEXTCELL	1
21	ALLSELOFF	1
22	CLICKCELL	2
23	GETSEL	2
24	SELPROG	11
25	INPUTPROG	11
26	CLEAR	1
27	SETNBLINE	8

28	GETLASTCELL	1
30	SORT	12
31	GETCELLBACKCOLOR	13
32	GETCELLTEXTCOLOR	13
33	PUTBACKCOLOR	14
34	PUTTEXTCOLOR	14
35	GETVARNAME	15
36	HISTORICAL	See the topic SELECTOR mode HISTORICAL

All Syntaxes

Argument	Meaning
Window	The name of the window which contains the grid control animation which is to be used. Type STR.
Branch	The branch of the window (if any). Using a "*" means the current branch of the program. Type STR.
Identity	The identity of the grid control animation within the specified window. Type STR.

Syntax 1

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	GETNBLINE	Return the number of lines that may be displayed at any one time. (Dependent on the size of the support drawing element.) A return of 0 is unsuccessful.
2	GETNBCOL	Return the number of columns. A return of 0 is unsuccessful.
12	GETLISTART	Return the index of the first visible line at the top of the grid control.
13	GETNBLINEMAX	Return the total number of lines present in memory.
16	SELECTLINE	Return the line index of the cell currently selected. Returns -1 if no cell selected.
17	SELECTCOL	Return the column index of the cell currently selected. Returns -1 if no cell selected.
19	SFIRSTCELL	For a multiple selection, find the first selected cell searching from the beginning of the table. Use modes 16 (SELECTLINE) and 17 (SELECTCOL) to obtain the position of the cell found. Returns 1 if a cell as been found else 0.
20	SNEXTCELL	For a multiple selection, find the next selected cell starting from the last cell found with the same function or with mode 19 (SFIRSTCELL). Use modes 16 (SELECTLINE) and 17 (SELECTCOL) to obtain the position of the cell found. The search is made from left to right then top to bottom. Return 1 if a cell as been found else 0.
21	ALLSELOFF	Deselect all the selected cells of a table. Returns 1 if successful, else 0.
26	CLEAR	Reset the table. Returns 1 if successful, else 0.
28	GETLASTCELL	Return the number of lines used.

Syntax 2

StrVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Line*, *Column*);

Argument	Meaning
Line	The index of a line. Type INTEGER.
Column	The index of a column. Type INTEGER.

Execution

Mode	Mnemonic	Action
3	GETCELL	Return the contents of the cell pointed to by the line and column index. Line and column indices both start at 0. The return type is STR.
22	CLICKCELL	Simulate a click of the mouse on the cell specified by *Line* and *Column*. Returns 1 if successful, else 0. The return type is INTEGER.
23	GETSEL	Check whether a cell is selected. Returns 1 if the cell is selected, else 0. The return type is INTEGER.

Syntax 3

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Line*, *Hbuf*, *Csepa*);

The return type is INTEGER.

Argument	Meaning
Hbuf	The location (handle) of a memory buffer as returned by ALLOC_BUFFER. Type LONG.
Csepa	A single character used as a delimiter for the text in each column. Type STR.

Execution

Mode	Mnemonic	Action
4	GETLINE	Get the contents of the indicated line and store in a memory buffer.
8	PUTLINE	Write the contents of the indicated line using the contents of a memory buffer.

Syntax 4

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Col*, *Hbuf*, *Lsepa*);

The return type is INTEGER.

Argument	Meaning
Lsepa	A single character used as a delimiter at the end of each line. Type STR.

Execution

Mode	Mnemonic	Action
5	GETCOL	Get the contents of the indicated column and store in a memory buffer.
9	PUTCOL	Write the contents of the indicated column using the contents of a memory buffer.

Syntax 5

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Hbuf*, *StartLine*, *StartCol*, *EndLine*, *EndCol*, *CLsepa*);

The return type is INTEGER.

Argument	Meaning
Hbuf	The location (handle) of a memory buffer as returned by ALLOC_BUFFER. Type LONG.
StartLine	The index of a line. Type INTEGER.

StartCol	The index of a column. Type INTEGER.
EndLine	The index of a line. Type INTEGER.
EndCol	The index of a column. Type INTEGER.
CLsepa	Two characters used as delimiters for the text in each column and line. Type STR.

Execution

Mode	Mnemonic	Action
6	GETARRAY	Get the contents of the cells indicated by *StartLine*, *StartCol* and *EndLine*, *EndCol* and store in a memory buffer.
10	PUTARRAY	Write the contents of the cells indicated by *StartLine*, *StartCol* and *EndLine*, *EndCol* using the contents of a memory buffer.

In both modes the two characters supplied by *CLsepa* are used as column and line delimiters.

Syntax 6

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Line*, *Column*, *Chain*);

The return type is INTEGER.

Argument	Meaning
Line	The index of a line. Type INTEGER.
Column	The index of a column. Type INTEGER.
Chain	A character string. Type STR.

Execution

Mode	Mnemonic	Action
7	PUTCELL	Write the supplied chain into the indicated cell.

Syntax 7

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *OpMode* [, *LineNo*]);

The return type is INTEGER.

Argument	Meaning
OpMode	A number indicating (according to the Mode selected) the operational mode for the grid control.
LineNo	A number indicating a specific line. Type INTEGER.

Execution

Mode	Mnemonic	Action
11	SCROLL	Scroll the contents of the grid control within the area provided by the support drawing element. The *OpMode* argument provides the scroll mode: 1 TOBEGIN 2 PAGEUP 3 LINEUP 4 LINEDOWN 5 PAGEDOWN 6 TOEND 7 TOLINE. A specific line number is displayed. 8 TOFIRSTCOL 9 PAGELEFT 10 COLLEFT 11 COLRIGHT

12 PAGERIGHT
13 TOLASTCOL
14 TOCOL A specific column number is displayed.

Return: 1 if action OK, else 0.

18	SELECTMODE	Change the response of the cells to user actions. The *OpMode* argument defines the behaviour: 0 OFF 1 MONO 3 MULTI 5 INPUT Return: 1 if OK, else 0.

Syntax 8

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *LineNo*);

The return type is INTEGER.

Argument	Meaning
LineNo	A number indicating a specific line. Type INTEGER.

Execution

Mode	Mnemonic	Action
27	SETNBLINE	Define the last line that can be made visible by scrolling. If the lines currently visible are greater than the new *LineNo* value, the table is position to the new value. Return: 1 if OK, else 0.

Syntax 9

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Filter*, *VarType*);

The return type is INTEGER.

Argument	Meaning
Filter	A filter, applied to the variable name, for the variables that are displayed. Depending on how the grid control is configured, either: • A regular expression - For details, see the topic Regular Expressions. • An SQL expression. See the topic Using an SQL Expression to Filter the List of Variables in the HMI help book.
VarType	An integer defining the class of variable: 1 Bits 2 Alarms 4 Registers 8 Text

Any combination may be used, for example, 3 (1 + 2) would correspond to Bits and Alarms.

Execution

Mode	Mnemonic	Action
14	VARIABLE	Display a list of variables and their real time values in a two column grid control. The variables which are displayed are filtered by regular expression (see following note) and an integer corresponding to the class of variable.

Return: 1 if successful, else 0.

To use this mode, the variable tracking property of the grid control must first be set.

If a variable has the command attribute set the background colour of its value display will correspond to that in the grid control definition for a selected cell. The value of the variable may be changed by clicking on the cell displaying its value.

Syntax 10

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *VarType*);

The return type is INTEGER.

Argument	Meaning
VarType	An integer defining what to display for the variable: 0 Name 1 Title 2 Name and title

Execution

Mode	Mnemonic	Action
15	VARMODE	Display a list of variable names in the first column of a grid control. Return: 1 if successful, else 0.

Syntax 11

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *program*, *pbranch*, *function* [*farg*]);

Argument	Meaning
program	Program name. Type STR.
pbranch	Branch name. Type STR.
function	Function name. Type STR.
Farg	Optional. String of 2,047 characters maximum, used to give from 1 to 8 arguments (separated by ",") to the function using the verb GETARG.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
24	SELPROG	Specify a function that will be executed when a cell of the table is clicked. The table must be in MONO or MULTI mode. SELECTLINE and SELECTCOL are used to define the cell. Return: 1 if OK, else 0.
25	INPUTPROG	Specify a function that will be executed when a cell of the table is clicked. A dialog box provides the operator with the option to confirm or cancel. The table must be in INPUT mode. SELECTLINE and SELECTCOL are used to define the cell. Return: 1 if OK, else 0.

Syntax 12

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Column*, *Flag*);

The return type is INTEGER.

Argument	Meaning
Column	Column number. Type INTEGER.
Flag	A flag indicating the sort order. Type INTEGER.

Execution

Mode	Mnemonic	Action
30	SORT	The contents of the table are sorted using the contents of the specified column. The sort order is according to the ASCII code of the characters in each field: 0 or greater: ascending less than 0: descending. Return: 1 is successful, else 0.

When using the grid control to display the value of variables in the database, only variables in the visible area of the grid control are subscribed and have their real time values updated.

Syntax 13

Color = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *Line*, *Column*);

Argument	Meaning
Line	The index of a line. Type INTEGER.
Column	The index of a column. Type INTEGER.
Color	A color as a byte weighted value to code an RGB value. Red is byte 0, green byte 1 and blue byte 2. For example, red is 255, green 65280 and yellow (red + green) 65535. Type LONG.

Execution

Mode	Mnemonic	Action
31	GETCELLBACKCOLOR	Return the background color of a cell as a byte weighted value. The return type is LONG.
32	GETCELLTEXTCOLOR	Return the text color of a cell as a byte weighted value. The return type is LONG.

Syntax 14

IntVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *StartLine*, *StartCol*, *EndLine*, *EndCol*, *Color*);

The return type is INTEGER.

Argument	Meaning
StartLine	The index of a line. Type INTEGER.
StartCol	The index of a column. Type INTEGER.
EndLine	The index of a line. Type INTEGER.
EndCol	The index of a column. Type INTEGER.
Color	The color as a byte weighted value. Type LONG.

Execution

Mode	Mnemonic	Action
33	PUTBACKCOLOR	Change the background color of a range of cells to the color given by the *Color* parameter. The range of cells is defined by the *StartLine*, *StartCol*, *EndLine*, *EndCol* parameters.

34	PUTTEXTCOLOR	Change the text color of a range of cells to the color given by the *Color* parameter. The range of cells is defined by the *StartLine*, *StartCol*, *EndLine*, *EndCol* parameters.

Syntax 15

StrVal = SELECTOR(*Mode*, *Window*, *Branch*, *Identity*, *LineNo*);

The return type is STR.

Argument	Meaning
LineNo	A number indicating a specific line. Type INTEGER.

Execution

Mode	Mnemonic	Action
35	GETVARNAME	Returns the name of the variable in the specified line when the grid control is in variable tracking mode. Return: The variable name.

Example

For an example, select the Example link above.

SELECTOR mode HISTORICAL

See Also Example Further information

Select the data that appears in a grid control animation. This topic is only for the mode HISTORICAL. For all other modes see the main SELECTOR topic.

WebVue support - No.

If using a project with multiple regions, you must set the region before executing any instructions that interacts with the HMI. For further information see the REGION topic.

Sub-mode Mnemonic	Syntax
GETCOUNT	1
INSERTTREND	2
REMOVETREND	3
CLEARTRENDS	1
SETTREND	4
GETTRENDBYINDEX	5
GETINDEX	6
RESTOREDEFAULT	1
CANCEL	1
REQUEST	7
EXPORT	1

All Syntaxes

Argument	Meaning
Window	The name of the window which contains the grid control to be used. Type STR.
Branch	The branch of the window (if any). Using a "*" means the current branch of the program. Type STR.
Identity	The identity of the grid control within the specified window. Type STR.

Syntax 1

IntVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity*);

The return type is INTEGER.

Execution

Sub-mode	Action
GETCOUNT	Return the number of trended variables in the list. Return: -1 if the function fails, otherwise the number of trends in the list.
CLEARTRENDS	Remove all trended variables in the list. The return type is INTEGER. Return: -1 if the function fails, otherwise the number of trended variables removed.
RESTOREDEFAULT	Reload the list of trended variables and thresholds as originally configured in the grid control. Return: The quantity of restored trend variables if successful, else -1.
CANCEL	Cancel the current historical request. Return: 1 if successful, else -1.
EXPORT	Starts the Export Wizard from the specified grid control. Return: 1 if successful, -1 if Grid Control not in historical mode, else 0.

Syntax 2

IntVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity, VarName* [*, Index*]);

The return type is INTEGER.

Argument	Meaning
VarName	The name of a variable. Type STR.
Index	The index in the list. 1 = start of the list. Type INTEGER.

Execution

Sub-mode	Action
INSERTTREND	Insert a new trended variable to the list of variables at the Index position. If the Index parameter is omitted the variable is inserted at the start of the list. Return: -1 if the function fails, otherwise the position of the variable in the list.

Syntax 3

IntVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity*[*, Index*]);

The return type is INTEGER.

Argument	Meaning
Index	The index in the list. 1 = start of the list. Type INTEGER.

Execution

Sub-mode	Action
REMOVETREND	Remove the trended variable, at the Index position, from the list of variables. If the Index parameter is omitted the variable at the end of the list is removed. Return: -1 if the function fails, otherwise the position of the variable in the list.

Syntax 4

IntVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity*, *Index*, *VarName*[, *Thresholds*]);

The return type is INTEGER.

Argument	Meaning
VarName	The name of a variable. Type STR.
Index	The index in the list. 1 = start of the list. Type INTEGER.
Thresholds	A flag. Type INTEGER.

Execution

Sub-mode	Action
SETTREND	Change the trended variable at the Index position. The Threshold flag indicates if the threshold system should be kept (1) or removed (0). The default is 0. Return: 1 if successful, else -1.

Syntax 5

StrVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity, Index*);

The return type is STR.

Argument	Meaning
Index	The index in the list. 1 = start of the list. Type INTEGER.

Execution

Sub-mode	Action
GETTRENDBYINDEX	Return the name of the trended variable at position of the Index parameter. Return: The name of the variable if successful, else an empty string.

Syntax 6

IntVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity, VarName*);

The return type is INTEGER.

Argument	Meaning
VarName	The name of a variable. Type STR.

Execution

Sub-mode	Action
GETINDEX	Return the index of the named trended variable. Return: The index of the variable name of the variable if successful, else an empty string.

Syntax 7

IntVal = SELECTOR("HISTORICAL", *Sub-mode*, *Window*, *Branch*, *Identity, StatusVariable [,StartTime, EndTime[, SamplingRateInterval[, SamplingRateValue[, SynchroSecondValue, SynchroMinuteValue, SynchroHourValue, SynchroDayValue]]]]*);

The return type is INTEGER.

Argument	Meaning
StatusVariable	The name of a register variable. Set to 1 while the request is running, else 0. Type STR.
StartTime	Equivalent to the value provided by the Start time variable in the Grid Control configuration dialog, Historical Period tab. Type DOUBLE.
EndTime	Equivalent to the value provided by the End time variable in the grid control configuration, Historical Period tab.Type DOUBLE.
SamplingRateInterval	Equivalent to the sampling rate units in the Grid Control configuration dialog, Historical Sampling tab. Type INTEGER. 0 = milliseconds 1 = seconds 2 = minutes 3 = hours 4 = days 5 = weeks 6 = months
SamplingRateValue	Equivalent to the Sampling Rate property in the Grid Control configuration dialog, Historical Sampling tab. Type INTEGER.
SynchroSecondValue	Equivalent to the Second property in the Grid Control configuration dialog, Historical Sampling tab. Type INTEGER.
SynchroMinuteValue	Equivalent to the Minute property in the Grid Control configuration dialog, Historical Sampling tab. Type INTEGER.
SynchroHourValue	Equivalent to the Hour property in the Grid Control configuration dialog, Historical Sampling tab. Type INTEGER.
SynchroDayValue	Equivalent to the Day property in the Grid Control configuration dialog, Historical Sampling tab. Type INTEGER.

Execution

Sub-mode	Action
REQUEST	Generate a new historical request for the gird using the supplied parameters. See the grid control help for a full explanation of the operation of the parameters.

Return: 1 if successful, -1 if Grid Control not in historical mode, else 0.

Example

For an example, select the Example link above.

SENDLIST

See Also Further information

Send a list of variables to an item of equipment.

WebVue support - Yes.

This instruction is used in conjunction with the keyword SET.

Mode	Mnemonic	Syntax
0	BLOC	1
1	MULTIPLE	1

Syntax 1

IntVal = SENDLIST(*Mode*[, *VarName*[, *OPCMode*]]);

The return type is INTEGER.

Argument	Meaning
VarName	The name of a database register variable with the control attribute set. It cannot be used elsewhere in the program. Type STR. It will contain the status of the instruction: 0 Sending 1 Accepted 2 Not accepted
OPCMode	The sending mode for when the source of the variable's value is OPC. 0 Optimised serialisation (default). 1 Full serialisation. 2 No optimisation. 3 Full optimisation.

Execution

See the topic SENDLIST send mode for an explanation of the execution.

Return (for both modes): 1 if successful, else 0.

To be set, variables must have the Command attribute enabled.

Examples

```
Dim strVar as STR;

strVar = "BR1.VAL";
'note NOT "@BR1.VAL"
SET(strVar,12);
SENDIST("BLOC");
```

```
SUB Main()
i1 = SET ("Register1",600,);
'address AP of Register 1 : S1
i1 = SET ("Register2",700,);
'address AP of Register 1 : S2
i1 = SET ("Register3",500,);
'address AP of Register 1 : S5
i1 = SET ("Register4",400,);
'address AP of Register 1 : S6
```

```
i1 = SET ("Register5",300,);
'address AP of Register 1 : S7
SENDLIST ("BLOC"); 'Send a frame, writing words S1 to S7
' (S3 and S4 will be written with their current values)
END SUB
```

SEQ_BUFFER

See Also Example

Manipulate lines of text in a memory buffer.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	CLEAR	1
2	PUT_LINE	2
3	BEGIN	3
4	END	3
5	NEXTFIELD	4, 7
6	PREVFIELD	4, 7
7	SEEKFIELD	5
8	REPLACEFIELD	6
8	INSERTFIELD	6
9	CRLFTOCR	3, 8
10	CRTOCRLF	3, 8
11	LEN	3
12	ASCIITOANSI	3
13	ANSITOASCII	3

Syntax 1

IntVal = SEQ_BUFFER(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a buffer that has been allocated using an ALLOC_BUFFER instruction. Type LONG.

Execution

Mode	Mnemonic	Action
1	CLEAR	The contents of the buffer are cleared. Return: 1 if successful, else 0.

Syntax 2

IntVal = SEQ_BUFFER(*Mode*, *Handle*, *Text*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a buffer that has been allocated using an ALLOC_BUFFER instruction. Type LONG.
Text	The text line to be placed in the buffer. Type STR.

Execution

Mode	Mnemonic	Action
2	PUT_LINE	The specified text is written into the buffer at the current pointer position.

SEQ_BUFFER may be used repeatedly to append several lines of text to the buffer.

Return: 1 if successful, -1 in case of buffer overflow, else 0.

Syntax 3

IntVal = SEQ_BUFFER(*Mode*, *Handle*);

The return type is INTEGER except for mode 11 where it is LONG.

Argument	Meaning
Handle	The handle of a buffer that has been allocated using an ALLOC_BUFFER instruction. Type LONG.

Execution

Mode	Mnemonic	Action
3	BEGIN	Position the pointer at the start of the buffer.
4	END	Position the pointer at the end of the buffer.
9	CRLFTOCR	Convert any occurrences of "Carriage Return – Line Feed" in the buffer to "Carriage Return".
10	CRTOCRLF	Convert any occurrences of "Carriage Return" in the buffer with "Carriage Return – Line Feed".
11	LEN	Return the size of the buffer.
12	ASCIITOANSI	Converts a buffer of ASCII characters to ANSI characters.
13	ANSITOASCII	Converts a buffer of ANSI characters to ASCII characters.
		Return: 1 if successful, else 0 (except Mode 11 – see above).

CRTOCRLF requires enough space in the buffer for the extra characters created when all instances of CR are replaced with CRLF.

For example, it will not work on a buffer created directly by FILETOBUF as this creates a buffer with the exact number of characters for the original file.

Syntax 4

IntVal = SEQ_BUFFER(*Mode*, *Handle*, *Separator*, *ResultHandle*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a buffer containing a number of delimited values. Type LONG.
Separator	The character used to separate values in the buffer. Type STR.
ResultHandle	The handle of a buffer into which the result of the instruction is placed.

Execution

Mode	Mnemonic	Action
5	NEXTFIELD	Retrieve the value at the current pointer position and place the result in the buffer indicated by ResultHandle.
6	PREVFIELD	Retrieve the value preceding the current pointer position and place the result in the buffer indicated by ResultHandle.
		Return: 1 if successful, else 0.

Syntax 5

IntVal = SEQ_BUFFER(*Mode*, *IncPosition*, *Handle*, *Separator*);

The return type is INTEGER.

Argument	Meaning
IncPosition	The number of values by which the pointer is to be moved (negative or positive, by counting fields from current position of the pointer). Type INTEGER.
Handle	The handle of a buffer containing a number of delimited values. Type LONG.
Separator	The character used to separate values in the buffer. Type STR.

Execution

Mode	Mnemonic	Action
7	SEEKFIELD	Move the pointer to a new position using the old position plus the value contained in *IncPosition*. Return: 1 if successful, else 0.

Syntax 6

IntVal = SEQ_BUFFER(*Mode*, *IncPosition*, *Handle*, *Separator*, *StringOrBuff*);

The return type is INTEGER.

Argument	Meaning
IncPosition	A count of fields to determine the position of the field in the buffer to be replaced and at which a new string is to be inserted. (Must be greater than 0.) Type INTEGER.
Handle	The handle of a buffer containing a number of delimited values. Type LONG.
Separator	The character used to separate values in the buffer. Type STR.
StringOrBuff	Either a string, or the handle of a buffer containing a string. Type STR or Long.

Return type : INTEGER.

Execution

Mode	Mnemonic	Action
8	REPLACEFIELD or INSERTFIELD	Replace the field or insert a new string in the buffer at the field position calculated using the current position plus the value contained in *IncPosition*, as a count of fields. Return: 1 if successful, else 0.

The verb names REPLACEFIELD and INSERTFIELD are synonymous. The instruction always replaces one field. To retain an existing field, include it in the Chain argument before or after the new field value(s). This has the effect of insertion.

If the length of the string of the new field(s) to be inserted is more than that of the string to be replaced, you must ensure that the buffer is large enough or else the action will fail. Be careful when using this verb with the verbs FILETOBUF and BUFTOFILE.

For instance to insert a field nField_i before an existing field oField_i, use:

SEQ_BUFFER("INSERTFIELD", i, Handle, ",", "nField_i, oField_i"

Syntax 7

RetVal = SEQ_BUFFER(*Mode*, *Handle*, *Separator*, *SubMode*);

The return type is INTEGER.

Argument	Meaning
Handle	The handle of a buffer containing a number of delimited values. Type LONG.

Separator	The character used to separate values in the buffer. Type STR.
SubMode	The string determining the type of value to be read and the return type: STR - String. INT - Integer. DOUBLE - Double. FLOAT - Float.

Execution

Mode	Mnemonic	Action
5	NEXTFIELD	Return the value at the current pointer position.
6	PREVFIELD	Return the value preceding the current pointer position.

If the types of value in the buffer are mixed you must return the value as a type STR and then interpret the result

Syntax 8

LongVal = SEQ_BUFFER(*Mode*, *Handle*, *"NEW_BUFFER"*);

The return type is LONG.

Argument	Meaning
Handle	The handle of a buffer that has been allocated using an ALLOC_BUFFER instruction. Type LONG.

Execution

Mode	Mnemonic	Action
9	CRLFTOCR	Convert any occurrences of "Carriage Return - Line Feed" in the buffer to "Carriage Return". The result is placed in a new buffer the handle of which is returned by the instruction.
10	CRTOCRLF	Convert any occurrences of "Carriage Return" in the buffer to "Carriage Return - Line Feed". The result is placed in a new buffer the handle of which is returned by the instruction. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

SESSION

See Also Example

For session management.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	TRACE	1
2	GETCURRENTID	2

Syntax 1

IntVal = SESSION(*Mode*, *SubMode*);

The return type is INTEGER.

Argument	Meaning
SubMode	Either "ON" or "OFF". Type STR.

Execution

Mode	Mnemonic	Action
1	TRACE	Activate or deactivate trace messages associated with session management. Return: always 1.

Syntax 2

IntVal = SESSION(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	GETCURRENTID	Return the session Id in which the program is running. Return: the session Id.

SET

See Also

Construction of a list of variables and values.

WebVue support - Yes.

This is used in conjunction with the keyword SENDLIST.

Syntax

IntVal = SET(*Variable*, *Value*);

The return type is INTEGER.

Argument	Meaning
Variable	The name of the variable that is to be assigned a value. Type STR.
Value	The value that is to be assigned. This will be of the same type as the variable.

Execution

The value is assigned to the variable in memory. It may then be sent to the equipment using SENDLIST.

Return : 1 if successful, else 0.

To be set, variables must have the Command attribute enabled.

Conflict processing does not work with recipes.

Examples

Please refer to the examples for SENDLIST.

SGET_BUFFER

See Also

Read a SINGLE located in a memory area.

WebVue support - Yes.

Syntax

SinVal = SGET_BUFFER(*Handle*, *Offset*);

The return type is SINGLE.

Argument	Meaning
Handle	The location of the memory buffer as returned by ALLOC_BUFFER. Type LONG.
Offset	The offset in bytes at which the integer is to be found. Any numeric type.

Execution

Return: A SINGLE located in a memory buffer allocated by ALLOC_BUFFER.

Example

```
SUB main()
'Declare variables
DIM lngBuffer1 as Long;
DIM strLine as Str;
DIM sngResult as Single;
DIM intLengh as Integer;

'Create a buffer
lngBuffer1 = ALLOC_BUFFER(110);
strline = "123.34;string_here;345;123456789;,t,\n";

'Put the string srtline into the buffer
PUT_BUFFER(lngBuffer1, 0, strline);
sngResult = SGET_BUFFER(lngBuffer1,0);
PRINT("Result: ",sngResult);

'Display "Result: 5.10584E+256"
FREE_BUFFER(lngBuffer1); 'release the memory area
END SUB
```

SIN

See Also

Sine function.

WebVue support - Yes.

Syntax

Dblval = SIN(*Angle*);

The return type is DOUBLE.

Argument	Meaning
Angle	The angle in degrees of which the sine is to be taken. Any numeric type.

Execution

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
'Variables
'@ARC - type Register
'@ANGLE - type Register

SUB Main()
'Declare variables
DIM dblangle as double;
DIM dblarc as double;

dblangle = 45;
dblarc = SIN (dblangle);
print("SIN(",dblangle,") = ",dblarc);
END SUB

SUB proc2()
'SIN() using a variable: @ARC.
'Use of TOS to convert DOUBLE to SINGLE (variable type Register)
@ARC= TOS( SIN (@ANGLE) ) ;
END SUB
```

SMS

See Also

Manages sending text messages.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	SEND	1

Syntax 1

SMS(*Mode*, *SmsProfile*, *PhoneNumbers*, *Subject*, *Message* [, *Format*]);

The return type is INTEGER.

Argument	Meaning
SmsProfile	The name of an SMS profile as configured in Actions.Messages.SMS Profiles. Type STR.
PhoneNumbers	A list of the phone numbers of the recipients, using semicolon (;) as the separator between the phone numbers. Type STR.
Subject	The subject of the message. Type STR.
Message	The body of the message. Type STR.
Format	The message format. (Optional) Type INTEGER. 0: Auto (default value) 1: Text 2: PDU 7-bit 3: Unicode. For an explanation of the *Format* argument and the size limits for messages in each format, see the topic Actions.Event actions.Configuring an event to send a message.Configuring the SMS Options

If executed on WebVue the SMS is sent from the WebVue back end.

Execution

Mode	Mnemonic	Action
1	SEND	Set up a message for sending via SMS. Return: Return: 0 if successful, else 1. The sending status is recorded in the system variable.

Example

```
sub main ()
end sub

sub SendSms ()
SMS("SEND", "SMSPROFILE1","0033687867361","Scada Basic test", "Message sent by SV");
end sub

sub AddSms ()
EVENT ("ADDSMS", "SBSEND","", "", "BitSmsSB", 1, "MSGTEMPLATE1");
end sub

sub DelSms ()
EVENT("DELSMS","SBSEND");
end sub
```

SNMP

See Also

Manages communication via the SNMP protocol.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	START	1
2	STOP	1
3	REFRESH	2, 3
4	CONFIG	4, 5

Syntax 1

IntVal = SNMP(Mode, CommObject[, Resultvar]);

The return type is integer.

Argument	Meaning
CommObject	Either the network name (Example "Network01") , or the network and device name separated by a full stop (Example "Network01.Device01"). Type STR.
ResultVar	The name of a register variable in which the result of the instruction will be placed. Type Str. -1 - Network name not found. -2 - Device name not found. 1 - Action pending. 2 - Action completed OK. 3 - Action completed with error.

Execution

Mode	Mnemonic	Action
1	START	Start the network or device.
2	STOP	Stop the network or device.

Return for both modes: 0 = correct format, -1 = unknown mode, -2 = CommObject format incorrect (empty or too many fields)

Syntax 2

IntVal = SNMP(Mode, CommObject, SubMode[, ResultVar]);

Argument	Meaning
CommObject	The network and device name seperated by a full stop. Example "Network01.Device01". Type STR.
SubMode	See below. Type STR.
ResultVar	The name of a register variable in which the result of the instruction will be placed. Type Str. -1 - Network name not found. -2 - Device name not found. -3 - The polling group name not found. 1 - Action pending. 2 - Action completed OK. 3 - Action completed with error.

Execution

Mode	Mnemonic	Action
3	REFRESH	Sub mode "ALL". Refresh the value of all variables mapped to the device.
3	REFRESH	Sub mode "SYNCHRO". Refresh only the variables updated by (Trap and ad-hoc synchronization).

Return for both modes: 0 correct format, -1 unknown mode, -2 CommObject format incorrect (either empty or more than two fields), -4 unknown sub-mode.

Syntax 3

IntVal = SNMP(Mode, CommObject, SubMode, PollingGroup[, ResultVar]);

Argument	Meaning
CommObject	The network and device name separated by a full stop. (Example "Network01.Device01"). Type STR.
SubMode	See below. Type STR.
PollingGroup	The name of a polling group. Type STR.
ResultVar	The name of a register variable in which the result of the instruction will be placed. Type STR. -1 - Network name not found. -2 - Device name not found. -3 - The polling group name not found. 1 - Action pending. 2 - Action completed OK. 3 - Action completed with error.

Execution

Mode	Mnemonic	Action
3	REFRESH	Sub mode "POLLINGGROUP". Refresh the value of all variables of the specified device and attached to the specified polling group.

Return: 0 correct format, -1 unknown mode, -2 CommObject format incorrect (either empty or more than two fields), -3 PollingMode format incorrect (empty), -4 unknown sub-mode.

Syntax 4

IntVal = SNMP(Mode, CommObject, "HOSTNAME", HostName[, ResultVar]);

Argument	Meaning
CommObject	The network and device name separated by a full stop. (Example "Network01.Device01"). Type STR.
HostName	The new name for the device. Type STR.
ResultVar	The name of a register variable in which the result of the instruction will be placed. Type Str. -1 - Network name not found. -2 - Device name not found. 1 - Action completed OK.

Execution

Mode	Mnemonic	Action
4	CONFIG	Change the device name.

Return: 0 correct format, -1 unknown mode, -2 Device name not specified, -3 Unknown sub-mode, -5 Empty IP address.

Syntax 5

IntVal = SNMP(Mode, CommObject, "IPADDRESS", IPaddress[, ResultVar]);

Argument	Meaning
CommObject	The network and device name separated by a full stop. (Example "Network01.Device01"). Type STR.
IPaddress	The new IP address for the device in the format xxx.xxx.xxx.xxx. Type STR.
ResultVar	The name of a register variable in which the result of the instruction will be placed. Type Str.

-1 - Network name not found.
-2 - Device name not found.
1 - Action completed OK.

Execution

Mode	Mnemonic	Action
4	CONFIG	Change the IP address.

Return: 0 correct format, -1 unknown mode, -2 Device name not specified, -3 Unknown sub-mode, -5 Empty IP address.

SPACE

See Also

Return a string composed of the specified number of spaces.

WebVue support - Yes.

Syntax

StrVal = SPACE(*N*);

The return type is STR.

Argument	Meaning
N	The number of spaces to return. Any numeric type.

Execution

The returned string is limited to 2,047 characters.

Example

```
SUB Main()
'Declare return code
DIM intCharCode as integer;
DIM strChaine as Str;

strChaine = SPACE(10);
PRINT("Contents of string: ",strChaine);
END SUB
```

SQL_COMMAND

See Also Example

Sends Sql commands using a pre-configured Sql connection.

WebVue support - Yes.

As of version 12.0, the use of pre-configured Sql connections along with the SQL_CONNECTION and SQL_COMMAND is preferred to using the verb SVSQL (based on ODBC).
Please refer to the Application Explorer book for more information about Sql connection configuration.

Some of the modes are asynchronous. The application developer must ensure that the value of status variables are monitored and take adequate actions when they change.

This verb can be executed either locally on the producer station of the Sql connection or from any client station of the producer. In the latter case, commands and returned data are routed transparently through the multi-station messaging system. Multiple connections to the same data source are only allowed if using different Sql Connections. Multiple connections using the same Sql Connection will be refused.

A connection and the data associated to it are managed in the context of user sessions. A connection should only be handled in the context of the session that had originally initiated it; and all related objects must be properly released/closed before the session is ended. For example it is not possible to initiate a connection at start up and have requests executed in any user context.

Mode	Mnemonic	Syntax
1	INITCONNECTION	1
2	CLOSECONNECTION	2
3	GETCONNECTIONLIST	3
4	GETCONNECTIONINFO	4
5	EXECREADER	5
6	EXECSCALAR	6
7	EXECNONQUERY	7
8	GETCOMMANDLIST	8
9	GETCOMMANDINFO	9
10	READBUFFER	9
11	SEEKBUFFER	10
12	READBUFFERLINE	11
13	READBUFFERCELL	12
14	BUFFERLINECOUNT	13
15	BUFFERFIELDCOUNT	13
16	BUFFERFIELDNAME	14
17	BUFFERFIELDTYPE	14
18	BUFFERSIZE	13
19	READSCALARVALUE	14
20	READAFFECTEDROW	13
21	CANCEL	13
22	DISPOSE	13

Arguments common to more than one mode

Argument	Meaning
SqlConnectionName	The name of the Sql Connection as configured in General.Data connections in the Application Explorer. Type STR
BufferHandle	The handle of a buffer as returned by ALLOC_BUFFER. Type LONG. All modes require the buffer to be allocated prior to calling the instruction.
SqlCmd	The Sql command to send. Type STR
StatusVariable	The name of the register variable used to monitor the status of an asynchronous operation. Succeeded = 0 Running = 1 Failed = 2 Canceled = 3
ErrortextVariable	The name of a text variable used to return additional information when the status variable value is set to failed (2).
SubstituteNullValueWithEmptyString	Substitute a null value with an empty string in returned data. 0 - a null value is represented by <NULL> 1 - a null value is represented by an empty string
LineNumber	A line position in the result buffer (First line is 0). Type LONG.
CmdId	The command Id as returned by mode EXECREADER, EXECSCALAR or EXECNONQUERY.

Syntax 1

IntVal = SQL_COMMAND(*Mode, SqlConnectionName [, MaxSimultaneousCommand]*);

Argument	Meaning
MaxSimultaneousCommand	The maximum number of simultaneous asynchronous commands. The primary use is to limit the memory used by the Supervisor for the reply buffer. The default value is 1. The maximum value is 100. If the value is incorrect, the default value is used instead.

Execution

Mode	Mnemonic	Action
1	INITCONNECTION	Initialize a connection using the specified Sql connection. The runtime object must be started prior to using this mode for initializing the actual Sql connection to the data source. Return: 0 if successful, else a negative number indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ SQLCON_DOES_NOT_EXIST CONNECTION_ALREADY_INIT MAXIMUM_CONNECTION_REACH CONNECTION_INIT_FAILED See the table below for integer return values.

Multiple connections to the same data source are only allowed if using different Sql Connections. Multiple connections using the same Sql Connection will be refused.

Syntax 2

IntVal = SQL_COMMAND(*Mode[, SqlConnectionName]*);

Execution

Mode	Mnemonic	Action
2	CLOSECONNECTION	Close the connection corresponding to the specified Sql connection. If the SqlConnectionName argument is omitted, all open connections will be closed. Return: 0 if successful, else a negative number indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ CONNECTION_ALREADY_CLOSE_OR_NOT_INIT CONNECTION_CLOSE_FAILED SQLCON_DOES_NOT_EXIST See the table below for integer return values.

Syntax 3

IntVal = SQL_COMMAND(*Mode, BufferHandle*);

Execution

Mode	Mnemonic	Action
3	GETCONNECTIONLIST	Get the name of all open Sql connections and return them in a buffer. Connection names are delimited with the comma character. Return: 0 if successful, else a negative number indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE RESULT_BUFFER_EMPTY RESULT_BUFFER_TOOSMALL See the table below for integer return values.

Syntax 4

IntVal = SQL_COMMAND(*Mode, C*onnection*P*aram*, SqlConnectionName, BufferHandle*);

Argument	Meaning
*C*onnection*P*aram	The sub mode. 1 or "NB_COMMAND" Get the number of active commands, e.g. ones that are not yet disposed of using the DISPOSE mode. 2 or "MAX_COMMAND" Get the maximum number of commands allowed for this connection.

Execution

Mode	Mnemonic	Action
4	GETCONNECTIONINFO	Return information about the number of commands. Return: 0 if successful, else a negative number indicating one of the following errors. CONNECTION_PARAM_INVALID SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE RESULT_BUFFER_TOOSMALL CONNECTION_ALREADY_CLOSE_OR_NOT_INIT

See the table below for integer return values.

Syntax 5

IntVal = SQL_COMMAND(*Mode, SqlConnectionName, SqlCmd, StatusVariable, ErrorTextVariable, LineSeparator, FieldSeparator [, SubstituteNullValueWithEmptyString]*);

Argument	Meaning
LineSeparator	The line separator character to be used in the buffer. Type STR
FieldSeparator	The field separator character to be used in the buffer. Type STR

Execution

Mode	Mnemonic	Action
5	EXECREADER	Execute a Sql query designed to return a set of records and store the result in a buffer. The contents are separated by the line and field separator arguments. Return: Either a positive value representing the command Id, or a negative value indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ SQLCMD_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_PARAM_NOT_STR_OR_NOT_READ ERRORTEXTVARIABLE_PARAM_NOT_STR_OR_NOT_READ LINESEPARATOR_PARAM_NOT_STR_OR_NOT_READ_OR_TOOLONG FIELDSEPARATOR_PARAM_NOT_STR_OR_NOT_READ_OR_TOOLONG STATUSVARIABLE_NOT_REGISTER_OR_DOES_NOT_EXIST ERRORTEXTVARIABLE_NOT_TEXT_OR_DOES_NOT_EXIST SQLCON_DOES_NOT_EXIST MAXIMUM_COMMAND_REACH SEND_COMMAND_FAILED See the table below for integer return values.

This mode is asynchronous. The program must monitor the value of the status variable and use either modes READBUFFER or READBUFFERLINE to get the buffer result.

Syntax 6

IntVal = SQL_COMMAND(*Mode, SqlConnectionName, SqlCmd, StatusVariable, ErrorTextVariable [, SubstituteNullValueWithEmptyString]*);

Execution

Mode	Mnemonic	Action
6	EXECSCALAR	Execute a Sql query designed to return a scalar value and store the scalar result in a buffer. The result is a single value. Return: Either a positive value representing the command Id, or a negative value indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ SQLCMD_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_PARAM_NOT_STR_OR_NOT_READ ERRORTEXTVARIABLE_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_NOT_REGISTER_OR_DOES_NOT_EXIST ERRORTEXTVARIABLE_NOT_TEXT_OR_DOES_NOT_EXIST SQLCON_DOES_NOT_EXIST MAXIMUM_COMMAND_REACH SEND_COMMAND_FAILED

See the table below for integer return values.

This mode is asynchronous. The program must monitor the value of the status variable and use mode READSCALAR to get the value.

Syntax 7

IntVal = SQL_COMMAND(*Mode, SqlConnectionName, SqlCmd, StatusVariable, ErrorTextVariable*);

Execution

Mode	Mnemonic	Action
7	EXECNONQUERY	Execute a Sql request that does not query data from the database, but causes a certain operation to be executed on the data source side (as defined by the SqlCmd argument). Such requests usually return the result of the operation execution. Return: Either a positive value representing the command Id, or a negative value indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ SQLCMD_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_PARAM_NOT_STR_OR_NOT_READ ERRORTEXTVARIABLE_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_NOT_REGISTER_OR_DOES_NOT_EXIST ERRORTEXTVARIABLE_NOT_TEXT_OR_DOES_NOT_EXIST SQLCON_DOES_NOT_EXIST MAXIMUM_COMMAND_REACH SEND_COMMAND_FAILED See the table below for integer return values.

This mode is asynchronous. The program must monitor the value of the status variable and use the mode READAFFECTEDROW to get the number of rows affected by the command.

Syntax 8

IntVal = SQL_COMMAND(*Mode, SqlConnectionName, BufferHandle*);

Execution

Mode	Mnemonic	Action
8	GETCOMMANDLIST	Retrieve the command list associated with a connection. The command Ids are returned in the buffer separated by commas. Return: Either a positive value representing the number of active commands, or a negative value indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE CONNECTION_ALREADY_CLOSE_OR_NOT_INITRESULT_BUFFER_EMPTY RESULT_BUFFER_TOOSMALL See the table below for integer return values.

Syntax 9

IntVal = SQL_COMMAND(*Mode, C*ommand*P*aram*, CmdId, BufferHandle*);

Argument	Meaning

Command*P*aram	The sub mode. 1 or "TYPE". 2 or "STATUS" 3 or "ERROR_TEXT" 4 or "SQL_CMD" 5 or "STATUS_VARIABLE" 6 or "ERROR_TEXT_VARIABLE"

Execution

Mode	Mnemonic	Action
9	GETCOMMANDINFO	Retrieve information, in a buffer, about a command specified by its CmdId. The information returned depends on the sub mode. TYPE - Retrieves the type of command that has been sent. 0 = EXECREADER 1 = EXECSCALAR 2 = EXECNONQUERY STATUS - Retrieves the status of the request. 0 = Succeeded 1 = Running 2 = Failed 3 =Canceled ERROR_TEXT - Retrieves the error text corresponding to the command. Empty unless the command has failed. SQL_CMD - The Sql request that has been sent. STATUS_VARIABLE - The name of the status variable. ERROR_TEXT_VARIABLE - The name of the error variable. Return: 0 if successful, else a negative number indicating one of the following errors. COMMAND_PARAM_INVALID BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE CMDID_DOES_NOT_EXIST COMMAND_PARAM_INVALID RESULT_BUFFER_TOOSMALL See the table below for integer return values.
10	READBUFFER	Used to read the result of EXEC_READER query once the status variable value is set to 0. It may be necessary to run this function one or more times depending on the size of the buffer and the quantity of data returned. Return: Either a positive value representing the number of lines read, or a negative value indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_EMPTY RESULT_BUFFER_TOOSMALL See the table below for integer return values.

Syntax 10

IntVal = SQL_COMMAND(Mode, CmdId, LineNumber);

Execution

Mode	Mnemonic	Action
11	SEEKBUFFER	Used to seek a line position in the buffer result, once the status variable is set to 0. First line is LineNumber 0.

Return: 0 if successful, else a negative number indicating one of the following errors.
LINENUMBER_PARAM_INVALID
CMDID_DOES_NOT_EXIST
CMD_TYPE_NOT_VALID
CMD_NOT_FINISHED
RESULT_BUFFER_EMPTY

See the table below for integer return values.

Syntax 11

IntVal = SQL_COMMAND(*Mode, CmdId, BufferHandle, LineNumber*);

Execution

Mode	Mnemonic	Action
12	READBUFFERLINE	Read a specific line in the buffer, produced as a result of the use of an EXECREADER function, once the status variable is set to 0. First line is LineNumber 0. Return: Either a positive value representing the number of lines read, or a negative value indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE LINENUMBER_PARAM_INVALID CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_EMPTY RESULT_BUFFER_TOOSMALL See the table below for integer return values.

Syntax 12

IntVal = SQL_COMMAND(*Mode, CmdId, BufferHandle, LineNumber, FieldNumber*)

Argument	Meaning
FieldNumber	A field position in a line of the result buffer (First field is FieldNumber 0). Type LONG.

Execution

Mode	Mnemonic	Action
13	READBUFFERCELL	Read a specific field in a specific line of the buffer, produced as a result of the use of the EXECREADER function, once the status variable is set to 0. First line is LineNumber 0. Return: Either a positive value representing the number of lines read, or a negative value indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE LINENUMBER_PARAM_INVALID FIELDNUMBER_PARAM_INVALID CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_EMPTY RESULT_BUFFER_TOOSMALL See the table below for integer return values.

Syntax 13

IntVal = SQL_COMMAND(Mode, CmdId);

Execution

Mode	Mnemonic	Action
14	BUFFERLINECOUNT	Get the number of lines in the buffer, produced as a result of the use of the EXECREADER function, once the status variable is set to 0. Return: Either zero or a positive value representing the number of lines, or a negative value indicating one of the following errors. CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_REACH_NBLINE_MAX (Maximum is 2147483647) See the table below for integer return values.
15	BUFFERFIELDCOUNT	Get the number of fields in the buffer, produced as a result of the use of the EXECREADER function, once the status variable is set to 0. Return: Either zero or a positive value representing the number of fields, or a negative value indicating one of the following errors. CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED See the table below for integer return values.
18	BUFFERSIZE	Get the size of the buffer, produced as a result of the use of the EXECREADER function, once the status variable is set to 0. Return: A positive value representing the number of bytes in the buffer, or a negative value indicating one of the following errors. CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_REACH_SIZE_MAX (Maximum 2147483647) See the table below for integer return values.
20	READAFFECTEDROW	Read the result of a EXECNONQUERY function, once the status variable is set to 0. Return: A positive value representing the affected row count, or a negative value indicating one of the following errors. CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED See the table below for integer return values.
21	CANCEL	Cancel the request corresponding to the CmdId. Return: 0 if successful, else a negative number indicating one of the following errors. CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID See the table below for integer return values.
22	DISPOSE	Release the context of a query including the internal buffer, corresponding to the CmdId. Must be called afterrequest result is processed to free up resources. The buffer result is no longer usable after the execution of the DISPOSE mode. Return: 0 if successful, else a negative number indicating one of the following errors.

CMDID_DOES_NOT_EXIST
CMD_TYPE_NOT_VALID
CMD_NOT_REMOVED

See the table below for integer return values.

Failing to call the mode DISPOSE once you have completed a command and the processing of its result prevents from freeing up memory and may lead to an uncontrolled usage of computer resources.

Syntax 14

IntVal = SQL_COMMAND(Mode, CmdId, BufferHandle);

Execution

Mode	Mnemonic	Action
16	BUFFERFIELDNAME	Get the header names, produced as a result of the use of the EXECREADER function, once the status variable is set to 0. Return: 0 if successful, else a negative number indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_TOOSMALL See the table below for integer return values.
17	BUFFERFIELDTYPE	Get the datatypes corresponding to the fields, produced as a result of the use of the EXECREADER function, once the status variable is set to 0. Return: 0 if successful, else a negative number indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_TOOSMALL See the table below for integer return values.
19	READSCALARVALUE	Read the scalar value, produced by an EXECSCALAR function, once the status variable is set to 0. Return: 0 if successful, else a negative number indicating one of the following errors. BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE CMDID_DOES_NOT_EXIST CMD_TYPE_NOT_VALID CMD_NOT_FINISHED RESULT_BUFFER_TOOSMALL See the table below for integer return values.

List of possible return values and meanings

Return value	Enum	Description
0	OPERATION_SUCCEEDED	Operation completed s
-1	SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ	The parameter *SqlCon* read
-2	SQLCMD_PARAM_NOT_STR_OR_NOT_READ	The parameter *SqlCmd*

-3	STATUSVARIABLE_PARAM_NOT_STR_OR_NOT_READ	The parameter *StatusV*
-4	LINESEPARATOR_PARAM_NOT_STR_OR_NOT_READ_OR_TOOLONG	The parameter *LineSep* too long
-5	FIELDSEPARATOR_PARAM_NOT_STR_OR_NOT_READ_OR_TOOLONG	The parameter *FieldSe* too long
-6	BUFFERHANDLE_PARAM_INVALID_OR_NULL_OR_NULLSIZE	The parameter *BufferH* allocation using ALLOC properly
-7	LINENUMBER_PARAM_INVALID	The parameter *LineNu*
-8	FIELDNUMBER_PARAM_INVALID	The parameter *FieldNu*
-9	ERRORTEXTVARIABLE_PARAM_NOT_STR_OR_NOT_READ	The parameter *ErrorTe*
-10	CONNECTION_PARAM_INVALID	The parameter *Connec*
-11	COMMAND_PARAM_INVALID	The parameter *Comma*
-12	SQLCON_DOES_NOT_EXIST	The specified Sql conn
-13	STATUSVARIABLE_NOT_REGISTER_OR_DOES_NOT_EXIST	The status variable pas does not exist
-14	ERRORTEXTVARIABLE_NOT_TEXT_OR_DOES_NOT_EXIST	The error text variable does not exist
-15	CONNECTION_ALREADY_INIT	The connection is alrea
-16	CONNECTION_INIT_FAILED	The connection could n
-17	CONNECTION_ALREADY_CLOSE_OR_NOT_INIT	The connection is alrea
-18	CONNECTION_CLOSE_FAILED	Failed to close the con
-19	SEND_COMMAND_FAILED	Failed to send the Sql service is stopped or if execute the command
-20	MAXIMUM_COMMAND_REACH	The maximum number
-21	MAXIMUM_CONNECTION_REACH	The maximum number
-22	CMDID_DOES_NOT_EXIST	The command *CmdId* p
-23	CMD_TYPE_NOT_VALID	The command type is i the READ_BUFFER with non-query
-24	CMD_NOT_FINISHED	The command is not ye
-25	CMD_NOT_REMOVED	The command could no
-26	RESULT_BUFFER_EMPTY	The result buffer is em
-27	RESULT_BUFFER_TOOSMALL	The result buffer is too
-28	RESULT_BUFFER_REACH_NBLINE_MAX	Failure to return inform because it exceeds the
-29	RESULT_BUFFER_REACH_SIZE_MAX	Failure to return inform return value range

SQL_CONNECTION

See Also Example

Manage the Supervisor's runtime objects used to handle Sql connections.

WebVue support - Yes.

As of version 12.0, the use of pre-configured Sql connections along with the SQL_CONNECTION and SQL_COMMAND is preferred to using the verb SVSQL (based on ODBC).
Please refer to the Application Explorer book for more information about Sql connection configuration.

Some of the modes are asynchronous. The application developer must ensure that the value of status variables are monitored and take adequate actions when they change.

This verb can be executed either locally on the producer station of the Sql connection or from any client station of the producer. In the latter case, commands and returned data are routed transparently through the multi-station messaging system. Multiple connections to the same data source are only allowed if using different Sql Connections. Multiple connections using the same Sql Connection will be refused.

A connection and the data associated to it are managed in the context of user sessions. A connection should only be handled in the context of the session that had originally initiated it; and all related objects must be properly released/closed before the session is ended. For example it is not possible to initiate a connection at start up and have requests executed in any user context.

Mode	Mnemonic	Syntax
1	TESTCONNECTION	1
2	START	2
3	STOP	2

Arguments common to more than one mode

Argument	Meaning
Sql*ConnectionName*	The name of an Sql connection as configured in General.Data connections in the Application Explorer. Type STR

Syntax 1

IntVal = SQL_CONNECTION(*Mode, SqlConnectionName, StatusVariable, ErrorTextVariable*);

Argument	Meaning
StatusVariable	The name of the register variable used to monitor the status of an asynchronous operation. Succeeded = 0 Running = 1 Failed = 2
ErrortextVariable	The name of a text variable used to return additional information when the status variable value is set to failed (2).

Execution

Mode	Mnemonic	Action
1	TESTCONNECTION	Test the connection with a Data Source using the specified Sql connection. Return: 0 if successful, else a negative number indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_PARAM_NOT_STR_OR_NOT_READ STATUSVARIABLE_NOT_REGISTER_OR_DOES_NOT_EXIST ERRORTEXTVARIABLE_PARAM_NOT_STR_OR_NOT_READ ERRORTEXTVARIABLE_NOT_TEXT_OR_DOES_NOT_EXIST SQLCON_DOES_NOT_EXIST SEND_COMMAND_FAILED See the table below for integer return values.

This mode is asynchronous. The program must monitor the value of the status variable.

Syntax 2

IntVal = SQL_CONNECTION(*Mode, SqlConnectionName*);

Execution

Mode	Mnemonic	Action
2	START	Change the specified Sql connection object state to 'started'. No action is performed if it is already started. Return: 0 if successful, else a negative number indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ SQLCON_DOES_NOT_EXIST SEND_COMMAND_FAILED See the table below for integer return values.
3	STOP	Change the Sql Connection state to 'stopped'. No action is performed if it is already stopped. Return: 0 if successful, else a negative number indicating one of the following errors. SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ SQLCON_DOES_NOT_EXIST SEND_COMMAND_FAILED See the table below for integer return values.

Similar to the state of communication objects, these modes are designed to prevent a Sql connection's producer station to execute actual commands towards the Data Source by giving the application designer the ability to set Sql connections in a state (stopped) in which actual commands are not processed. The modes START and STOP are always executed locally (as opposed to routed to the station producing the Sql connection). As a consequence, they must be executed on producer station(s) of the Sql connection to be truly effective - as opposed to being executed on stations issuing requests that are routed to the Sql connection producer over the multi-station network.

List of possible return values and meanings

Return value	Enum	Description
0	OPERATION_SUCCEEDED	Operation completed s
-1	SQLCONNAME_PARAM_NOT_STR_OR_NOT_READ	The parameter *SqlCon* read

-2	STATUSVARIABLE_PARAM_NOT_STR_OR_NOT_READ	The parameter *StatusV*
-3	ERRORTEXTVARIABLE_PARAM_NOT_STR_OR_NOT_READ	The parameter *ErrorTe*
-4	SQLCON_DOES_NOT_EXIST	The specified Sql conn
-5	STATUSVARIABLE_NOT_REGISTER_OR_DOES_NOT_EXIST	The status variable pas does not exist
-6	ERRORTEXTVARIABLE_NOT_TEXT_OR_DOES_NOT_EXIST	The error text variable does not exist
-7	SEND_COMMAND_FAILED	Failed to send the Sql service is stopped or if execute the command

SQRT

See Also

Square root function.

WebVue support - Yes.

Syntax

DblVal = SQRT(*Val*);

The return type is DOUBLE.

Argument	Meaning
Val	The value of which to take the square root. Any numeric type.

Execution

The square root of the value is returned. The value must be positive, else 0 will be returned.

Example

```
SUB Main()
DIM dblResult as Double;
DIM sngValue as Single;

sngValue = 4;
dblResult  = SQRT (sngValue);
PRINT("Return value = ",dblResult);
'Display "Return value = 2"
END SUB
```

STATION_FILTER

See Also Example

Applies a Population filter to the database of a station to control the distribution of variable values.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	CLEAR	1
2	APPLY	2, 3
3	DUMP	1

Syntax 1

IntVal = STATION_FILTER(*Mode*, *Distribution*);

The return type is INTEGER.

Argument	Meaning
Distribution	A code that defines which functions of the Supervisor are to be affected: 1 Alarm Viewers and audible alarm. 2 Log Viewers. 3 Recording log data on the printer. 4 Recording log data on the archive units. 5 Sending values by the operator. 6 Sending values from the Run macro animation. 7 Sending values from the program language. 8 Accepting alarms using the Run macro animation. 9 Accepting alarms using the program language.

Execution

Mode	Mnemonic	Action
1	CLEAR	All population filters that have been previously applied using the specified distribution code are cleared. Return: 1 if successful, else 0.
3	DUMP	Display the configuration of the specified distribution code in the results area of the SCADA BASIC Program Management window. (You can display this window using F9.) Return: 1 if successful, else 0.

Syntax 2

IntVal = STATION_FILTER(*Mode*, *Distribution*, *List*);

The return type is INTEGER.

Argument	Meaning
Distribution	See syntax 1.
List	A comma-separated list of population names, for example ZONE1,ZONE2,COMMON. Type STR.

The list is limited to a maximum of 64 names and a total of 1,024 characters.

Execution

Mode	Mnemonic	Action
2	APPLY	The list of population filters is applied to the function defined by the distribution code, so as to display information for all populations in the list. Return: 1 if successful, else 0.

Syntax 3

IntVal = STATION_FILTER(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning
Handle	The location of a memory buffer containing a list of population filters and a function code. Type LONG.

Execution

Mode	Mnemonic	Action
2	APPLY	The list of population filters is applied to the function defined by the distribution code. If there are two or more populations on the list they are combined using an OR. That is the information belonging to all populations is displayed. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

STOP

See Also

Stop the current program.

WebVue support - Yes.

Syntax

STOP();

There is no return type.

Execution

The current program is stopped.

If the STOP instruction is used in a function in the global program, it is the program that called the function that is stopped.

Example

```
SUB Main()
While(1)
  Print("In the loop");
  'End of loop and program:
  Stop();
Wend
END SUB
```

STRING

See Also

Return a string composed of the same character repeated a number of times.

WebVue support - Yes.

Syntax

StrVal = STRING(*N*, *Char*);

The return type is STR.

Argument	Meaning
N	The number of characters to return. Any numeric type.
Char	The character to be repeated.

Execution

The returned string is limited to 2,047 characters.

Example

```
SUB Main()
DIM strChain as Str;

strChain = STRING(5,"A");
PRINT(strChain);
'Display: "AAAAA"
END SUB
```

SUB...ENDSUB

See Also Example

Define the start and end of a function.

WebVue support - Yes.

Syntax

SUB subname([*P1*, *P2* .....*P10*])

  [instruction list]

ENDSUB

Argument	Meaning
P1 to *P10*	Parameters to be passed to the function. All types supported.

Execution

A function *subname* is created. The parameters are not type cast but they must match the use within the function or a run time error will occur.

It is obligatory to have one MAIN routine per program (except for the global declaration program).

A value may be returned from a function using RETURN.

Example

For an example, select the Example link above.

SVAL

See Also

Return the numeric value of a character string.

WebVue support - Yes.

Syntax

SglVal = SVAL(*String*);

The return type is SINGLE.

Argument	Meaning
String	The character string. Type STR.

Execution

If the string contains a character or character sequence that is recognised as non-numeric, zero is returned.

Example

```
DIM s1 As Single;
s1=SVAL("123.456");
```

SVALA

See Also Example Further information

Extraction of alarm information.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	EXTRACT	1
2	COUNT	2

Syntax 1

IVal = SVALA(*Mode*, *DestHandle*, *ParamHandle* [,*EndVarName*, *Sense*] [, *CountVarName*]);

Argument	Meaning
DestHandle	The handle of a buffer in which the retrieved alarms will be placed. Type LONG.
ParamHandle	The handle of a buffer that contains parameters that will be used to filter the alarms to be extracted. The parameters take the format of a comma-separated string. Type LONG.
EndVarName	The name of a database bit variable that will be set to the value contained in *Sense* when the extract has finished. Type STR.
Sense	A flag (0 or 1) which will be written into the variable *EndVarName* when the extract has finished. Any numeric type.
CountVarName	The name of a database register variable into which will be placed the number of alarms extracted. Type STR.

Execution

Mode	Mnemonic	Action
1	EXTRACT	Alarms passing the filter criteria are extracted and placed in the buffer defined by the parameter *DestHandle*. Return values: 1 The instruction was executed successfully. 0 The mode is incorrect. -1 The parameter *DestHandle* is incorrect. -2 The parameter *ParamHandle* is incorrect. -3 The content pointed to by *ParamHandle* is incorrect. -4 The variable *EndVarName* does not exist. -5 The variable *EndVarName* cannot be written to. -5 The variable *EndVarName* cannot be written to. -6 The parameter *Sense* does not exist or has the wrong value. -7 The variable *CountVarName* does not exist. -8 The variable *CountVarName* cannot be written to. -9 A request is in progress.

Syntax 2

IVal = SVALA(*Mode*, *ParamHandle* [,*EndVarName*, *Sense*] [, *CountVarName*]);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	COUNT	Alarms passing the filter criteria are counted and the result placed in the variable CountVarName.

Return values:

1 The instruction was executed successfully.
0 The mode is incorrect.
-1 The parameter *DestHandle* is incorrect.
-2 The parameter *ParamHandle* is incorrect.
-3 The content pointed to by *ParamHandle* is incorrect.
-4 The variable *EndVarName* does not exist.
-5 The variable *EndVarName* cannot be written to.
-6 The parameter *Sense* does not exist or has the wrong value.
-7 The variable *CountVarName* does not exist.
-8 The variable *CountVarName* cannot be written to.
-9 A request is in progress.

For the format of the parameter buffer, see the topic Parameter Buffer Format and the topic on Native Filter Expressions.

Example

For an example, select the Example link above.

SVBATCH

See Also Specifics

SVBATCH produces a user configurable record of the start and end of a production batch in a database style format. Along with the start and end times, each batch record contains 1 LONG attribute (that may also be treated as 32 individual bits) and up to 32 string attributes. You can have one or more databases that are loaded into memory for rapid access.

Modes are provided to create, delete and modify and save the batch records. The string attributes may be configured as keys (as in a database system such as Microsoft Access) and used to rapidly search and filter very large numbers of batch records.

Before using SVBATCH it is essential to read the reference topic Batch Data Structures.

For the network mode, see the topic Network Mode of SVBATCH.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	CREATEBASE	1
2	SELECTBASE	2
3	BASELIST	3, 9
4	CREATE	4
5	UPDATE	5, 10
6	EXIST	6, 11
7	GETVALUE	7, 12
8	DELETE	3, 13
9	SELECT	8, 16
10	SAVEBASE	2
11	LOADBASE	2
12	GETNEXTBUFFER	3
13	CANCEL	3
14	ARCHIVE	13
15	ARCHIVELIST	3
16	BATCHLIST	3
17	NETWORKBROADCAST	14
18	NEXTCOMMAND	15

All syntaxes

Argument	Meaning
Basename	The name of the batch database. Type STR.
Handle	A handle representing a memory location. Type LONG.
Type	A mnemonic for one of the fields forming a batch record. #I1 IdName #I2 StartDate #I3 EndDate [#A1 to #A32] Modification of a text attribute. [#B] Modification of a LONG attribute. [#B1 to #B32] Modification of a bit attribute.

Value	The new value for the selected field: STR for Type = [#I1 to #I3] or [#A1 to #A32] DOUBLE for Type= [#I2 to #I3] LONG for Type = [#B] INTEGER for Type = [#B1 to #B32]
Table	The name of a batch database table. Type STR.
StartDate	The start date and time of a batch record. Type DOUBLE or STR.
Enddate	The end date and time of a batch record. Type DOUBLE or STR.

If the parameters *StartDate* and *EndDate* are of type DOUBLE they may be calculated using the instruction DATETIMEVALUE which returns a date and time in the form of a DOUBLE. If the date is supplied as a string it must conform to the following syntax: DD/MM/YYYY hh:mm:ss.lll For example : 12/05/2001 21:49:01.100.

The instruction DATETIMESTRING may be used to convert a date saved as a DOUBLE to text format.

Syntax 1

IVal = SVBATCH (*Mode*, *Basename* ,*Unused*, [*NbAttribute*, *NbBackup*])

The return type is INTEGER.

Argument	Meaning
Unused	This field is unused but must be included as a null string "". Type STR.
NbAttribute	The number of attribute fields. Default 4. Type INTEGER. Range 1 to 32.
NbBackup	The number of backup copies of the database that are created. Default 3. Type INTEGER. Range 3 to 9.

Execution

Mode	Mnemonic	Action
1	CREATEBASE	Create a batch database. The database is created in memory. Return: 0 the database was not created, 1 it was created.

The batch database is saved in the folder PER of the project when the Supervisor is shut down.

The *Unused* argument must be included in the syntax as a null string.
Example: SVBATCH ("CREATEBASE", "MyDatabase", "");

Syntax 2

iVal = SVBATCH (*Mode*, *Basename*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	SELECTBASE	Select the operational batch database. The database must have been previously created or loaded in memory. Return: 0 Batch database is not available. 1 Batch database is selected OK. -100 Error in the arguments.

10	SAVEBASE	Save the batch database. The database is saved in the PER folder in a sub-folder with the same name as that of the database. Return: 0 Error when saving. 1 Save successful.
11	LOADBASE	Load the batch database from the folder \PER\BaseName into memory. Return: 0 Error when loading. 1 Load successful. -100 Error in the arguments.

Syntax 3

IVal = SVBATCH (*Mode*, *Handle*)

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
3	BASELIST	List the names of the loaded batch databases, separated by commas. The list is saved in a memory buffer using the supplied memory location. Return: The number of databases. -100 Error in the arguments.
8	DELETE	Delete a batch record using the supplied memory location. Return: 0 Deletion failed. 1 Deletion successful.
12	GETNEXTBUFFER	Used after mode SELECT to retrieve the next buffer of information. The handle used must be the same as that used for the *ResultHandle* in the SELECT. If the size of the buffer is not large enough, this mode lets you extend it. When the final buffer of data is retrieved the internal storage used by mode SELECT is released. (You still need to execute a free-buffer command to release the memory allocated to the handle.) Return type INTEGER: 0 Final buffer of data. 1 One or more buffers of data still to retrieve. -100 Error in the arguments.
13	CANCEL	Cancel a previous SELECT mode, releasing its memory area. The handle used must be the same as that used for the *ResultHandle* in the SELECT. CANCEL may be used before all the data has been retrieved by mode GETNEXTBUFFER. Return: 0 There is no outstanding SELECT to cancel. 1 Mode SELECT cancelled OK. -100 Error in the arguments.
15	ARCHIVELIST	Return, in a buffer, a list of start and end dates for archived records. Return:

0 No records to return.
-100 Error in the arguments.

Else the number of table names returned.

The mode GETNEXTBUFFER can be used when the buffer *Handle* cannot contain all the data to be archived.

16	BATCHLIST	Return, in a buffer, a list of tables contained in the selected batch database. The table names are delimited by commas. Return: 0 No table names to return. -100 Error in the arguments. Else the number of table names returned.

When using BASELIST (mode 3) the format of the returned data is as follows:

Batch record name, number of records \n
Batch record name, number of records \n
(where \n = Chr(10) or line feed)

For GETNEXTBUFFER (mode 12) and CANCEL (mode 13), the buffer *Handle* must be the same as that used by the SELECT command.

Example of a buffer list

The contents of the buffer *Handle* could be as follows:

```
14/10/1997 10:17:55:870 14/10/1998 11:46:09:100\n
14/10/1997 10:17:55:870 14/10/1998 12:40:40:574\n
14/10/1998 10:17:55:870 14/10/1998 11:18:05:870\n
14/10/1998 11:03:11:678 14/10/1998 11:03:21:678\n
14/10/1998 11:15:34:066 14/10/1998 11:15:44:066\n
14/10/1998 11:17:55:870 14/10/1998 11:18:05:870\n
14/10/1998 11:47:36:901 14/10/1998 12:43:19:052\n
14/10/1998 12:43:09:242 14/10/1998 12:43:24:249\n
```

Syntax 4

IVal =SVBATCH (*Mode*, *Table*, *StartDate* [, *EndDate*])

The return type is LONG.

Execution

Mode	Mnemonic	Action
4	CREATE	Create a new record in the currently selected batch database. If the table does not exist it will be created. Return: The location of the batch record in memory. (If 0 then creation failed.) Type LONG.

The location returned may be used in the modes UPDATE, EXIST, and DELETE.

The start date and time must be unique. Creation with a date and time already used will cause a failure. The end date and time may be changed subsequently using the mode UPDATE.

Syntax 5

IVal = SVBATCH (*Mode*, *Handle*, *Type*, *Value*)

The return type is INTEGER

Execution

Mode	Mnemonic	Action
5	UPDATE	Update the batch record changing the value of the specified field. Return: 0 Update impossible. 1 Update successful.

A batch record may be updated using either the memory location returned by modes CREATE or SELECT, or by using the *Table* and *StartDate* parameters (Syntax 10).

Syntax 6

IVal = SVBATCH (*Mode*, *Handle*)

The return type is LONG.

Execution

Mode	Mnemonic	Action
6	EXIST	Test the existence of a batch record using the supplied memory location. Return: The handle of the batch record, else 0 if it does not exist.

Syntax 7

SVBATCH (*Mode*, *Handle*, *Type*)

Execution

Mode	Mnemonic	Action
7	GETVALUE	Retrieve the value of a single field from the selected batch record. Return: Depends on the field type.

Syntax 8

SVBATCH (*Mode*, *Table*, *StartDate*, *EndDate*, *FormatHandle*, *ResultHandle* [,*ExpHandle* [,*LogVarName*]]);

The return type is INTEGER.

Argument	Meaning
FormatHandle	The handle of a location in memory containing a string of comma separated mnemonics that specifies the data to be retrieved. The possible mnemonics are : [#I1 to #I3](Table name, Start date and End date) [#A1 to #A32] (Text attributes 1 to 32) [#B] (Long attribute) [#B1 to #B32] (Binary attributes 1 to 32)
ResultHandle	The handle of a memory location in which the result is placed. Each record is terminated with a carriage return character. Type LONG.
ExpHandle	The handle of a buffer containing a filter expression. See the instruction SVALA for information on filter expressions. Type LONG.
LogVarName	The name of a bit variable that is set to 1 either when the instruction has finished, or the result buffer is full. Type STR

Execution

Mode	Mnemonic	Action
9	SELECT	Selection of records in memory between two dates, optionally using an expression.

Return:

-100 Error in the arguments.
-1 No batch record selected.
-2 Table name unknown.

Else number of records returned.

The operation of mode 9 is asynchronous. That is, once the instruction executed the program thread carries on. When selecting or searching large numbers of records the return of the results may occur after a few seconds.

To manage this, create an event that is triggered when the *LogVarName* changes to 1.

The event can then run a function that processes the results or runs another SVBATCH instruction with the mode GETNEXTBUFFER.

Syntax 9

IVal = SVBATCH (*Mode*, *FileName*)

The return type is INTEGER.

Argument	Meaning
FileName	The name of a file. Type STR.

Execution

Mode	Mnemonic	Action
3	BASELIST	Create a file containing the names of the batch databases that have been loaded into memory. Return: The number of databases. -100 Error in the arguments.

When using Mode 3 the format of the returned data is as follows.

Batch record name, number of records \n
Batch record name, number of records \n

where \n = Chr(10) or line feed.

Syntax 10

IVal = SVBATCH (*Mode*, *Table*, *StartDate*, *EndDate*, *Type*, *Value*)

The return type is INTEGER

Execution

Mode	Mnemonic	Action
5	UPDATE	Update a batch record in memory. Return: 0 Update impossible. 1 Update successful.

An update of a batch record is may be achieved using either the memory location returned by modes CREATEBASE or SELECTBASE (Syntaxes 1 & 2), or by using the *Table* and *StartDate* parameters (Syntax 10). The *EndDate* parameter is optional

Syntax 11

IVal = SVBATCH (*Mode*, *Table*, *StartDate*, *EndDate*)

The return type is LONG.

Execution

Mode	Mnemonic	Action
6	EXIST	Test the existence of a batch record. Return: The memory location of the batch record, else 0 if it does not exist.

Syntax 12

SVBATCH (*Mode*, *Table*, *StartDate*, *EndDate*, *Type*)

Execution

Mode	Mnemonic	Action
7	GETVALUE	Retrieve the value of a single field from the selected batch record. Return: Depends on the field type.

Syntax 13

IVal = SVBATCH (*Mode*, *Table*, *StartDate*, *EndDate*)

The return type is Integer.

Execution

Mode	Mnemonic	Action
8	DELETE	Delete a batch record from memory. Return: 0 Deletion failed. 1 Deletion successful.
14	ARCHIVE	Archive all records from the specified table between the two dates. The records are removed from the database and saved in a file named STARTDATE_ENDDATE. The file is located in the folder Per\"BaseName"\AR\"Table", where BaseName and Table are the names of the selected batch database and table respectively. If the table is not specified (null string) all tables from the selected batch database are archived. Return: 0 Archiving did not take place. 1 Archiving completed. else: -1 No database was selected. -2 Conversion of *StartDate* failed. -3 Conversion of *EndDate* failed. -100 Parameter error.

Effects of archiving

If the batch identifier is absent, the archiving is applied to all of the data.

Archiving automatically purges the batches archived in the current database.

The archive files are created as follows in the PER folder of the project:

Folder: *BatchName*

File: *StartDate EndDate*

Syntax 14

SVBATCH (*Mode*, *StationList*, *StatusHandle*, *EventVarName*)

Argument	Meaning
StationList	The name of a station list as in the Network Station configuration dialog (Configure.Communication.Network Stations). Type STR.
StatusHandle	The handle of a memory location which receives information about the progress of SVBATCH requests. Type LONG.
EventVarName	The name of a bit variable that is set to 1 when either the instruction has finished, or the result buffer is full. Type STR

Execution

Mode	Mnemonic	Action
17	NETWORKBROADCAST	Broadcast SVBATCH commands to all the stations contained in the station list.

Commands are held in a queue and then executed each time a NEXTCOMMAND command is issued. When a command has completed and its results are available, the bit specified by the argument *EventVarName* is set to 1. The status of each command is reported in the buffer pointed to by *StatusHandle* in the format *Mode, Station Number, Result*.

All requests are asynchronous; the result of the request becomes available when the *EventVarName* changes to 1.

At that point the results may be processed and a NEXTCOMMAND issued.

NETWORKBROADCAST may be used with these modes:

SELECT, UPDATE, DELETE, CANCEL, ARCHIVE, BATCHLIST, ARCHIVELIST, GETNEXTBUFFER.

Buffer formats

On completion of the request, the format of the *Status* buffer is as follows:

Mode,*StationName*,*Result*

Field	Meaning
Mode	Mode of the SELECT command (e.g. 9 for SELECT).
StationName	Name of the station (as specified in the topic Communications.Multi-station).
Result	Result code returned by the station.

Syntax 15

SVBATCH (*Mode*);

Execution

Mode	Mnemonic	Action
18	NEXTCOMMAND	Execute the next SVBATCH command in the queue when using NETWORKBROADCAST.

Syntax 16

SVBATCH (*Mode, IdName, StartDate, EndDate, FormatHandle, ResultHandle,* [,*ExpHandle* [,*LogVarName*]]);

Argument	Meaning
ExpHandle	The handle of a buffer containing a filter expression. See the instruction SVALA for information on filter expressions. Type LONG.

LogVarName The name of a bit variable that is set to 1 when either the instruction has finished, or the result buffer is full. Type STR.

Execution

Mode	Mnemonic	Action
16	SELECT	Select records between 2 dates.

The selection works whether the entries have been archived or not.

When the requested batch records are retrieved, *LogName* is set to 1. The buffer *ResultHandle* contains the records up to the limit of its size.

You can now allocate buffers of 64KB by ALLOC_BUFFER.

In this mode, you need to create an event on *LogVarName*. The request is carried out asynchronously, then *LogVarName* becomes 1 and *ResultHandle* identifies the buffer used. For details, see Syntax 8 above.

The event can be created by the Actions menu or dynamically by the verb EVENT.

SVBRANCH

See Also

Change the status of a group of variables according to their branch.

WebVue support - Yes.

Mode	Mnemonic	Syntax
2	MASK	1
3	UNMASK	1
4	ENABLE	2
5	DISABLE	2
8	SIMU	3
12	SETALARMLEVEL	5
13	RESTOREALARMLEVEL	2
14	RESETANDSETALARM	4
15	RESETCOMPUTEALARM	4

Syntax 1

SVBRANCH (*Mode*, *Branch*, *MaskLevel*)

The return type is INTEGER.

Argument	Meaning
Branch	The branch. Type STR.
MaskLevel	The mask level. Type INTEGER: 1 User program 1 2 User program 2 4 User program 3 8 User program 4 16 Operator 32 Dependent (on another variable) 64 Expression

Execution

Mode	Mnemonic	Action
2	MASK	All variables in the selected branch are masked. See also the VARIABLE instruction mode MASK. The mask level depends on the MaskLevel parameter.
3	UNMASK	All variables in the selected branch are unmasked. See also the VARIABLE instruction mode UNMASK. The mask level depends on the MaskLevel parameter. Return: 1 OK, else 0.

The *MaskLevel* parameter is an integer whose value represents the binary combination of one or more levels. Values 0 to 29. For example a value of 11 would represent User program 1, User program 2 and User program 4.

Syntax 2

SVBRANCH (*Mode*, *Branch*)

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
4	ENABLE	All variables in the selected branch are enabled.
5	DISABLE	All variables in the selected branch are disabled.
13	RESTOREALARMLEVEL	Restore the configured alarm level of all alarm variables in the selected branch (after SETALARMLEVEL). Return: 1 OK, else 0.

Syntax 3

SVBRANCH (*Mode*, *Branch*, *Flag*)

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
8	SIMU	The setting of Flag determines whether all equipment variables belonging to the selected branch become simulated or non-simulated: 1 simulated. 0 non-simulated Return: 1 OK, else 0.

Syntax 4

SVBRANCH (*Mode, Branch[, Filter[, Type]]*)

The return type is INTEGER.

Argument	Meaning
Branch	The branch. Type STR.
Filter	A filter expression. See the topic Native Filter Expressions. Type STR.
Type	The alarm type. Type STR. ALARMALLTYPE Perform the action on all alarms. The default value. ALARMTHRESHOLD Perform the action on alarms generated from the threshold system only.

Execution

Mode	Mnemonic	Action
14	RESETANDSETALARM	Toggles (resets and set) all alarms that are on and pass the filter criteria (if any). It has no effect if the alarm is off.
15	RESETCOMPUTEALARM	Reset to 0 and re-calculate the number of alarms for a branch and a filter. Return: 1 OK, else 0.

Syntax 5

SVBRANCH (*Mode*, *Branch*, *AlarmLevel*)

The return type is INTEGER.

Argument	Meaning
Branch	The branch. Type STR.
AlarmLevel	Priority level of alarms (0 to 29). Type INTEGER:

Execution

Mode	Mnemonic	Action
12	SETALARMLEVEL	Change all alarms to the supplied alarm level. See also the ALARM instruction mode SETALARMLEVEL. Return: 1 OK, else 0.

SVKEY

See Also

Read the contents of the protection key on multi-station systems.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	READ	1
2	ORDER	2
3	CANCEL	2

Syntax 1

Ival = SVkey(*Mode[, LogVar[, ResultVar]]*);

Argument	Meaning
LogVar	The name of a variables tree bit variable. Type STR.
ResultVar	The name of a variables tree register variable. Type STR.

The return type is INTEGER.

1 = OK.
-1 = The previous request is still being processed.
-5 = The variable *LogVar* does not exist or is the wrong type.
-6 = The variable *ResultVar* does not exist or is the wrong type.

Execution

Mode	Mnemonic	Action
1	READ	Request a re-read of the licence from the protection key on the local station. Value of the LogVar bit variable: 0 = Request in progress. 1 = Request complete. If the read is successful the result is available in the ResultVar register. Value of ResultVar register variable: 0 = Request in progress. 1 = Request successful. -1 = Request failed (no key).

Syntax 2

Ival = SVkey(*Mode, RightsType, Slot, Station[, LogVar[, ResultVar]]*);

Argument	Meaning
RightsType	Always 1 Type INTEGER.
Slot	1 = The first slot. 2 = The second slot. Type INTEGER.
Station	The station number on which the key is connected. Type INTEGER.
LogVar	The name of a variables tree bit variable. Type STR.
ResultVar	The name of a variables tree variable. Type STR.

The return type is INTEGER.

1 = OK.
-1 = The previous request is still being processed.
-2 = The parameter *RightsType* is out of range.

-3 = The station number was not found in this project.
-4 = The parameter *Slot* is out of range.
-5 = The variable *LogVar* does not exist or is the wrong type.
-6 = The variable *ResultVar* does not exist or is the wrong type.

Execution

Mode	Mnemonic	Action
2	ORDER	Request (order) a licence from a protection key on another station. Value of the LogVar bit variable: 0 = Request in progress. 1 = Request complete. If the request was successful the result is available in the ResultVar register. Value of ResultVar register variable: 0 = Request in progress. 1 = Request successful. The licence was read. -1 = Request failed. No protection key found. -2 = Request failed. No licences remaining. -3 = The licence has already been used. -4 = Request failed, network disconnection or PC failure. -5 = Request timeout.
3	CANCEL	Release a licence previously read from a protection key on another station. Value of the LogVar bit variable: 0 = Release in progress. 1 = Release complete. If successful the result is available in the ResultVar register. Value of ResultVar register variable: 0 = Request in progress. 2 = Request successful. The licence was released. -3 = The licence to release was not found. -4 = Request failed, network disconnection or PC failure. -5 = Request timeout.

SVLOG

See Also Example Further information

Extract of logged event records.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	EXTRACT	1

Syntax 1

LVal = SVLOG(*Mode*, *DestHandle*, *ParamHandle* [,*EndVarName* [, *StatusVarName* [, *LinesVarName* [,*StartTime* [,*EndTime*]]]]]);

The return type is LONG.

Argument	Meaning
DestHandle	The handle of a buffer in which the extracted log information will be placed. Type LONG.
ParamHandle	The handle of a buffer that contains parameters which will be used to filter the logs to be extracted. The parameters take the form of a comma-separated string. Type LONG.
EndVarName	The name of a database bit variable that will be set to 1 when the extract has finished. Type STR.
StatusVarName	The name of a database register variable that will hold the status of the instruction. This parameter is ignored if you use a null string. Type STR. 0 All required logs are extracted. 1 All logs not extracted due to the line limit parameter *MaxLines*. 2 All lines not extracted due to the limit imposed by the size of the destination buffer. Each line in the buffer will require approximately 300 characters.
LinesVarName	The name of a database register variable into which will be written the number of lines returned. A negative number indicates an error. This parameter is ignored if you use a null string. Type STR.
StartTime	The name of a variable in which will be written the time of the first record extracted. If a register is used the value represents the number of milliseconds since 1980 (See instruction DATETIMEVALUE). If a text variable is used the format is #D/#M/###Y #h:#m:#s:###I. This parameter is ignored if you use a null string. Type STR.
EndTime	The name of a variable in which will be written the time of the last record extracted. See above for format information. Type STR.

Execution

Mode	Mnemonic	Action
1	EXTRACT	Logs that meet the filter criteria are extracted and placed in the buffer defined by the parameter *DestHandle*. Return: 0 if syntax of instruction is incorrect, else OK.

For the format of the parameter buffer, see the topic Parameter Buffer Format and the topic on Native Filter Expressions.

The destination buffer size is limited to 32 Kb. A possible alternative is to use EXPORT_LOG.

Attributes names used in Filter Expressions are not permitted to include spaces. For example (#A5==T 5) is not permitted.

Example

For an example, select the Example link above.

SVSQL

See Also Example

Database management using SQL commands

WebVue support - Yes.

As of version 12.0, the use of pre-configured ADO.Net Sql connections along with the SQL_CONNECTION and SQL_COMMAND is preferred to using the verb SVSQL (based on ODBC). Please refer to the Application Explorer book for more information about Sql connection configuration.

'Database' is used here in the general sense, and bears no relation to the SV Configuration Variables Tree (previously known as the configuration 'database').

Mode	Mnemonic	Syntax
1	CONNECT	1
2	DISCONNECT	2
3	BEGINTRANS	2
4	COMMIT	2
5	ROLLBACK	2
6	EXECUTE	3, 7
7	FETCH	2
8	GETCOL	4
9	GETROW	5
10	NUMCOL	2
11	ROWCOUNT	2
12	GETCOLNAME	4
13	GETCOLSIZE	6
14	FREE	2
15	ERROR	7

The SVSQL instruction allows SCADA BASIC to access an ODBC source using SQL commands. The ODBC source must first be configured using the host operating system. How to do this is explained in the book Using ODBC.

Syntax 1

LongVal = SVSQL(*Mode*, *Source* [,*UserID*] [,*Password*] [,*Timeout*]);

The return type is LONG.

Argument	Meaning
Source	The name of the ODBC source. Type STR.
UserID	The user name as entered when the ODBC source was configured. Type STR.
Password	The user password as entered when the ODBC source was configured. Type STR.
Timeout	The connection time-out in seconds. Type LONG.

Execution

Mode	Mnemonic	Action

1	CONNECT	An attempt is made to connect to the specified ODBC source. Return: >0 The return is the handle for the connection which is then used in subsequent transactions. -1 An error, either a problem in the instruction syntax or an error when connecting.

Syntax 2

LongVal = SVSQL(*Mode*, *CHandle*);

The return type is LONG.

Argument	Meaning
CHandle	The handle returned when connecting to an ODBC source. Type LONG.

Execution

Mode	Mnemonic	Action
2	DISCONNECT	An attempt is made to disconnect from the ODBC source using the given handle. If a transaction is in progress it should be stopped or completed using COMMIT or ROLLBACK before attempting to DISCONNECT. If you execute a DISCONNECT before a transaction is finished, a ROLLBACK will be automatically executed. Return: 0 if disconnection successful. -1 if there is a problem with the syntax or with disconnection. -2 if the handle is incorrect.
3	BEGINTRANS	Start a transaction. By default a connection operates in auto-commit; all modifications to the Variables Tree are automatically registered. of this function causes the auto-commit operation to cease. Auto-commit is restored when executing either a COMMIT or ROLLBACK function. Return: 0 if transaction successful. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. Not all ODBC drivers support transactions and those that do treat them in a different ways. This instruction only operates under the following conditions: The driver supports transactions. When a COMMIT or ROLLBACK instruction is processed the driver maintains the cursors in their current position or closes them, but does not destroy them. No requests are being processed. (Use the FREE function)
4	COMMIT	Execute a COMMIT. Modifications made to the ODBC source during the current transaction are accepted. Return: 0 if successful. -1 if there is a problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. Before you can execute a COMMIT you must first have started a transaction with BEGINTRANS instruction. The operation of COMMIT depends on the ODBC

driver used.

5	ROLLBACK	Execute a ROLLBACK. Cancel any modifications made to the ODBC source during the current transaction. Return: 0 if successful. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. Before you can execute a ROLLBACK you must first have started a transaction with BEGINTRANS instruction. The operation of ROLLBACK depends on the ODBC driver used.
7	FETCH	Fetches a single record from the result of an EXECUTE instruction. Each time the instruction runs it increments an internal pointer so that the next time it runs it returns the next record and so on until all records have been retrieved. This behaviour is similar to reading records from a sequential file. Return: 0 if FETCH is successful and a record was returned. 1 if FETCH is successful but no records returned. (Either there were no records to return or the last record was returned by the previous FETCH.) -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. FETCH places a record in an internal buffer. To retrieve individual fields it must be followed with a GETCOL or GETROW instruction.
10	NUMCOL	Return the number of columns resulting from executing a command. Return: >=0 the instruction was executed successfully. The return is the number of columns. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. The instruction must be preceded by an EXECUTE.
11	ROWCOUNT	Return the number of records affected by an Insert, Update or Delete query. Return: >=0 the instruction was executed successfully. The return is the number of records. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. The instruction must be preceded by an EXECUTE.
14	FREE	Frees resources used by an SQL request. Return: 0 if successful. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. This instruction allows the recovery of resources that may still be allocated due to a failed request. A FREE action is also executed automatically when the instruction DISCONNECT is used. The FREE instruction must also be used prior to starting a transaction.

Syntax 3

LongVal = SVSQL(*Mode*, *CHandle*, *MsgString*);

The return type is LONG.

Argument	Meaning
CHandle	The handle returned when connecting to an ODBC source. Type LONG.
MsgString	The SQL command to be executed. Type STR.

Execution

Mode	Mnemonic	Action
6	EXECUTE	The SQL command supplied is executed. Return: 0 if command is executed successfully. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. Only a single SQL command may be active per connection. If you execute another command before the previous one has completed, the previous command will be terminated and the result lost.

Syntax 4

LongVal = SVSQL(*Mode*, *Shandle*, *ColNum*, *BHandle*);

The return type is LONG.

Argument	Meaning
SHandle	The handle returned when connecting to an ODBC source. Type LONG.
ColNum	The column number of the field to retrieve. Starts at 1 and increments from left to right. Type INTEGER.
BHandle	The handle of a memory buffer into which the retrieved field is placed. Type LONG.

Execution

Mode	Mnemonic	Action
8	GETCOL	Retrieves a single field of data and places it in the specified memory buffer. This instruction must be preceded by an EXECUTE and a FETCH. Return: 0 < 2,047 the instruction was executed successfully. The return is the number of characters placed in the buffer. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. -3 if the buffer is not large enough to accommodate the number of characters returned. The field is always retrieved as a character string regardless of its format within the Variables Tree. The maximum length field that can be retrieved is 255 characters.
12	GETCOLNAME	Retrieves the name of a column and places it in a memory buffer. This instruction must be preceded by an EXECUTE and a FETCH. Return: 0 < 2,047 the instruction was executed successfully. The return is the number of characters placed in the buffer.

-1 if there is problem with the syntax or the operation is refused by the driver.
-2 if the handle is incorrect.
-3 if the buffer is not large enough to accommodate the number of characters returned.

Syntax 5

LongVal = SVSQL(*Mode*, *Shandle*, *Bhandle*, *Delim*);

The return type is LONG.

Argument	Meaning
CHandle	The handle returned when connecting to an ODBC source. Type LONG.
BHandle	The handle of a memory buffer into which the retrieved field is placed. Type LONG.
Delim	A chain of characters to be used as a field delimiter. Type STR.

Execution

Mode	Mnemonic	Action
9	GETROW	Retrieves an entire row of data and places it in the specified memory buffer. This instruction must be preceded by an EXECUTE and a FETCH. Return: >= 0 the instruction was executed successfully. The return is the number of characters placed in the buffer. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. -3 if the buffer is not large enough to accommodate the number of characters returned. The data is always retrieved as a character string regardless of its format within the Variables Tree.

Syntax 6

LongVal = SVSQL(*Mode*, *CHandle*, *ColNum*);

The return type is LONG.

Argument	Meaning
CHandle	The handle returned when connecting to an ODBC source. Type LONG.
ColNum	A column number. Starts at 1 and increments from left to right. Type INTEGER.

Execution

Mode	Mnemonic	Action
13	GETCOLSIZE	Retrieves the maximum size of a column from a preceding EXECUTE instruction. Return: >= 0 the instruction was executed successfully. The return is the size of the column. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. -3 if the buffer is not large enough to accommodate the number of characters returned. The size returned corresponds to the number of characters necessary to store

the column as text, rather than the number of bytes that the column occupies.

Syntax 7

LongVal = SVSQL(*Mode*, *CHandle*, *BHandle*);

The return type is LONG.

Argument	Meaning
CHandle	The handle returned when connecting to an ODBC source. Type LONG.
BHandle	The handle of a memory buffer. Type LONG.

Execution

Mode	Mnemonic	Action
6	EXECUTE	The SQL command contained in the memory buffer is executed. Return: 0 if command is executed successfully. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. Only a single SQL command may be active per connection. If you execute another command before the previous one has completed, the previous command will be terminated and the result lost.
15	ERROR	Returns the last error message from the ODBC driver and places in the memory buffer specified by the handle. Return: >=0 if command is executed successfully. The return corresponds to the number of bytes in the message. -1 if there is problem with the syntax or the operation is refused by the driver. -2 if the handle is incorrect. -3 if the buffer is not large enough to accommodate the number of characters returned.

Example

For an example, select the Example link above.

SVTREND

See Also Example

Retrieve the historic values of a group of variables between two given dates.

WebVue support - Yes.

The variables must be configured for trend recording.

Mode	Mnemonic	Syntax
1	GETTREND	1
2	GETNEXTBUFFER	2

Syntax 1

IVal = SVTREND (*Mode*, *ListVar*, *Branch*, *StartDate*, *EndDate*, *Period*, [*DisplayMode*, *Handle_Result*, *Parameters*]);

Argument	Meaning
ListVar	EITHER The handle of a buffer containing a list of variable names (separated by commas) involved in the extraction. Type LONG. OR A list of variable names (separated by commas) involved in the extraction. Type STR.
Branch	The branch of the list of variables. Type STR.
StartDate	The earliest date for extraction, in milliseconds since 1980. Type DOUBLE.
EndDate	The latest date for extraction, in milliseconds since 1980. Type DOUBLE.
Period	The sampling period in seconds. Type DOUBLE.
DisplayMode	The output format for the list of variables. *DisplayMode*= 1 - The data is extracted using the list of variables in the buffer *Handle_Listvar* to the format defined by *Format*. *DisplayMode* = 2 - The sampled data for each variable will be contained on the same line preceded by its time and date stamp. See the example for further details.
Handle_Result	The handle of the buffer in which the result will be placed. Type LONG.
Parameters	EITHER The handle of a buffer containing a list of comma separated values. Type LONG. OR A string containing a list of comma separated values. Type STR. In either case there are four values *Format*, *Filter*, *LogVar*, *NameVar*. See below.
Format	The format of the time and date. See below. Type STR.
Filter	The filter applied in the extraction. See the topic Native Filter Expressions. Type STR.
LogVar	The name of a bit variable that will be set to 1 when the instruction is complete. Type STR.
NameVar	The name of a register variable that will numeric ID of the name of the variable currently being processed. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	GETTREND	Extract trend (Historic) data for the list of variables, between two dates. Return: 1 Parameters are correct. -1 Parameter Handle_ListVar is null or of null length. -2 Variable Str_ListVar could not be read. -3 Parameter 1 is neither of type LONG nor STR.

-4 Parameter 2 is not of type STR.
-5 Parameter 3 is not of type DOUBLE.
-6 Parameter 4 is not of type DOUBLE.
-7 Parameter 5 is not of type DOUBLE.
-8 Parameter 6 is not of type LONG.
-10 Parameter 7 is not of type STR.
-11 Parameter 8 is not of type STR.
-12 Parameter 9 is not of type STR.
-13 Variable StrVar does not exist or has not been configured for trend recording.
-14 One of the items Handle_ListVar or Str_ListVar is not a variable.
-15 One of the variables Handle_ListVar or Str_ListVar does not exist.
-16 One of the variables Handle_ListVar or Str_ListVar is already being processed by SVTREND or HISTORY.
-17 Handle_ListVar or Str_ListVar is empty.

Time and date format characters

Character	Meaning
#D	Date
#M	Month
#Y	Year as two characters
###Y	Year as four characters
#h	Hour
#m	Minute
#s	Second
##l	Millisecond

Any other characters are interpreted literally.

Syntax 2

IVal = SVTREND (*Mode*, *Handle_Result*);

Handle_Result The handle of the buffer in which the data will be placed. Type LONG.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	GETNEXTBUFFER	Continues filling the buffer from the point previously reached.

Return:

0 No more data in the buffer.
1 Parameters are correct.
-1 No previous SVTREND instruction.
-2 Parameter Handle_Result is null or not initialised.

When SVTREND fills the buffer to its limit before all the data have been extracted, the mode GETNEXTBUFFER allows the process of extraction to be continued automatically.

Example

For an example, select the Example link above.

SYSTEM

See Also Example Access Rights Weighting

Read and write the internal system time, change the graphic mode and execute a system function.

Partial WebVue support - see mode table.

The modes SETDATE, SETTIME and SETDATETIME require the Supervisor to be run with elevated privileges to function correctly - specifically SE_SYSTEMTIME_NAME. From Windows Vista and Server 2008 onwards this privilege is not given by default to a user and should not be given for security reason.
These legacy modes are for compatibility only. We recommend setting up a global clock, and time distribution via a network infrastructure level mechanism such as NTP, SNTP...

Mode	Mnemonic	Syntax	WebVue
1	SETDATE	1	No
2	SETTIME	2	No
3	GETDATE	3	Yes
4	GETTIME	3	Yes
5	SETREGION	4	No
6	SETSYSREGION	4	No
7	GETREGION	7	No
8	GETSYSREGION	7	No
10	SYSTEM	5	No
11	LOGIN	6	No
12	LOGOUT	7	No
13	SETDATETIME	8	No
14	LANGUAGE	7	Yes
15	USER	9, 10	Yes
16	EXIT	7	No
18	CHANGEPASSWORD	16	No
19	PRINTER	15	No
20	REPAINT	7	No
23	MKDIR	12	No
24	RMDIR	12	No
25	OPERATORMODE	14	No
26	GETDRIVETYPE	13	No
27	GETDISKSIZE	13	No
28	GETDISKFREESPACE	13	No
29	GETDIRECTORYSIZE	13	No

Syntax 1

IntVal = SYSTEM(*Mode*, *Day*, *Month*, *Year*);

The return type is INTEGER.

Argument	Meaning
Day	The day of the month. Type INTEGER, range 1 to 31.
Month	The month number. Type INTEGER, range 1 to12.
Year	The year number. Type INTEGER, range 1980 to 2099.

Execution

Mode	Mnemonic	Action
1	SETDATE	Modify the internal system date. Return: 1 if successful, else 0.

Syntax 2

Set the system internal time.

IntVal = SYSTEM(*Mode*, *Hour*, *Minute*, *Second*);

The return type is INTEGER.

Argument	Meaning
Hour	The number of hours. Type INTEGER, range 0 to 23.
Minute	The number of minutes. Type INTEGER, range 0 to 59.
Second	The number of seconds. Type INTEGER, range 0 to 59.

Execution

Mode	Mnemonic	Action
2	SETTIME	Modify the internal system time. Return: 1 if successful, else 0

For this to work correctly the computer's time zone must be set to GMT0.

Syntax 3

StrVal= SYSTEM(*Mode*);

The return type is STR.

Execution

Mode	Mnemonic	Action
3	GETDATE	Return the system date in the format dd/mm/yyyy. If executed in a WebVue session context the date from the computer that hosts the web back end is returned.
4	GETTIME	Return the system time in the format hhmmss. If executed in a WebVue session context the time from the computer that hosts the web back end is returned.

Syntax 4

IntVal = SYSTEM(*Mode*, *Region*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action

5	SETREGION	Select the *region* that is to be used: Single screen 1 the default Two screens 1 the left screen. 2 the right screen. Three screens 1 the left screen. 2 the centre screen. 3 the right screen. Four screens 1 the left screen. 2 the left centre screen. 3 the right centre screen. 4 the right screen.
6	SETSYSREGION	Select the number of regions on the system, by setting *Region*. 1 A single screen, the default. 2 Two screens, horizontally placed. 3 Three screens, horizontally placed.

This instruction is used to inform the software that it is running on a multi-screen system. After it has been used, any windows that are subsequently called will appear in the selected region.

Syntax 5

IntVal = SYSTEM(*Mode*, *Command*);

The return type is integer.

Argument	Meaning
Command	A string containing an operating system command.

Execution

Mode	Mnemonic	Action
10	SYSTEM	The operating system command contained in the string is executed. This is used for operating system resident commands. It cannot be used for launching other programs. Return: Always 0.

Syntax 6

IntVal = SYSTEM(*Mode*, *Username*, *Password*);

The return type is INTEGER.

Argument	Meaning
Username	The name of a user known to the system. Type STR.
Password	The password for the above user. Type STR.

Execution

Mode	Mnemonic	Action
11	LOGIN	Log the named user onto the system. Return: 1 if successful, else 0.

Syntax 7

IntVal = SYSTEM(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
7	GETREGION	Return the current region. (See Syntax 4 for explanation.)
8	GETSYSREGION	Return the current region set-up. (See Syntax 4 for explanation.)
12	LOGOUT	Log the current user off the system. Return: Always 1.
14	LANGUAGE	Return the language code of the current session. Return: 1 or 2 if OK, else 0.
16	EXIT	Shut down the Supervisor.  Shutdown occurs without further User confirmation unless there are any unsaved changes to the mimics. Return: 1 if the shutdown is successful. 0 if shutdown is impossible. For example, if a dialog box is open.
20	REPAINT	Refresh the screen (equivalent to function key F11). Return: Always 1.

Syntax 8

IntVal = SYSTEM(*Mode*, *Date*, *Time*);

The return type is INTEGER.

Argument	Meaning
Date	A string in the format DD/MM/YY. Type STR.
Time	A string in the format HHMMSS. Type STR.

Execution

Mode	Mnemonic	Action
13	SETDATETIME	Change the system date and time. Return: 1 if successful, else 0.

 This mode is not available when SYSTEM is used in a Macro animation.

Syntax 9

StrVal = SYSTEM(*Mode*, *Rights*);

The return type is STR.

Argument	Meaning
Rights	Always 0.

Execution

Mode	Mnemonic	Action

15	USER	Returns the name of the user for the current session.

Syntax 10

IntVal = SYSTEM(*Mode*, *Rights*);

The return type is INTEGER.

Argument	Meaning
Rights	A number specifying which set of rights to return. Type INTEGER. For a list of weights for Administration, System Access, Recipe and window printing rights, see the topic Access Rights Weighting. 1 System access. 5 Recipe. 7 Administration (as in a user profile). 10 Window printing. --- For command, window, alarm acknowledge and layer access: the binary weight corresponds directly to the level. For example, a return of 9 (binary 1001) would correspond to levels 1 and 4. 2 Command. 3 Window. 4 Alarm acknowledge. 6 Layer access. 8 Masking (0 to 29). --- 9 (Not used.)

Execution

Mode	Mnemonic	Action
15	USER	Return an integer representing the rights of the user for the current session. The rights are returned as binary weights within the integer.

For more information see the book User Rights in the help for Configuration of the Supervisor.

Syntax 12

IntVal = SYSTEM(*Mode*, *Folder*);

The return type is INTEGER.

Argument	Meaning
Folder	The name of a folder. For example "C\\NEWDIR". Type STR.

Execution

Mode	Mnemonic	Action
23	MKDIR	The specified folder is created. Return: 1 if successful, else 0.
24	RMDIR	The specified folder is deleted. Return: 1 if successful, else 0.

Syntax 13

IntVal = SYSTEM(*Mode*, *DiskInfo*);

Argument	Meaning
DiskInfo	The name of a disk or folder. You must include a colon after the disk name (for example "C:") and two back slashes when specifying a folder (for example "C:\\temp"). Type STR.

Execution

Mode	Mnemonic	Action
26	GETDRIVETYPE	Return the disk type as configured in the BIOS. The return type is INTEGER.
27	GETDISKSIZE	Return the disk size in kilobytes. The return type is LONG.
28	GETDISKFREESPACE	Return the free disk space in kilobytes. The return type is LONG.
29	GETDIRECTORYSIZE	Return the size of the specified folder in kilobytes. The return type is LONG.

Syntax 14

IntVal = SYSTEM(Mode, Flag[, Global[, SetSendList]]);

The return type is integer.

Argument	Meaning
Flag	A flag, either 0 or 1: 1 logging is enabled 0 logging is disabled. Default value.
Global	For when *Flag* is 1: 0 logging is enable only in function where variables are set. (default) 1 global, logging is enable in all SCADA Basic functions. Optional.
SetSendList	0 If *Flag* = 1 then logging with the current user is done on variables set by the Set/SendList instruction, else no logging takes place. 1 For compatibility, logging without the current user is done on variables set by the Set/SendList instruction independently of *Flag*'s value. Default value is 0 for a new project. Optional.

Execution

Mode	Mnemonic	Action
25	OPERATORMODE	Enables the logging of variables set by a program to be attributed to a user. Return: 1 if OK, else 0.

The instructions SYSTEM("OPERATORMODE", 0, 0); and SYSTEM("OPERATORMODE", 0, 1); have the same effect: logging is disabled in all SCADA Basic functions.

Syntax 15

IntVal = SYSTEM(*Mode*, *PrinterNo*, *Flag*);

The return type is integer.

Argument	Meaning
PrinterNo	Number of a printer for line printing.
Flag	A flag, either 0 or 1.

Execution

Mode	Mnemonic	Action
19	PRINTER	Sets the line printer spooler to be used. If *Flag* is: 0 spooling is enabled 1 spooling is disabled. Return: 1 if OK, else 0.

Syntax 16

IntVal = SYSTEM(*Mode*, *UserName*,*Password*, *NewPassword*);

The return type is integer.

Argument	Meaning
UserName	The user account name. Type STR.
Password	The password's existing value. Type STR.
NewPassword	The password's replacement value. Type STR.

Execution

Mode	Mnemonic	Action
18	CHANGEPASSWORD	Changes the value of a user's password. Returns: 0 Syntax error 1 Successful 2 Error - User name and old password do not match 3 Error - New password is empty 4 Error - New password length is less than 6 characters - Applied only if the advanced strategy is activated 5 Error - New password has already been used - Applied only if the advanced strategy is activated 6 Error - New password does not meet the strong password criteria - Applied only if the strong password enforcement is activated

Example

For an example, select the Example link above.

T

TAN

See Also

Tangent function.

WebVue support - Yes.

Syntax

DblVal = TAN(*Angle*);

The return type is DOUBLE.

Argument	Meaning
Angle	The angle to be converted expressed in degrees. Any numeric type.

Execution

Register variables are of type SINGLE, so type conversion must be used to assign a value using this function.

Example

```
'variables
'@ARC - type Register
'@ANGLE - type Register
SUB Main()
'declare variables
DIM dblangle as double;
DIM dblarc as double;

dblangle = 45;
dblarc = TAN (dblangle);
PRINT("TAN(",dblangle,") = ",dblarc);
END SUB
```

TEMPORARY_DB

See Also Example

Activation, creation and deletion of temporary variables.

WebVue support - Yes.

For further information see the Variables Tree book in the Configuration Help.

Mode	Mnemonic	Syntax
0	OFF	1
1	ON	1
2	ADDBIT	2
3	ADDREG	3
4	ADDTXT	4

Syntax 1

IntVal = TEMPORARY_DB(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
0	OFF	The automatic creation of temporary variables is disabled. Return: Always 1
1	ON	The automatic creation of temporary variables is enabled. Any variables referred to in an animation that do not exist in the Variables Tree are automatically created at the HMI level, not in the Variables Tree. Return: Always 1

Syntax 2

IntVal = TEMPORARY_DB(*Mode*, *VarName*, *Title_1*, *Title_2*);

Argument	Meaning
VarName	The name for the temporary variable. Type STR
Title_1	The title for the temporary variable in the primary language. Type STR
Title_2	The title for the temporary variable in the alternative language. Type STR

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	ADDBIT	A temporary bit variable is created using the supplied attributes. Return: 1 if successful, 0 if the variable already exists.

Syntax 3

IntVal = TEMPORARY_DB(*Mode*, *VarName*, *Title_1*, *Title_2*, *Min*, *Max, Format*, *Units*);

Argument	Meaning
VarName	The name for the temporary variable. Type STR.
Title_1	The title for the temporary variable in the primary language. Type STR.

Min	The minimum value of the range of the register. Any numeric type.
Max	The maximum value of the range of the register. Any numeric type.
Format	The display format for the register, for example ###.##. Type STR.
Units	The units text for the register, for example DegC. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
3	ADDREG	A temporary register variable is created using the supplied attributes. Return: 1 if successful, 0 indicates a syntax error or the variable already exists.

Syntax 4

IntVal = TEMPORARY_DB(*Mode*, *VarName*, *Title_1*, *Title_2*);

Argument	Meaning
VarName	The name for the temporary variable. Type STR
Title_1	The title for the temporary variable in the primary language. Type STR
Title_2	The title for the temporary variable in the alternative language. Type STR

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
4	ADDTXT	A temporary text variable is created using the supplied attributes. Return: 1 if successful, 0 indicates a syntax error or the variable already exists.

Example

```
DIM ConstVarName As Str;
DIM VarName As Str;
DIM Index As Integer;
ConstVarName = "TempBit"
```

```
'Create 100 bit variables: TempBit0 to TempBit99
For(Index = 0; Index < 100; Index ++)
VarName = AddString(ConstVarName, TOC(Index));
Temporary_DB("ADDBIT", VarName, "", "");
Next
```

For a further example, select the Example link above.

TEXTBOX

See Also Example

Access some properties of the Supervisor's Text-box form control.

WebVue support - No.

Mode	Mnemonic	Syntax
1	GETTEXT	1
2	SETTEXT	2
3	APPENDTEXT	3
4	GETSELECTEDTEXT	1
5	GETLINETEXT	4
6	GETLINECOUNT	5
7	CLEAR	5
8	COPY	5
9	CUT	5
10	PASTE	5
11	SETREADONLY	6
12	SETBACKCOLOR	7
13	SETTEXTCOLOR	7
14	GETTEXTLENGTH	5
15	SELECTTEXT	8
16	INDEXOF	9
17	ENABLEEVENTS	6
18	REPLACE	10

All syntaxes

Argument	Meaning
Window	The name of the window that contains the form control. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR.
Identity	The identity of the form control within the specified window. Type STR.
Text	A text string. Type STR.

Syntax 1

StrVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*);

The return type is STR.

Execution

Mode	Mnemonic	Action
1	GETTEXT	Returns all the text displayed in the text-box. Return: Text if successful, else empty.
4	GETSELECTEDTEXT	Returns all the text that is selected in the text-box. Return: Text if successful, else empty.

Syntax 2

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Text*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
2	SETTEXT	Set the text in the text-box.

Return: 1 if successful, else -1.

Syntax 3

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Text [,NewLine]*);

The return type is INTEGER.

Argument	Meaning
Newline	Flag indicating how the text is to be appended. Optional. Default = 0. 1 = Text added as a new line. 0 = Text added immediately following any existing text (no new line).

Execution

Mode	Mnemonic	Action
3	APPEND	Append the supplied text. Return: 1 if successful, else -1.

Syntax 4

StrVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Line*);

The return type is STR.

Argument	Meaning
Line	1 based index of a line in the text-box. Type INTEGER.

Execution

Mode	Mnemonic	Action
5	GETLINETEXT	Return the text from the given line. Return: Text if successful, else empty.

Syntax 5

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	GETLINECOUNT	Returns the number of lines in the text-box Return: The number of lines if successful, else -1.
7	CLEAR	Clear all text in the text-box. Return: 1 if successful, else -1.
8	COPY	Copies the selected text in the text-box to the clipboard. Return: 1 if successful, else -1.
9	CUT	Cuts the selected text from the text-box and pastes it in the clipboard. Return: 1 if successful, else -1.
10	PASTE	Replaces the selected text in the text-box with the contents of the clipboard. Return: 1 if successful, else -1.
14	GETTEXTLENGTH	Return the text length in the text-box. Return: number of characters if successful, else -1.

Syntax 6

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Flag*);

The return type is INTEGER.

Argument	Meaning
Flag	Flag specifying the operation. Either 0 or 1. 1 = Read only. 0 = Read and write.

Execution

Mode	Mnemonic	Action
11	SETREADONLY	Set the read / write mode of the text-box according to the supplied flag. 1 = Read only. 0 = Read and write. Return: 1 if successful, else -1.
17	ENABLEEVENTS	Enable or disable text-box events (text changes, key pressed) according to the su flag. 1 = Events enabled. 0 = Events disabled. Return: 1 if successful, else -1.

Syntax 7

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Red, Green, Blue*);

The return type is INTEGER.

Argument	Meaning
Red, Green, Blue	Color components in the range 0 to 255. For example 0, 0, 255 is blue. Type INTEGER.

Execution

Mode	Mnemonic	Action
12	SETBACKCOLOR	Set the text-box background color. Return: 1 if successful, else 0.
13	SETTEXTCOLOR	Set the text-box text color. Return: 1 if successful, else 0.

Syntax 8

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity[*, *Start [, Length]]*);

The return type is INTEGER.

Argument	Meaning
Start	Character Position with text of text-box. Type INTEGER.
Length	Number of characters. Type INTEGER.

Execution

Mode	Mnemonic	Action
15	SELECTTEXT	Select text in the text-box using the following criteria. If the Start is omitted then all text is selected. If the Start is given and Length is omitted then text is selected from the Start pos the end. If both Start and length are given the text is selected starting at Start and ending Start + Length. Return: 1 if successful, else -1.

Syntax 9

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *Substring [, Start]*);

The return type is INTEGER.

Argument	Meaning
SubString	Character string. Type STR.
Start	Character Position with text of text-box. Type INTEGER.

Execution

Mode	Mnemonic	Action
16	INDEXOF	Returns the zero-based index of the first occurrence of the SubString in the text-b The search starts at a the Start position. If Start is omitted then the search starts beginning of the text. Return: Position of the first occurrence of the substring if successful, else -1.

Syntax 10

IVal = TEXTBOX(*Mode*, *Window*, *Branch*, *Identity*, *OldString, NewString [, Start[, Length]]*);

The return type is INTEGER.

Argument	Meaning
OldString	Character string. Type STR.
NewString	Character string. Type STR.
Start	Character position with text of text-box. Type INTEGER.
Length	The length of the text in which the search takes place. Type INTEGER.

Execution

Mode	Mnemonic	Action
18	REPLACE	Search the text for instances of OldString and replace it with NewString. The searc starts at the Start position and continues until Start + length is reached. If Start is omitted the search starts at position 0. If Length is omitted the search continues t end of the text. Return: 1 if successful, else -1.

Example

For an example, select the Example link above.

TEXTVAR

See Also Example

Manipulate strings of characters in text variables.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	BUFTOTEXT	1
2	TEXTTOBUF	1
3	FILETOTEXT	2
4	TEXTTOFILE	3
5	TEXTCOMPARE	4
6	TEXTCOPY	5
7	TEXTLEN	6

Syntax 1

IntVal = TEXTVAR(*Mode*, *Text*, *Hbuf*, *Text_Offset*, *Hbuf_Offset*, *Size*);

Argument	Meaning
Text	A string terminated with a binary zero (\0).
Hbuf, Text_Offset, Hbuf_Offset,	See below.
Size	The number of characters to copy, or 0 for all characters.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
1	BUFTOTEXT	Copy the number of characters specified by Size from the buffer specified by its handle *Hbuf* starting at the offset *Hbuf_Offset* to the text variable *Text*, shifted by *Text_Offset*. Return: The number of characters copied by the command.
2	TEXTTOBUF	Copy the number of characters specified by Size from the text variable *Text* starting at the offset *Text_Offset* to the buffer specified by its handle *Hbuf* shifted by *Hbuf_Offset*. Return: The number of characters copied by the command.

Syntax 2

IntVal = TEXTVAR(*Mode*, *Text*, *Filename*, *Text_Offset*, *File_Offset*, *Size*);

Argument	Meaning
Text	A string terminated with a binary zero (\0).
Filename	The name of the source file. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file must exist on that computer.
Text_Offset, File_Offset, Size	See below.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
3	FILETOTEXT	Copy *Size* characters from the file Filename starting at the offset *File_Offset* to the text variable *Text* starting at the offset *Text_Offset*. Return: The number of characters copied by the command.

Syntax 3

IntVal = TEXTVAR(*Mode*, *Text*, *Filename*, *Text_Offset*, *Filemode*, *Size*);

Argument	Meaning
Text	A string terminated with a binary zero (\0).
Filename	The name of the destination file. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the file is created on that computer.
Text_Offset, Size	See below.
FileMode	0 Characters are written starting at the beginning of the file and overwrite the existing contents. 1 The characters are appended to the end of the file.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
4	TEXTTOFILE	Write the number of characters specified by *Size* from the variable *Text*, starting at the offset *Text_Offset*, to the file *Filename*. Return: The number of characters written by the command.

Syntax 4

IntVal = TEXTVAR(*Mode*, *Text1*, *Text2*, *size*);

Argument	Meaning
Text1	The name of a text variable. Type STR.
Text2	The name of another text variable. Type STR.
size	The length of the value of the text variables to compare. Type INTEGER.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
5	TEXTCOMPARE	Compare the string values of the text variables *Text1* and *Text2* for the number of characters specified in *size*.  If the size is invalid, the full strings will be compared. Return: Less than 0 if the extract of *Text1* < the extract of*Text2* 0 if the extract of *Text1*= the extract of *Text2* Greater than 0 if the extract of *Text1* > the extract of *Text2*

Syntax 5

IntVal = TEXTVAR(*Mode*, *Text1*, *Text2*);

Argument	Meaning
Text1	The name of a text variable. Type STR.
Text2	The name of another text variable. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	TEXTCOPY	Copy the variable *Text2* to *Text1*. Return: 1 if OK, else 0.

Syntax 6

IntVal = TEXTVAR(*Mode*, *Text*);

Argument	Meaning
Text	A string terminated with a binary zero (\0).

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
7	TEXTLEN	Determine the length of the text variable. Return: the length in characters of the string *Text*.

Example

This shows how you can pass a variable to a local integer variable (inside the basic function), and then use the local variable as an argument.

```
Dim iSize As Integer;
iSize =toi(@SIZE);
iReturn = Textvar("TEXTCOMPARE","@TEXT1","@TEXT2", iSize );
```

For further examples, select the Example link above.

TOC

See Also

Convert a number to a character string.

WebVue support - Yes.

Syntax

StrVal = TOC(*Num*);

The return type is STR.

Argument	Meaning
Num	The number to be converted. Any numeric type.

Execution

Scientific notation is not preserved.

Example

```
SUB Main()
DIM lngValue as Long;
DIM strString as Str;

lngValue = 125;
strString = TOC( lngValue );
PRINT("Result: ", strString);
'Display "Result: 125"
END SUB
```

TOD

See Also

Convert a number to a DOUBLE value.

WebVue support - Yes.

Syntax

DblVal = TOD(*Num*);

Argument	Meaning
Num	The number to be converted. Any numeric type.

Execution

Return: The number *DblVal* is converted to a DOUBLE value.

Example

```
SUB Main()
DIM lngValue as long;
DIM dblResult as double;

lngValue = 125;
dblResult = TOD( lngValue );
PRINT("Result: ", dblResult);
'Display "Result: 125"
END SUB
```

TODOUBLE

See Also

Convert a variable to a DOUBLE value.

WebVue support - Yes.

By default, SCADA Basic works in float mode, with some slight loss of precision.

Syntax

DblVal = TODOUBLE(*VarName*);

Argument	Meaning
VarName	The variable whose value is to be converted. Type STR.

Execution

Return: The value converted to type DOUBLE.

Example

```
DATETIMESTRING(TODOUBLE("EXTRACT.DATETIMEB"));
```

TOHMS

See Also

Convert a value in seconds to a string in time format.

WebVue support - Yes.

Syntax

StrVal = TOHMS(*Seconds*);

The return type is STR.

Argument	Meaning
Seconds	The number of seconds to be converted. Any numeric type.

Execution

The number of seconds is converted into a time string of the format HHMMSS.

Example

```
SUB Main()
DIM strString1 as STR;
DIM dblValue as DOUBLE;
dblValue = 6594;
strString1 = TOHMS (dblValue);
PRINT("6594 seconds = ",strString1);
'Display "6594 seconds = 01:49:54"
END SUB
```

TOI

See Also

Converts a number to an INTEGER.

WebVue support - Yes.

Syntax

IntVal = TOI(*Num*);

The return type is INTEGER.

Argument	Meaning
Num	The number to be converted. Any numeric type.

Execution

Return: The number *IntVal* is converted to an integer by truncation.

If the number is greater than 2^{31} it becomes negative.

Example

```
SUB Main()
DIM lngValue as Long;
DIM intResult as Integer;

lngValue = 125;
intResult = TOI( lngValue );
PRINT("Result: ", intResult);
'Display "Result: 125"
END SUB
```

TOL

See Also

Convert a number to a LONG value.

WebVue support - Yes.

Syntax

LongVal = TOL(*Num*);

The return type is LONG.

Argument	Meaning
Num	The value to be converted. Any numeric type.

Execution

The number is converted to a LONG value.

Example

```
SUB Main()
DIM lngValue as Long;
DIM lngResult as Long;

lngValue = 125;
lngResult = TOL( lngValue );
PRINT("Result: ", lngResult);
'Display "Result: 125"
END SUB
```

TOLL

See Also

Convert a number to a LONGLONG value.

WebVue support - Yes.

Syntax

LongLongVal = TOLL(*Num*);

The return type is LONGLONG.

Argument	Meaning
Num	The value to be converted. Any numeric type.

Execution

The number is converted to a LONGLONG value.

Example

```
SUB Main()
DIM lngValue as Long;
DIM lnglngResult as LongLong;

lngValue = 125;
lngResult = TOLL( lngValue );
PRINT("Result: ", lnglngResult);
'Display "Result: 125"
END SUB
```

TOS

See Also

Convert a number to a SINGLE value.

WebVue support - Yes.

Syntax

SngVal = TOS(*Num*);

The return type is SINGLE.

Argument	Meaning
Num	The number to be converted. Any numeric type.

Execution

The number *SngVal* is converted to a SINGLE value.

TOS is equivalent to TOD if the property Use register variables as double in SCADA BASIC script has been selected (Configure.Application Explorer.Settings.Advanced.Programs).

This avoids a problem with precision when using register variables of type DOUBLE.

Example

```
SUB Main()
DIM lngValue as Long;
DIM sngResult as Single;

lngValue = 125;
sngResult = TOS( lngValue );
PRINT("Result: ", sngResult);
'Display "Result: 125"
END SUB
```

TRACE

See Also

To generate a trace in SCADA BASIC and output as part of the Supervisor's event log.
WebVue support - Yes.

Syntax

TRACE(*Mode*, *String*);

There is no return type.

Argument	Meaning
String	The message to be logged. Type STR.

Execution

Mode	Mnemonic	Action
	LOG	To send a message to the Event Viewer and TRACE.DAT file. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the trace is generated on that computer.
	FILE	To send a message to the T file. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the trace is generated on that computer.

TRACEON/TRACEOFF

See Also

Enable tracing of function calls.

WebVue support - Yes.

Syntax

TRACEON;

or

TRACEOFF;

There is no return type.

Execution

Trace allows the name of the calling program, the name of the called program, and the name of the called function to be known. The trace appears in the results area of the program dialog box.

The TRACEON instruction is active for all functions of the same program until the TRACEOFF instruction is encountered.

If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the output is created on that computer.

Example

```
SUB Main()
DIM intResult as Integer;

TRACEON;
procedure1();
procedure2("Parameter1");
intResult = procedure4();
PRINT("Return from procedure = ",intResult);
TRACEOFF;
END SUB
```

Any nested function calls will also be traced.

TREE

See Also

Select a branch of the database.

WebVue support - Yes.

Syntax

TREE(*Branch*);

There is no return type.

Argument	Meaning
Branch	The name of the branch to be used. Type STR.

Execution

The TREE instruction is effective, inside a function, until the next TREE instruction.

Example

```
SUB Main()
'Declare variables
DIM strBranche as Str;
DIM strTree as Str;

strBranche = "BRANCH01";
TREE (strBranch);
strTree = GETTREE();
PRINT("Branch: ",strTree);
propagationGetTree();
END SUB

SUB propagationGetTree()
DIM strTree as Str;

strTree = GETTREE();
PRINT("Branch: ",strTree);
END SUB
```

TREEVIEW

See Also Example

Access some properties of the Supervisor's Tree View form control.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	WebVue
1	COUNT	1	Yes
2	GETSELECTEDINDEX	1	Yes
3	SETSELECTEDINDEX	2	Yes
4	GETTEXT	3	Yes
5	GETUSERDATA	3	Yes
6	GETROOT	1	Yes
7	GETCHILDNODE	3	Yes
8	GETNEXTNODE	3	Yes
9	GETPREVNODE	3	Yes
10	GETPARENTNODE	3	Yes
11	EXPANDNODE	3	Yes
12	COLLAPSENODE	3	Yes
13	LOAD	4	Yes
14	SETUSERDATA	5	Yes
15	SETTEXT	5	Yes
16	CLEAR	6	No
17	INSERTNODE	7	No
18	REMOVENODE	3	No

All syntaxes

Argument	Meaning
Window	The name of the window that contains the form control. Type STR.
Branch	The branch (if any) of the window. Use "*" to indicate the current branch of the program. Type STR.
Identity	The identity of the form control within the specified window. Type STR.

Syntax 1

LongVal = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*);

Execution

Mode	Mnemonic	Action
1	COUNT	Returns the number of nodes. Type LONG.
2	GETSELECTEDINDEX	Returns the index of the node that is currently selected. Type LONG.
6	GETROOT	Returns the index of the first root node. Type LONG.

Syntax 2

LongVal = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*, *NodeId*, *Notification*);

Argument	Meaning
NodeId	The index of a node. Type LONG. If the index is -1 then no item is selected.
Notification	To indicate whether selection triggers the execution of the SCADA BASIC function defined in the Operations configuration of the form control. Type INTEGER.

For this to work, it also has to be enabled in the Operations tab of the Tree View properties dialog.

The function will scroll the control if necessary to display the item.

Execution

Mode	Mnemonic	Action
3	SETSELECTEDINDEX	Select a node according to its index. Returns a value that indicates whether the action succeeded (1) or failed (0). Type LONG.

Syntax 3

Return = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*, *NodeId*);

Argument	Meaning
NodeId	The index of a node. Type LONG.

Execution

Mode	Mnemonic	Action
4	GETTEXT	Returns the text of the node as defined for the language currently in use. Type STR.
5	GETUSERDATA	Returns the user data associated with the node. Type STR.
7	GETCHILDNODE	Returns the index of the first child node of a node. Type LONG.
8	GETNEXTNODE	Returns the index of the next sibling node. Type LONG.
9	GETPREVNODE	Returns the index of the previous sibling node. Type LONG.
10	GETPARENTNODE	Returns the index of the parent node. Type LONG.
11	EXPANDNODE	Expands the node. Returns a value that indicates whether the action succeeded (1) or failed (0). Type LONG.
12	COLLAPSENODE	Collapses the node. Returns a value that indicates whether the action succeeded (1) or failed (0). Type LONG.
18	REMOVENODE	Removes the selected node. Returns a value that indicates whether the action succeeded (1) or failed (0). Type LONG.

Syntax 4

IntVal = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*, *FileName*);

Argument	Meaning
FileName	The name of the file that contains the form control's data.

Execution

Mode	Mnemonic	Action
13	LOAD	Loads the file's content into the form control's list of items. Returns 0 if not loaded, 1 if loaded. Type INTEGER.

Syntax 5

LongVal = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*, *NodeID, Text*);

Argument	Meaning
NodeID	The index of a node. Type LONG.
Text	A text string. Type STR.

Execution

Mode	Mnemonic	Action
14	SETUSERDATA	Sets the node's user data using the supplied text string. Returns a value that indicates whether the action succeeded (1) or failed (0).
15	SETTEXT	Sets the node's text using the supplied text string. Returns a value that indicates whether the action succeeded (1) or failed (0).

Syntax 6

IntVal = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*);

Execution

Mode	Mnemonic	Action
16	CLEAR	Clears the entire contents of the tree view form control. Returns a value that indicates whether the action succeeded (1) or failed (0).

Syntax 7

LongVal = TREEVIEW(*Mode*, *Window*, *Branch*, *Identity*, *NodeID, Text*, *UserData*);

Argument	Meaning
NodeID	The index of a node. Type LONG.
Text	A text string. Type STR.
UserData	A text string. Type STR.

Execution

Mode	Mnemonic	Action
17	INSERTNODE	Insert a new node using the supplied parameters. Returns a value that indicates whether the action succeeded (1) or failed (0).

Example

For an example, select the Example link above.

TREND

See Also Example

Modify the operation of a Trend Viewer.

Partial WebVue support - see mode table.

If using a project with multiple regions, you must set the region before executing any instructions that interacts with the HMI. For further information see the REGION topic.

Mode	Mnemonic	Syntax	WebVue
1	SETTYPE	1	Yes
2	GETTYPE	2	Yes
3	SETYRANGE	3	Yes
4	GETYMIN	4	No
5	GETYMAX	4	No
6	SETPERIOD	5	No
7	GETPERIOD	6	No
8	SETDATETIME	7	Yes
9	SETDATETIMEPERIOD	8	No
10	SCROLLPERCENT	9	No
11	SCROLLTIME	10	No
12	GETDATETIME	11	No
13	GETCURSORVALUE	4	No
14	SETVAR	12	Yes
15	GETVAR	13	No
16	ADDVARLIST	14	No
17	SETVARLIST	15	No
18	SETVARBUF	16	No
19	RESETVAR	17	No
20	CLEARVAR	17	Yes
21	CHANGEID	18	No
22	SETTIMECAPACITY	5	No
23	SETCOLOR	19	Yes
24	SETSTYLE	1	Yes
25	HARDCOPY	20	No
26	GETSTYLE	22	No
27	SETYMIN	21	Yes
28	SETYMAX	21	Yes
29	GETCOLORRED	22	No
30	GETCOLORGREEN	22	No
31	GETCOLORBLUE	22	No
32	REFRESH	20	No
33	THRESHOLD_SETMODE	23	No
35	THRESHOLD_SETPROP	24	No
36	THRESHOLD_SETVALUE	25	No
37	THRESHOLD_DRAWLINE	26	No
38	DRAW_MINLINE	27	No
39	DRAW_MAXLINE	27	No
40	DISPLAY_MINMAXLINE	30	No
41	SETTIMEORIGIN	28	No
42	SETTIMEOFFSET	29	No
43	EXPORT	20	No
44	SET_VISIBLE	31	No
45	DISPLAY_SCALE	31	No

All syntaxes

Argument	Meaning

Window	The name of the window which contains the Trend Viewer which is to be used. Type STR.
Branch	The branch of the window (if any). Using a "*" means the current branch of the program. Type STR.
Identity	The identity of the Trend Viewer animation within the specified window. Type STR.

Syntax 1

IntVal = TREND(*Mode*, *Window*, *Branch*, *Identity*, *Trace*, *SubMode*);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace to modify. Type INTEGER, range 0 to 16.
SubMode	The *Mode* of operation for the selected trace. Type INTEGER.

Execution

Mode	Mnemonic	Action
1	SETTYPE	Change all traces to historic (*SubMode*= 2) or real-time (*SubMode* = 1). The Trace parameter is not taken into account and should be set to 0.
24	SETSTYLE	Change the style of the selected trace according to the *SubMode*: 0 A square wave. 1 A triangular wave. 2 A series of dots. Return: 1 if successful, else 0.

When using Mode 1 SETTYPE, the *Trace* parameter must still be included but it is ignored. All traces are affected.

Syntax 2

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*);

The return type is INTEGER.

Argument	Meaning
Trace	Unused. Always 0.

Execution

Mode	Mnemonic	Action
2	GETTYPE	Return the *Mode* of operation for all traces. The Trace parameter is unused and always set to 0. Return: 1 for real time 2 for historic 0 if there is an error.

Syntax 3

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace* [, *Ymin*, *YMax*]);

The return type is INTEGER.

Argument	Meaning

Trace	The number of the trace. Type INTEGER, range 0 to 16.
Ymin, YMax	The range for the selected trace. Type DOUBLE.

Execution

Mode	Mnemonic	Action
3	SETYRANGE	The Y axis range for the selected trace is changed. If the Ymin and Ymax are omitted the range is reset to that of the trend viewer's original configuration. If Trace is 0 then all traces are affected. If Trace is between 1 and 16 then the corresponding trace is affected. Return: 1 if successful, else 0.

Syntax 4

DblVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*);

The return type is DOUBLE.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.

Execution

Mode	Mnemonic	Action
4	GETYMIN	Return the Y axis minimum value for the selected trace.
5	GETYMAX	Return the Y axis maximum value for the selected trace.
13	GETCURSORVALUE	Return the value of the selected trace at the current cursor position.

Syntax 5

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Duration*);

The return type is INTEGER.

Argument	Meaning
Duration	A time period in expressed in milliseconds. Type DOUBLE.

Execution

Mode	Mnemonic	Action
6	SETPERIOD	Change the X axis period for the Trend Viewer.
22	SETTIMECAPACITY	Change the period of the Trend Viewer internal buffer. Return: 1 if successful, else 0.

Syntax 6

DblVal = TREND(*Mode*, *Window, Branch*, *Identity*);

The return type is DOUBLE.

Execution

Mode	Mnemonic	Action
7	GETPERIOD	Return the period of the X axis expressed in milliseconds.

Syntax 7

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *StartTime*, *EndTime*]);

The return type is INTEGER.

Argument	Meaning
StartTime	The earliest date and time, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE.
EndTime	The latest date and time, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue. Type DOUBLE.

Execution

Mode	Mnemonic	Action
8	SETDATETIME	Modify the start and end time of the Trend Viewer. If the end time is omitted, only the start time is changed and the period will remain the same. Return: 1 if successful, else 0.

Syntax 8

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *StartTime*, *Duration*);

The return type is INTEGER.

Argument	Meaning
StartTime	The earliest date and time, expressed as the number of milliseconds since 1980. See the instruction DateTimeValue.Type DOUBLE.
Duration	A time period expressed in milliseconds. Type DOUBLE.

Execution

Mode	Mnemonic	Action
9	SETDATETIMEPERIOD	Modify the start time and period for the Trend Viewer. Result: 1 if successful, else 0.

Syntax 9

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *PScroll*);

The return type is INTEGER.

Argument	Meaning
Pscroll	The percentage by which the trend X axis will be scrolled. Type INTEGER, range -100 to +100.

Execution

Mode	Mnemonic	Action
10	SCROLLPERCENT	The X axis of the trend is scrolled by a percentage of its current period. A negative value will cause the trend to scroll back in time, a positive value forward. Return: 1 if successful, else 0.

Syntax 10

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *TScroll*);

The return type is INTEGER.

Argument	Meaning

Tscroll	The time period by which the trend X axis will be scrolled expressed in milliseconds. Type DOUBLE.

Execution

Mode	Mnemonic	Action
11	SCROLLTIME	The X axis of the trend is scrolled by the supplied period. A negative value will cause the trend to scroll back in time, a positive value forward. Return: 1 if successful, else 0.

Syntax 11

DblVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*, *Function*);

The return type is DOUBLE.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
Function	A flag indicating which operation to perform. Type INTEGER, range 1 to 3.

Execution

Mode	Mnemonic	Action
12	GETDATETIME	Return the time for the trend expressed as milliseconds since 1980, according to *Function*: 1 Start time 2 End time 3 Cursor time

Syntax 12

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *ClearFlag*, *Trace*, *VarName* [,*Label*[*Type*]]);

The return type is INTEGER.

Argument	Meaning
ClearFlag	A flag indicating how to treat the current trace. Type INTEGER, range 0 or 1.
Trace	The number of the trace. Type INTEGER, range 1 to 16.
VarName	The name of the variable to display. Type STR.
Label	The label that is displayed on the trend left border. Type STR.
Type	A flag indicating the *Mode*of operation of the trace. Type INTEGER, range 0 to 2.

Execution

Mode	Mnemonic	Action
14	SETVAR	Change the name of the variable for the specified trace, according to *ClearFlag*: 0 The current trace is preserved 1 The trace is erased The mode of operation is set by the *Type*argument: 0 leaves it unchanged 1 forces a real-time trace 2 forces a historic trace Return: 1 if successful, else 0.

Syntax 13

StrVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*);

The return type is STR.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.

Execution

Mode	Mnemonic	Action
15	GETVAR	Return the name of the variable for the trace.

Syntax 14

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*, *Varname*[,*Label* [*Type*]]);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
VarName	The name of the variable to display. Type STR.
Label	The label that is displayed on the trend left border. Type STR.
Type	A flag indicating the *Mode*of operation of the trace. Type INTEGER, range 0 to 2.

Execution

Mode	Mnemonic	Action
16	ADDVARLIST	Sets up a change in variable name for the specified trace. The change only takes place when followed by a mode 17 instruction. A number of mode 16 instructions are usually followed by a mode 17 instruction to enable a number of traces on the same Trend display to be changed simultaneously. The mode of operation is set by the *Type* argument: 0 leaves it unchanged 1 forces a real-time trace 2 forces a historic trace Return: 1 if successful, else 0.

Syntax 15

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *ClearFlag*)

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
17	SETVARLIST	Used with *Mode*16 to change the variable for a number of traces on the same Trend Viewer simultaneously. The current traces are preserved or erased according to *ClearFlag*: 0 preserved 1 erased
	Return	1 if successful, else 0.

Syntax 16

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *ClearFlag*, *Handle*)

Argument	Meaning
Handle	The location of a memory buffer as created by the instruction FILETOBUF. Type LONG.

The return type is INTEGER.

Argument	Meaning
ClearFlag	See below.

Execution

Mode	Mnemonic	Action
18	SETVARBUF	Used with a memory buffer to simultaneously change a number of traces on the same Trend Viewer, according to *ClearFlag*: 0 The current traces are preserved 1 The traces are erased The syntax for each line of the memory buffer must be as follows: *Trace*, *VarName*, *Label*, *Type* Return: 1 if successful, else 0.

Syntax 17

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*[,*Trace*]);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.

Execution

Mode	Mnemonic	Action
19	RESETVAR	The specified trace is reset to its original configuration. If the trace is not specified than all traces will be reset. Return: 1 if successful, else 0.
20	CLEARVAR	The specified trace is erased. If the trace is not specified than all traces will be erased. Return: 1 if successful, else 0.

Syntax 18

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *NewIdentity*);

Argument	Meaning
NewIdentity	A new identity for the Trend Viewer animation. Type STR.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
21	CHANGEID	Allows the identity of the trend to be changed. Return: 1 if successful, else 0.

Syntax 19

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*, *Red*, *Green*, *Blue*);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
Red	The red component of the trace color. Type INTEGER, range 0 to 255.
Green	The green component of the trace color. Type INTEGER, range 0 to 255.
Blue	The blue component of the trace color. Type INTEGER, range 0 to 255.

Execution

Mode	Mnemonic	Action
23	SETCOLOR	Change the color of the selected trace according to the supplied RGB components. Return: 1 if successful, else 0.

In general where a function operates on a single trace, the trace is identified by a number in the range 1 to 16. However if the trace number is 0 or omitted then all traces of the trend will be affected.

The time parameters are of type DOUBLE, representing the number of milliseconds from the 1st January 1980.

Syntax 20

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
25	HARDCOPY	Make a (proportional) hard copy of the Trend Viewer window.
32	REFRESH	Re-requests historical data and refreshes the display.
42	EXPORT	Initiate a Trend export. Has the same effect as using the Export button of a Trend Viewer. Return: 1 if successful, else 0.

The effect of HARDCOPY is similar to that of the Print button on the trend toolbar.

Example

For an example, select the Example link above.

Syntax 21

IntVal = TREND(*Mode*, *Window, Branch*, *Identity,Trace, dbY*);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
dbY	The minimum or maximum value of the scale for a trace.

Execution

Mode	Mnemonic	Action
27	SETYMIN	Sets the minimum value for the scale of a trace. Return: 1 if successful, else 0.
28	SETYMAX	Sets the maximum value for the scale of a trace. Return: 1 if successful, else 0.

If both *Min*and *Max* are 0, the variable's *Min* and *Max* values are used instead.

Syntax 22

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*,*Trace*);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.

Execution

Mode	Mnemonic	Action
26	GETSTYLE	Gives the type of trace. Return: -1 Error. 0 A square wave. 1 A triangular wave. 2 A series of dots.
29	GETCOLORRED	Return: -1 Error. 0 to 255 The red component of the trace's color.
30	GETCOLORGREEN	Return: -1 Error. 0 to 255 The green component of the trace's color.
31	GETCOLORBLUE	Return: -1 Error. 0 to 255 The blue component of the trace's color.

Syntax 23

IntVal = TREND(Mode, Window, Branch, Identity, Trace, ThresholdMode[, ThresholdValue1, DisplayThreshold1[, ThresholdValue2, DisplayThreshold2[,ThresholdValue3, DisplayThreshold3[,ThresholdValue4, DisplayThreshold4]]]]);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
ThresholdMode	The new threshold mode. Type INTEGER, range -1 to 5. See table below.
ThresholdValueN	The new value for threshold N. Any numeric type.
DisplayThresholdN	Display mode for threshold N. Type INTEGER. 0 - New threshold not displayed. 1 - New threshold displayed. 2 - No change.

Execution

Mode	Mnemonic	Action
33	THRESHOLD_SETMODE	Change the configuration of the threshold display for the selected trace. Return: 1 if successful, else 0.

Threshold mode

Mode	Threshold system
-1	None
0	From variable configuration
1	ppphigh/pphigh/hihi/high
2	pphigh/hihi/high/low
3	hihi/high/low/lolo
4	high/low/lolo/pplow
5	low/lolo/pplow/ppplow

Syntax 24

IntVal = Trend(Mode, Window, Branch, Identity, Trace , ThresholdNumber, Red, Green, Blue, PenStyle, DrawThresholdLine);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
ThresholdNumber	A number identifying the threshold. Type INTEGER, range 0 to 3.
Red	The red component of the threshold color. Type INTEGER, range -1 to 255.
Blue	The blue component of the threshold color. Type INTEGER, range -1 to 255.
Green	The green component of the threshold color. Type INTEGER, range -1 to 255.
PenStyle	The style of the threshold line. See table below
DrawThresholdLine	A flag indicating if the threshold line is to be drawn. Type INTEGER. 0 hides the line, 1 draws the line.

Execution

Mode	Mnemonic	Action
35	THRESHOLD_SETPROP	Change the properties of the selected threshold. The color is that of the trace when the threshold is passed. If -1 is used for each of the color components the color does not change. Return: 1 if successful, else 0.

Pen style

Value	Style
0	Solid
1	Dash
2	Dot
3	Dash Dot
4	Dash Dot Dot

Syntax 25

IntVal = Trend(Mode, Window, Branch, Identity, Trace , ThresholdNumber, ThresholdValue);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
ThresholdNumber	A number identifying the threshold. Type INTEGER, range 0 to 3.
ThresholdValue	The new value for the selected threshold. Any numeric type.

Execution

Mode	Mnemonic	Action
36	THRESHOLD_SETVALUE	Change the value of the selected threshold. Return: 1 if successful, else 0.

Syntax 26

IntVal = Trend(Mode, Window, Branch, Identity, Trace , ThresholdNumber, DrawThresholdLine);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
ThresholdNumber	A number identifying the threshold. Type INTEGER, range -1 to 3. A value of -1 means all thresholds.
DrawThresholdLine	A flag indicating if the threshold line is to be drawn. Type INTEGER. 0 hides the line, 1 draws the line

Execution

Mode	Mnemonic	Action
37	THRESHOLD_DRAWLINE	Draw or hide the selected threshold(s). Return: 1 if successful, else 0.

Syntax 27

IntVal = Trend(Mode, Window, Branch, Identity, Trace , Red, Green, Blue, PenStyle);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
Red	The red component of the trace color. Type INTEGER, range -1 to 255.
Blue	The blue component of the trace color. Type INTEGER, range -1 to 255.
Green	The green component of the trace color. Type INTEGER, range -1 to 255.
PenStyle	The style of the threshold line. See table in Syntax 24.

Execution

Mode	Mnemonic	Action
38	DRAW_MINLINE	Display the variable's minimum value as a line. The color is that of the trace when the line is crossed. If -1 is used for each of the color components the color does not change. Return: 1 if successful, else 0.
39	DRAW_MAXLINE	Display the variable's maximum value as a line. The color is that of the trace when the line is crossed. If -1 is used for each of the color components the color does not change. Return: 1 if successful, else 0.

Syntax 28

IntVal = Trend (MODE, Window, Branch, Identity, Trace, TimeOrigin);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
TimeOrigin	The time of the trace origin (left side of the trace). Type DOUBLE as produced by DATETIMEVALUE..

Execution

Mode	Mnemonic	Action
41	SETTIMEORIGIN	Set the origin (left side) of the trace to a specific time. Return: 1 if successful, else 0.

Syntax 29

IntVal = Trend (MODE, Window, Branch, Identity, Trace, Time, TimeUnit);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
Time	A period as a number of TimeUnit units. Type INTEGER.
TimeUnit	A number representing the time unit to use. See table below. Type INTEGER.

Execution

Mode	Mnemonic	Action
42	SETTIMEOFFSET	Offset the origin (left side) of the trace using the given values. Return: 1 if successful, else 0.

Time unit

Value	Unit
0	Cancel the offset.
1	Seconds
2	Minutes
3	Hours
4	Days

Syntax 30

IntVal = Trend (MODE, Window, Branch, Identity, Trace, MinMax [, Showline);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
MinMax	Paramater indicarting which line to affect. Type INTEGER. -1 = both lines 0 = maximum line 1 = minimum line
Showline	A flag to select hide or display the line(s). 0 = Hide 1 = Show

Execution

Mode	Mnemonic	Action
40	DISPLAY_MINMAXLINE	Show/hide the variable minimum or maximum horizontal lines Return: 1 if successful, else 0.

Syntax 31

IntVal = TREND(*Mode*, *Window, Branch*, *Identity*, *Trace*, *Flag*);

The return type is INTEGER.

Argument	Meaning
Trace	The number of the trace. Type INTEGER, range 1 to 16.
Flag	A flag indicating which operation to perform. 0 or 1.

Execution

Mode	Mnemonic	Action
44	SET_VISIBLE	Hide or display a trace according to the value of the flag. 0 = invisible, 1 = visible. Return: 1 if successful, else 0.
45	DISPLAY_SCALE	Hide or display the trace scale according to the value of the flag. 0 = invisible, 1 = visible. Return: 1 if successful, else 0.

Example

For an example, select the Example link above.

U

UCASE

See Also

Convert all the characters of a string to upper case.

WebVue support - Yes.

Syntax

StrVal = UCASE(*Input*);

The return type is STR.

Argument	Meaning
Input	The string to be converted. Type STR.

Example

```
SUB Main()
DIM strResult as Str;
DIM strString as Str;

strString = "string in lower case";
strResult = UCase( strString );
PRINT("Result: ", strResult );
'Display "Result: STRING IN LOWER CASE"
END SUB
```

UNLINK

See Also

Delete a file.

WebVue support - Yes.

Syntax

IntVal = UNLINK(*Filename*);

The return type is INTEGER.

Argument	Meaning
Filename	The name of the file to be deleted. Type STR. If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.

If executed on WebVue the file name is with respect to the WebVue back end.

Execution

The file must be closed before it can be deleted.

Return: 1 if successful, else 0.

Example

```
'For this. there must be a file "file1.txt" in
' the TP folder of your project
SUB Main()
IF (UNLINK("file1.txt")==1) THEN
  PRINT ("file1.txt has been deleted");
END IF
END SUB
```

v

VARIABLE

See Also Example Further information

Management of variables.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	STATUS	1
2	MASK	2
3	UNMASK	2
4	ENABLE	3
5	DISABLE	3
6	LONGLABEL	4
7	ASSOCLABEL	4
8	SIMU	5
9	DOMAIN	4
10	NATURE	4
11	UNIT	4
12	NUMBER	6
13	THRESHOLD_GETTYPE	3
14	THRESHOLD_GETVALUE	7
15	THRESHOLD_SETVALUE	8
22	BATT	9
23	TATT	10
24	WRITE	11
25	LOCKTONODE	12
26	UNLOCKTONODE	12
27	FLOWPARAMTONODE	13
28	IMPORTBUFFER	14
29	IMPORTFILE	15
30	STARTWATCHLIST	16
31	STOPWATCHLIST	17
32	GET_LONG_IN_DB	18
33	GET_DOUBLE_IN_DB	19
34	GET_PHYSICAL_MIN	19
35	GET_PHYSICAL_MAX	19
36	GET_CONTROL_MIN	19
37	GET_CONTROL_MAX	19
38	SETBATT	20
39	SETTATT	21
40	SAVE	22
41	SETLONGLABEL	25
42	GET_ALARM_PRIORITY	3
43	GET_COMMAND_LEVEL	3

44	GET_CONTROL_LEVEL	3
45	GET_TEXT_LEVEL	3
46	GET_TYPE	3
53	GET_DEADBAND_TYPE	3
54	GET_DEADBAND_VALUE	23
55	SET_DEADBAND	24
56	TREND	3
57	GETSERVERSLIST	3
60	ADD_HMIBIT	26
61	ADD_HMIREG	27
62	ADD_HMITXT	28
63	REMOVE_HMIVAR	29

Syntax 1

IntVal = VARIABLE (*Mode*, *Varname*, *Property*);

The return type is INTEGER.

Argument	Meaning
Varname	The name of a variable. Type STR.
Property	The property to be tested: 1 EXIST 2 VALID 3 MASK 4 ENABLE 5 COM 6 TS_TYPE 7 TEMPORARY_DB 8 SIMU

Execution

Mode	Mnemonic	Action
1	STATUS	Test for the specified property of the variable: EXIST Returns 1 if the variable exists, else 0. VALID Returns 1 if the variable has a valid value, else 0. MASK Return the mask of a variable represented by a binary weight: 1 User program 1 2 User program 2 4 User program 3 8 User program 4 16 Operator 32 Dependent (on another variable) 128 Expression ENABLE Returns 1 if the variable is inhibited, else 0. COM Returns 1 if the variable is invalid due to a communication fault, else 0. TS_TYPE Return the type of time stamp for a variable: 0 Valid timestamp set by the SCADA'. 1 Valid timestamp provided by the device. 2 Uncertain timestamp provided by the device (clock failure, device not synchronized….). TEMPORARY_DB: Return 1 if the variable is temporary, else 0.

SIMU: Returns 1 if the variable is simulated, else 0.

Syntax 2

IntVal = VARIABLE (*Mode*, *Variable_name*, *MaskLevel*);

The return type is INTEGER.

Argument	Action
Variable_name	Name of the alarm variable whose level is to be changed.
MaskLevel	A sixteen bit binary mask. 1 Masked by User program 1 2 Masked by User program 2 4 Masked by User program 3 8 Masked by User program 4 16 Masked by Operator 32 Masked by dependency (on another variable) 64 Masked by expression

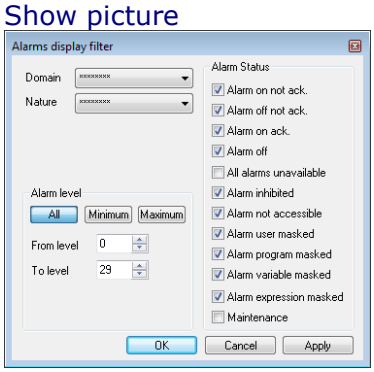
Execution

Mode	Mnemonic	Action
2	MASK	Set the mask of the variable using the given mask level(s).
3	UNMASK	Clear the mask of the variable using the given mask level(s). Return: 1 if successful, else 0.

In the Alarm or Log Viewer the behavior is as follows.

- To display an alarm masked using any combination of User program 1, User program 2, User program 3 and User program 4 the filter Alarm program masked must be ticked.
- To display an alarm masked using Operator the filter Alarm user masked must be ticked.
- To display an alarm masked using Dependent the filter Alarm variable masked must be ticked.
- To display an alarm masked using Expression the filter Alarm expression masked must be ticked.

Show picture



You can set or clear a combination of masks together by using the binary weight. For example 9 would set the 1st and 4th mask bits.

Syntax 3

IntVal = VARIABLE (*Mode*, *VarName*);

The return type is INTEGER.

Argument	Action

VarName	Name of a variable. Type STR.

Execution

Mode	Mnemonic	Action
4	ENABLE	Enable the variable.
5	DISABLE	Disable the variable.
13	THRESHOLD_GETTYPE	Return the threshold system of the variable: 0 - No thresholds. 1 - ppppigh / pphigh / hihi / high 2 - pphigh / hihi / high / low 3 - hihi / high / low / lolo 4 - high / low / lolo / pplow 5 - low / lolo / pplow / ppplow
42	GET_ALARM_PRIORITY	Return the alarm priority of the variable.
43	GET_COMMAND_LEVEL	Return the command level of a bit variable.
44	GET_CONTROL_LEVEL	Return the control level of a register variable.
45	GET_TEXT_LEVEL	Return the control level of a text variable.
46	GET_TYPE	Return the variable type: 1 Bit 2 Register 4 Text 8 Alarm
53	GET_DEADBAND_TYPE	Return the deadband type: -2 The variable does not exist or is not a register. -1 The variable is a temporary variable. 0 The deadband is type absolute. 1 The deadband is a percentage of the range. 2 The deadband is a percentage of the value.
56	TREND	Checks whether the variable is a trend variable. Returns 1 = yes or 0 = no.
57	GETSERVERSLIST	Returns the value for the list of producer stations.

Syntax 4

StrVal = VARIABLE(*Mode*, *VarName*);

The return type is STR.

Argument	Action
VarName	Name of a variable. Type STR.

Execution

Mode	Mnemonic	Action
6	LONGLABEL	Return the long label (title) of the variable.
7	ASSOCLABEL	Return the associated label of a bit variable, corresponding to its current state.
9	DOMAIN	Return the variable's domain.
10	NATURE	Return the variable's nature.
11	UNIT	Return the variable's units text. (Register only)

Syntax 5

IntVal = VARIABLE(*Mode*, *VarName*, *Flag*);

The return type is INTEGER.

Argument	Meaning
VarName	Name of a variable. Type STR.
Flag	A flag indicating the required operation.

Execution

Mode	Mnemonic	Action
8	SIMU	Enable and disable the simulation mode for an equipment variable: 0 for OFF 1 for ON When the simulation mode is ON the variable may be forced internally. Return: 1 if successful, else 0.

Syntax 6

DblVal = VARIABLE(*Mode*);

The return type is LONG.

Execution

Mode	Mnemonic	Action
12	NUMBER	Return the total number of variables in the Variables Tree.

Syntax 7

DblVal = VARIABLE(*Mode*, *VarName, Rank*);

Argument	Meaning
VarName	The name of a variable within the Variables Tree. Type STR.
Rank	A number between 0 and 3 specifying the rank of the threshold (1st, 2nd etc.). Type INTEGER.

The return type is DOUBLE.

Execution

Mode	Mnemonic	Action
14	THRESHOLD_GETVALUE	Return the value of the threshold specified by its rank.

Syntax 8

IntVal = VARIABLE(*Mode*, *VarName, Rank*, *Value*[, *Rank*, *Value*] [*, Rank, Value*] [*, Rank, Value*][, *Saveflag*]);

Argument	Meaning
VarName	The name of a variable within the Variables Tree. Type STR.
Rank	A number between 0 and 3 specifying the threshold (1st, 2ndetc.). Type INTEGER. Any thresholds that are omitted remain unchanged.
Value	A value for the threshold.
Saveflag	When set to 1 indicates that the threshold change should be permanent and saved when the software is shut down.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
15	THRESHOLD_SETVALUE	Assign a new value to the specified threshold. The value given is checked: With the maximum and minimum value of the variable. To see whether it fits into the sequence of thresholds (for example a High threshold must be higher than a corresponding Low threshold). Return: 1 if successful, else 0 (out of range etc.)

Syntax 9

IntVal = VARIABLE(*Mode*, *VarName, BattNum*);

Argument	Meaning
VarName	The name of a variable within the Variables Tree. Type STR.
BattNum	The rank of a binary attribute. Type INTEGER. Range 1 to 30

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
22	BATT	Return the value of the binary attribute specified by the value *BattNum*.

Syntax 10

StrVal = VARIABLE(*Mode*, *VarName, TattNum*);

Argument	Meaning
VarName	The name of a variable within the Variables Tree. Type STR.
TattNum	The number of a text attribute. Type INTEGER. Range 1 to 16.

The return type is STR.

Execution

Mode	Mnemonic	Action
23	TATT	Return the specified text attribute: 1 - domain 2 - nature

Syntax 11

IntVal = VARIABLE(*Mode*, *SubMode*);

Argument	Meaning
SubMode	A code indicating the action to be taken: INTEGER values 0, 1, 2 or 3: 0 Inhibit forcing of variables not produced by the local station ('external'). 1 Authorize forcing of variables not produced by the local station. (Default for all versions) 2 Send a request to write even if the value remains unchanged. (Default before version 10) 3 Send a request to write only if the value has changed in relation to the real time value(Default from version 10)

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
24	WRITE	Handling of writing requests on the local station. Return: 1 if successful, else 0.

There are always two default values, 1 and 2 for software versions before 10.0 and values 1 and 3 for version 10.0 onwards.

Sub-modes 0 and 1 allow the Supervisor, for example, to inhibit or authorize command sending from passive, redundant stations or a client station.
Sub-modes 2 and 3 are applicable to all variables (Internal, Equipment, OPC, BACnet etc.).
Sub-mode type 3 optimizes writing requests.

Sub-mode type 3 does not apply to recipes.

Syntax 12

IntVal = VARIABLE(*Mode*, *StationNo*);

The return type is INTEGER.

Argument	Meaning
StationNo	A number representing the station number of a Supervisor. Type INTEGER.

Execution

Mode	Mnemonic	Action
25	LOCKTONODE	Locks (disables) change of value messages to the specified station. This is used on multi-station architecture projects using dial-up networking. Return: 1 if successful, else 0.
26	UNLOCKTONODE	Unlocks (enables) change of value messages to the specified station. This is used on multi-station architecture projects using dial-up networking. Return: 1 if successful, else 0.

Syntax 13

IntVal = VARIABLE(*Mode*, *StationNo*, *StopLimit*, *StartLimit*, *Filter*);

The return type is INTEGER.

Argument	Meaning
StationNo	A number representing the station number of a Supervisor. Type INTEGER.
StopLimit	The number of variables in the LAN manager mailbox for the station *StationNo* at which network flow control stops. Type INTEGER.
StartLimit	The number of variables in the LAN manager mailbox for the station *StationNo* at which network flow control starts. Type INTEGER.
Filter	A number representing the variable types to be affected by the flow control. Each variable type is represented by a binary weight. Type INTEGER. 1 For the regulation of bit variables. 2 For the regulation of alarm variables. 4 For the regulation of register variables. 8 For the regulation of text variables. For example a value of 7 would represent bit, alarm and register variables.

Execution

Mode	Mnemonic	Action
27	FLOWPARAMTONODE	Change the flow regulation parameters for the specified station. Return: 1 if successful, else 0.

The default settings (before use of the VARIABLE instruction) are:
StopLimit 10
StartLimit 1000
Filter 15

Syntax 14

IntVal = VARIABLE(*Mode*, *Handle [,Flag, VarName, FileReport]*);

The return type is INTEGER.

Argument	Meaning
Handle	The location of the memory buffer as returned by ALLOC_BUFFER. Type LONG.
Flag	A flag indicating local use only (0) or global distribution (1) of changes. Global distribution is not yet implemented. Type INTEGER.
VarName	The name of a variable that returns the status of the import. 0 Import in progress. 1 Import completed without errors. 2 Import completed with errors. Type STR.
FileReport	The name of the file containing a report of the import. See the topic Variable Import Report Format for more information. Type STR.

Execution

Mode	Mnemonic	Action
28	IMPORTBUFFER	Import modifications to configuration items associated with variables. That is, variables, expressions, trends and events. In general items can be added, modified and / or deleted. For further information see the topic File syntax for VARIABLE mode IMPORTFILE and IMPORTBUFFER. The import makes a permanent change, that is the changes persist after the project is shutdown and restarted. The configuration items are stored in the memory buffer allocated by ALLOC_BUFFER. Return: 1 if the handle is correct, else 0.

This method is no longer recommended for configuration import. Instead it is preferable to use the Smart Generator and the Generic XML Import.

Syntax 15

IntVal = VARIABLE(*Mode*, *FileName [,Flag, VarName, FileReport, FileError]*);

The return type is INTEGER.

Argument	Meaning
FileName	The name of a text file that describes the configuration items. Type STR.
Flag	A flag indicating local use only (0) or global distribution (1) of changes. Global distribution is not yet implemented. Type INTEGER.
VarName	The name of a variable that returns the status of the import. 0 Import in progress. 1 Import completed without errors. 2 Import completed with errors.

	Type STR.
FileReport	The name of the file containing a report of the import. See the topic Variable Import Report Format for more information. Type STR.
FileError	The name of the file containing a report of any syntax errors for the import. Default FORM.ERR in the LOG FILES folder. Type STR.

Execution

Mode	Mnemonic	Action
29	IMPORTFILE	Import modifications to configuration items associated with variables. That is, variables, expressions, trends and events. In general items can be added, modified and / or deleted. For further information see the topic File syntax for VARIABLE mode IMPORTFILE and IMPORTBUFFER. The import makes a permanent change, that is the changes persist after the project is shutdown and restarted. The configuration items are stored in a file. Return: 1 if the file exists, else 0.

This method is no longer recommended for configuration import. Instead it is preferable to use the Smart Generator and the Generic XML Import.

Syntax 16

LongVal = VARIABLE(*Mode*, *Branch*, *Handle*, *VarName, Flag*);

The return type is LONG.

Argument	Meaning
Branch	An optional branch to be used in conjunction with the variable names provided by the buffer. Type STR.

Handle	The location of a memory buffer containing a list of variable names. The maximum size of the buffer is 128 KB. Type LONG.
	The format of the buffer is a list of the variable names. Those with a branch have a prefix of '@', for instance:
	`@VARNAME1`
	`VARNAME2`
	`VARNAME3`

VarName	The name of a bit variable. Type STR.
Flag	Either 0 or 1.

Execution

Mode	Mnemonic	Action
30	STARTWATCHLIST	The variables, the names of which are contained in the memory buffer, are put on scan. When all variables from the list are available the value specified in the flag is written to the specified bit variable. Return: The handle of a memory buffer containing a list of the variables successfully put on scan. This may then be used in conjunction with mode 31.

Syntax 17

IntVal = VARIABLE(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning

Handle	The handle of a memory buffer returned from a previous mode 30 instruction. Type LONG.

Execution

Mode	Mnemonic	Action
31	STOPWATCHLIST	The variables that have been previously put on scan with a mode 30 instruction are taken off scan. 1 if successful, else 0.

Syntax 18

LongVal = VARIABLE(*Mode*, *VarName*);

The return type is LONG.

Argument	Meaning
VarName	The name of a variable. Type STR.

Execution

Mode	Mnemonic	Action
32	GET_LONG_IN_DB	The real time value of the named variable is returned as a LONG.

Syntax 19

DblVal = VARIABLE(*Mode*, *VarName*,);

The return type is DOUBLE.

Argument	Meaning
VarName,	The name of a variable. Type STR.

Execution

Mode	Mnemonic	Action
33	GET_DOUBLE_IN_DB	The real time value of the named variable is returned as a DOUBLE.
34	GET_PHYSICAL_MIN	The physical minimum for the named variable is returned.
35	GET_PHYSICAL_MAX	The physical maximum for the named variable is returned.
36	GET_CONTROL_MIN	The control minimum for the named variable is returned.
37	GET_CONTROL_MAX	The control maximum for the named variable is returned.

Syntax 20

IntVal = VARIABLE(*Mode*, *VarName, Value*);

The return type is Integer.

Argument	Meaning
VarName	The name of a variable. Type STR.
Value	The new value of the Boolean attributes of the variable. Type: INTEGER, LONG, SINGLE, DOUBLE or CONST.

Execution

Mode	Mnemonic	Action

38	SETBATT	Changes the value of the binary attributes of the specified variable. Return: 1 if successful, else 0.

Syntax 21

IntVal = VARIABLE(*Mode*, *VarName, TattNo*, *Text*);

The return type is Integer.

Argument	Meaning
VarName,	The name of a variable. Type STR.
TattNo	A text attribute number in the range 3 to 16. Any numeric type
Text	The new value for the specified text attribute. Type STR.

Execution

Mode	Mnemonic	Action
39	SETTATT	Set a new value for the specified text attribute. Return: 1 if successful, else 0.

You cannot use mode SETATT to change Domain or Nature (attributes 1 and 2).

Syntax 22

IntVal = VARIABLE(*Mode*, 0, *NoDisplayBarGraph*);

The return type is Integer.

Argument	Meaning
0	Reserved, always equal to 0.
NoDisplayBarGraph	0 Display the bargraph. 1 Hide the bargraph. Default is 0.

Execution

Mode	Mnemonic	Action
40	SAVE	Save the variable configuration. Return: 1 if successful, else 0.

Syntax 23

SVal = VARIABLE (*Mode*, *VarName*);

The return type is SINGLE.

Execution

Mode	Mnemonic	Action
54	GET_DEADBAND_VALUE	Returns the deadband value of the variable.

Syntax 24

IntVal = VARIABLE (*Mode*, *VarName, DBvalue*, *DBtype*);

The return type is INTEGER.

Argument	Meaning

VarName	The name of a variable within the Variables Tree. Type STR.
DBvalue	A new deadband value. Type SINGLE.
DBtype	A new deadband type. Type INTEGER

Execution

Mode	Mnemonic	Action
55	SET_DEADBAND	Set a new deadband value and type. Return: -5 The variable does not exist or is not a register. -4 The parameter *Dbtype* is out or range (must be 0,1 or 2). -3 The parameter *Dbtype* is not a integer. -2 The parameter *Dbvalue* is not a single. -1 The variable is a temporary variable. 1 OK.

Syntax 25

IntVal = VARIABLE (*Mode*, *VarName, Lang0[, Lang1]*);

The return type is INTEGER.

Argument	Meaning
VarName	The name of a variable within the Variables Tree. Type STR.
Lang0	String for language 0. Type STR.
Lang1	String for language 1. Optional. Type STR

Execution

Mode	Mnemonic	Action
41	SETLONGLABEL	Change a variable's Description property. If the project is configured as bilingual you can change the string for both language 0 and language 1. The change is permanent. Return: 1 if successful, else 0.

Syntax 26

IntVal = VARIABLE (*Mode, VarName[, Scope [,Description_1[, Description_2[,AssocLabel]]]]*);

The return type is INTEGER.

Argument	Meaning
VarName	The name for the variable. Type STR.
Scope	The scope for the variable 0 = Session Context. 1 = Client Context. 2 = Local Station. The default value is 0.
Description_1	The description for the variable for language 0. Default value = "". Type STR
Description_2	The description for the variable for language 1. Default value = "". Type STR
AssocLabel	The name of the associated label for the variable. Default value = "". Type STR

Execution

Mode	Mnemonic	Action
60	ADD_HMIBIT	An HMI bit variable is created using the supplied attributes.

Return:
0 if successful.
1 if syntax error.
2 if a variable of the same name already exists.

Syntax 27

IntVal = VARIABLE (*Mode, VarName[, Scope [,Description_1[, Desciption_2[, AssocLabel[, Min[, Max[, Format_1[, Format_2[, Unit_1[, Unit_2]]]]]]]]]]*);

The return type is INTEGER.

Argument	Meaning
VarName	The name for the variable. Type STR.
Scope	The scope for the variable 0 = Session Context. 1 = Client Context. 2 = Local Station. The default value is 0.
Description_1	The description for the variable for language 0. Default value = "". Type STR
Description_2	The description for the variable for language 1. Default value = "". Type STR
AssocLabel	The name of the associated label for the variable. Default value = "". Type STR
Min	The minimum value of the range of the register. Default value = 0. Any numeric type.
Max	The maximum value of the range of the register. Default value = 65535. Any numeric type.
Format_1	The display format for language 0, for example ###.##. Default value = "". Type STR.
Format_2	The display format for language 1, for example ###.##. Default value = "". Type STR.
Unit_1	The units text for language 0, for example DegC. Default value = "". Type STR.
Unit_2	The units text for language 1, for example DegC. Default value = "". Type STR.

Execution

Mode	Mnemonic	Action
61	ADD_HMIREG	An HMI register variable is created using the supplied attributes. Return: 0 if successful. 1 if syntax error. 2 if a variable of the same name already exists.

Syntax 28

IntVal = VARIABLE (*Mode, VarName[, Scope [, Description_1[, Description_2[, AssocLabel[, TextSize]]]]]*);

The return type is INTEGER.

Argument	Meaning
VarName	The name for the variableof a variable within the Variables Tree. Type STR.
Scope	The scope for the variable 0 = Session Context. 1 = Client Context. 2 = Local Station. The default value is 0.
Description_1	The description for the variable for language 0. Default value = "". Type STR. String for language 0. Type STR.
Description_2	The description for the variable for language 1. Default value = "". Type STR.
AssocLabel	The name of the associated label for the variable. Default value = "". Type STR.
TextSize	The maximum size for the text. Default value 132. Any numeric type.

Execution

Mode	Mnemonic	Action
62	ADD_HMITXT	An HMI text variable is created using the supplied attributes. Return: 0 if successful. 1 if syntax error. 2 if a variable of the same name already exists.

Syntax 29

IntVal = VARIABLE (*Mode, VarName*);

The return type is INTEGER.

Argument	Meaning
VarName	The name of an HMI variable. Type STR.

Execution

Mode	Mnemonic	Action
63	REMOVE_HMIVAR	Remove the named HMI variable. Return: 0 if successful. 1 if the variable does not exist.

A note about variable validity

In addition to having a value, each variable in the Variables Tree also has a validity status. A variable which is invalid is displayed in a special way by any animations using it; for example, a register display will be replaced by a "?". Several conditions may cause a variable to be invalid and these may be divided into 3 main classes:

Cause	Meaning

Masking	See the topic Masking by Program.
Inhibition	When a variable is inhibited, its scanning is abandoned. A variable is normally inhibited when the hardware associated with it fails.
Inaccessible	Inaccessibility is a condition that is detected automatically as a result of the external source of a variable becoming unavailable.

Examples

For mode 15, THRESHOLD_SETVALUE:

To set the 1st threshold of variable var1 to 100 and the 3rd threshold to 500, permanently:

```
intval = VARIABLE("THRESHOLD_SETVALUE","var1",0,100,2,500,1);
```

For mode 38, SETBATT:

To force the Boolean attributes of the variable int.b1 from the register variable int.b1.batt.

```
SUB SetBAtt()
VARIABLE("SETBATT", "int.b1", int.b1.batt);
END SUB
```

To force the third Boolean attribute of the variable int.b1 to 1 and the other bits to 0:

```
SUB SetBAtt3()
VARIABLE("SETBATT", "int.b1", 4);
END SUB
```

For mode 56, TREND:

```
PRINT("x1 is a trend variable");
PRINT(VARIABLE(56, "x1"));
PRINT("x2 is not a trend variable");
PRINT(VARIABLE("TREND", "x2"));
```

Results:

```
 x1 is a trend variable
 1
 x2 is not a trend variable
 0
```

For longer examples, select the Example link above.

w

WEBVUE

See Also Example

Manage the connections to WebVue clients.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	LIST	1
2	SETWINDOW	2
3	RESTOREWINDOW	3
4	CONNECTURL	4
5	RESTOREURL	5
6	USERNAME	6
7	WEBVUE_CONTEXT	8
8	MULTIMEDIA	7
9	CONTEXT	6

Syntax 1

IntVal = WEBVUE(*Mode*, *Handle*);

The return type is INTEGER.

Argument	Meaning

Handle	Identifies the list of user-station pairs of connected client stations. Type LONG.
	The list's format is:
	`Username1,hostname1\thostipaddress1\trouteripaddress1,`
	`Username2,hostname2\thostipaddress2\trouteripaddress2,`
	`. . .`

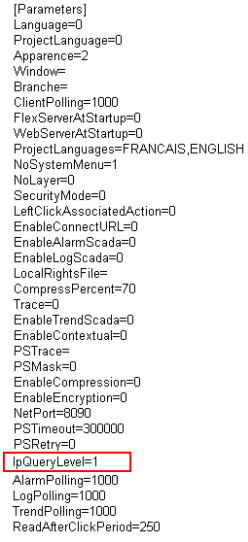
Execution

Mode	Mnemonic	Action
1	LIST	List of web clients. Returns: Number of connected clients, or: -1 if the allocation for *Handle* is not large enough. (See ALLOC_BUFFER.)

Prerequisite

Before using the LIST mode, you must change a parameter in the file WEBVUE.DAT:

1. Select the file WEBVUE.DAT in the WEB folder of the project and make a backup copy. Show picture



2. Open the file in a text editor and change the parameter IpQueryLevel as follows:

Setting	Effect
1	The applet tries to obtain its IP address.
2	IIS tries to obtain the client's IP address.
3	To try to get the IP address of the client computer.
4	The Supervisor tries to resolve the host's name via IP.

If the Supervisor does not have the required access rights for the network, or if it is in a purely Internet-based environment, it will be unable to obtain the client's IP address.

Syntax 2

IntVal = WEBVUE(Mode, Branch, WindowName[, Saved]);

The return type is INTEGER.

Argument	Meaning
Branch	The name of the branch of the window. Type STR.
WindowName	The name of the default window. Type STR.
Saved	An optional flag indicating if the change should be permanent. (After the Supervisor has been restarted) (No longer used.) 0 The change is not permanent. This is the default value. 1 The change is permanent.

Execution

Mode	Mnemonic	Action
2	SETWINDOW	Select the start-up window for the Web client. Return: 1 if successful, else: -1 Client name is incorrect. -2 Window name is incorrect.

Syntax 3

IntVal = WEBVUE(Mode, ClientName);

The return type is INTEGER.

Argument	Meaning
ClientName	The name of the client station. Type STR.

Execution

Mode	Mnemonic	Action
3	RESTOREWINDOW	By default, display the web configuration in the start-up window on the client station. (No longer used.) Return: 1 if successful, else -1.

Syntax 4

IntVal = WEBVUE(Mode, UserName, HtmlPage , BitVariableName, Reconnect, NewBrowser);

The return type is INTEGER.

Argument	Meaning
UserName	Leave empty (""). Type STR.

HtmlPage The name of the HTML page to call when connecting, whether directly as Name.HTML (on the local HTTP server) or remotely (http://www.*xxxx*.html). Type STR.

BitVariableName A database BIT variable that is set to 1 when the connection is made. Type STR.

Reconnect After disconnection: a delay before reconnection is attempted. Type INTEGER.

0: Re-connect immediately.
1: Delay time (in seconds) after inactivity of mouse and keyboard.

NewBrowser Whether to open a new instance of the browser. Type INTEGER.

0: Use the current browser window.
1: Open a new one.

Execution

Mode	Mnemonic	Action
4	CONNECTURL	Select the start window of the Web client. Return: 1 if successful, else 0 (no such bit, or wrong type).

Syntax 5

IntVal =*WEBVUE(Mode*, *UserName*);

The return type is INTEGER.

Argument	Meaning
UserName	The user name as specified in the rights. Type STR.

Execution

Mode	Mnemonic	Action
5	RESTOREURL	Provide the user name for connection to a URL address. Return: 1 if successful, else 0.

Syntax 6

IntVal = WEBVUE(*Mode*, *WebSession, UserHandle*);

The return type is INTEGER.

Argument	Meaning
WebSession	Web session type. Type INTEGER. (For details see GETARGmode 29: WEB.)
UserHandle	The handle of a buffer in which the result of the instruction is returned. Type LONG.

Execution

Mode	Mnemonic	Action
6	USERNAME	Get the name of the user that started the WebVue session. Return: 0 if successful, else: -1: *WebSession* is not an INTEGER. -2: *WebSession* is empty. -3: *UserHandle* is not a LONG. -4: *UserHandle* has a size of 0 (i.e. no buffer was allocated). -5: *UserHandle* is empty. (Normally this is impossible.) -6: *UserHandle.*'s length is larger than the size allocated by ALLOC_BUFFER.
9	CONTEXT	Get the name of the context variable when the program is run from a context window.

Return: 1 if successful, else 0.

Both modes return a null string in the buffer when run from the Supervisor. You can get a list of active WebVue connections at the Supervisor by using mode 1 (LIST).

Syntax 7

IntVal = WEBVUE(*Mode*, *Sub-mode*);

The return type is INTEGER.

Argument	Meaning
Sub-mode	The sub-mode. Type STR. BEEP - Play an audio file.

Execution

Mode	Mnemonic	Action
9	MULTIMEDIA	Play a multimedia file on the PC running the WebVue client. Return: 1 if successful, else 0.

The operation of mode MULTIMEDIA, sub-mode BEEP will depend on the web context of the program containing it.

If the web context is zero then the audio file is played on all current WebVue sessions. The web context is zero if the program is running at the WebVue server, for example if triggered by an event.

If the web context is non zero then the audio file is played only on the host WebVue session. The web context is non-zero if the program is running at the WebVue client, for example triggered by a button in a mimic displayed at the client.

The ability of a WebVue client to play a multimedia file depends on the setting of the property Allow multimedia operation in the profile of the User that started the WebVue session. See the topic Adding a WebVue User in the main help.

It is not possible to specify the sound file played by mode MULTIMEDIA, sub-mode BEEP as it is embedded in a Java applet that is part of WebVue. However it is possible to edit the applet to use a different sound file. See the topic Changing the sound played by WEBVUE.

Syntax 8

IntVal = *WEBVUE(Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
7	WEBVUE_CONTEXT	Return the context in which the program was executed. Return: 0 when the program is executed on the Supervisor, else !=0 when executed on a web client.

Example

For an example, select the Example link above.

WHILE...WEND

See Also

Execute a series of instructions in a loop while a given condition remains true.

WebVue support - Yes.

Syntax

WHILE(*expression*)

[block of instructions]

WEND

Argument	Action
expression	See below.

Execution

- If the expression is true, all instructions included between WHILE and WEND are executed, then the expression is evaluated again.
- If the expression is still true, the process is repeated.
- If the expression is false, execution continues at the instruction following WEND.

Example

```
SUB Main()
DIM i as Integer;

i=0;
WHILE (i<3) 'so long as i<3
  i++;  'increment i by 1
  PRINT(i);
WEND
END SUB
```

Nesting is possible within the limits of available memory.

WINDOW

See Also Example

Open and close windows under program control.

Partial WebVue support - see mode table.

Mode	Mnemonic	Syntax	WebVue
0	CLOSE	1	Yes
1	OPEN	1, 2, 5	Yes
2	IS_OPEN	1	
3	SHOW	1	No
4	HIDE	1	Yes
6	CLOSEUNDER	1	
5	CHANGE	10	
7	PRELOAD	1, 5	
8	CLOSEALL	6	Yes
9	MAIN	3	
10	CURRENTNAME	7	
11	CURRENTBRANCH	7	
12	CAPTION	8	
13	REFSET	9	
14	CURRENTREF	7	
15	POPUPCLOSE	6	Yes
16	ACCESSLEVEL	1	
17	OPENNEW	1, 2, 5	Yes
18	PRINT	1	
19	ZOOM	11	
20	GETREGION	1	
21	GETSUBWINDOW	12	Yes
22	GETSUBBRANCH	12	Yes
23	SETPREVIOUS	6	
24	LAYER	13	
25	HARDCOPY	14	
26	SAVE	15	
27	SELECTTAB	16	

Syntax 1

IntVal = WINDOW(*Mode*, *WinName, Branch* [,*Refset*]);

The return type is INTEGER.

Argument	Meaning
WinName	The name of a window. Type STR.
Branch	A Variables Tree Branch. Type STR.
Refset	The submode in which the window is opened: 0 Real time mode. 1 Mode REF1 2 Mode REF2 3 Mode TEST

Execution

Mode	Mnemonic	Action
0	CLOSE	Close the window.

If a dialog box is open for an animation in the mimic in Run mode, that mimic cannot be closed. The return from this mode is then 0.

If the mimic is in a child window, then the parent window cannot be closed.

1	OPEN	Open the window. The window is automatically moved to the front and given focus, even when it is already open.
2	IS_OPEN	Test whether the window is open.
3	SHOW	Make a hidden window visible.
4	HIDE	Hide a window is temporarily, although it remains active.
6	CLOSEUNDER	Close all windows that match *WinName*, whether in foreground or background.
7	PRELOAD	Preload the window into memory. It is not displayed.
16	ACCESSLEVEL	Return the access level of the specified window.
17	OPENNEW	Open a window. If the window is already open no action is taken.
18	PRINT	Print the selected window on the default printer. Correct functioning of the HARDCOPY or PRINT modes may be affected by the operating system. If in doubt check the operation first before including in a project.
20	GETREGION	Return the region (on a multi-region system) in which the window is located. Return for all modes except 16 and 20: 1 if successful, else 0.

Syntax 2

IntVal = WINDOW(*Mode*, *Child*, *Branch*, *Parent*);

The return type is INTEGER.

Argument	Meaning
Child	The name of a window. Type STR.
Branch	Branch name.
Parent	The name of a window. Type STR.

Execution

Mode	Mnemonic	Action
1	OPEN	Open a window as a child of another window. Focus is transferred to the child window even if it is already open.
17	OPENNEW	Open a window as a child of another window. Focus is transferred to the child window only if it is not already open. Return: 1 if successful, else 0.

The parent window must be open. The child window can only be moved within the bounds of the parent window. If the parent window is closed the child window also closes.

Syntax 3

IntVal = WINDOW(*Mode*, *X*, *Y*, *Width*, *Height*);

The return type is INTEGER.

Argument	Meaning

X, Y	The co-ordinates of the top left hand corner of the window, expressed in pixels. Type INTEGER.
Width	The width of the window expressed in pixels. Type INTEGER.
Height	The height of the window expressed in pixels. Type INTEGER.

Execution

Mode	Mnemonic	Action
9	MAIN	Change the position and size of the main window. Return: 1 if successful, else 0.

Syntax 5

IntVal =WINDOW(*Mode*, *Child*, *ChildBranch*, *Parent*, *ParentBranch*, *dX*, *dY*);

The return type is INTEGER.

Argument	Meaning
Child, *ChildBranch*, *Parent*, *ParentBranch*	See below.
dX, *dy*	Relative X and Y co-ordinates.

Execution

Mode	Mnemonic	Action
1	OPEN	Open a window as a child of another window, using the supplied relative co-ordinates. Focus is transferred to the child window even if it is already open.
7	PRELOAD	Pre-load a window as a child of another window, using the supplied relative co-ordinates. Pre-load loads the window into memory but it is not displayed.
17	OPENNEW	Open a window as a child of another window, using the supplied relative co-ordinates. Focus is transferred to the child window only if it was not already open. Return: 1 if successful, else 0.

For modes 1 and 17, the parent window must be open. The child window can only be moved within the bounds of the parent window. Wen the parent window closes, the child window also closes.

Syntax 6

IntVal = WINDOW(*Mode*);

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
8	CLOSEALL	Close all open windows including any resident in memory but not displayed.
15	POPUPCLOSE	Close all pop-up windows.
23	SETPREVIOUS	Moves the pointer to the previously opened window. This selects the window that is opened with the next use of a Link-window open animation when using the substitution #P (open previous window). Return: 1 if successful, else 0.

Syntax 7

StrVal = WINDOW(*Mode*);

The return type is STR.

Execution

Mode	Mnemonic	Action
10	CURRENTNAME	Return the name of the currently active window.
11	CURRENTBRANCH	Return the branch of the currently active window.
14	CURRENTREF	Return 1 if the currently active window is in Reference status, else 0.

Syntax 8

IntVal = WINDOW(*Mode*, *WinName, Branch*, *TitleLang1 [,TitleLang2]*);

Argument	Meaning
WinName, Branch	See below.
TitleLang1	The new window caption for language 1. Maximum length 40 characters.
TitleLang2	The new window caption for language 2. Optional. Maximum length 40 characters.

The return type is STR.

Execution

Mode	Mnemonic	Action
12	CAPTION	Change the window caption. Several special formatting character sequences are recognised: #D - Substitute the date in DD/MM/YY format. #h - Substitute the time in HHMMSS format. #B - Substitute the window's *Branch*. #N - Substitute the current user name.

Syntax 9

IntVal = WINDOW(*Mode*, *WinName, Branch*, *OldMode*, *NewMode*);

The return type is INTEGER.

Argument	Meaning
WinName	The name of a window. Type STR.
Branch	A branch. Type STR.
OldMode	The current mode of the window. (0, 1, 2, or 3) Type INTEGER.
NewMode	The new mode for the window. (0, 1, 2, or 3) Type INTEGER.

Execution

Mode	Mnemonic	Action
13	REFSET	Change the operational mode of the window. Return: 1 if successful, else 0

When a mimic uses a reference set (Ref Set 1 or Ref Set 2), it receives values replayed from a stored report, not from the Variables Tree.

Syntax 10

IntVal = WINDOW(*Mode*, *WinName, Branch*, *RefSet, X*, *Y*, *W, H*);

The return type is INTEGER.

Argument	Meaning
WinName	Window name.
Branch	Branch name.
RefSet	The RefSet for the window. Normally 0 but can be 1, 2 or 3 if opened in RefSet mode. Type INTEGER.
X, *Y*	The co-ordinates of the top left hand corner of a window. Type INTEGER.
W, H	The width and height of a window. Type INTEGER.

Execution

Mode	Mnemonic	Action
5	CHANGE	Change the position and size of a window. Return: 1 if successful, else 0.

This instruction does not work if the window contains one of the macro animations that is a trend viewer, alarm viewer, log viewer or grid control.

Syntax 11

IntVal = WINDOW(*Mode*, *WinName, Branch*, *X*, *Y*, *Zoom*);

The return type is INTEGER.

Argument	Meaning
WinName	Window name.
Branch	Branch name.
X, *Y*	The co-ordinates of the central point of the displayed zone. Type INTEGER.
Zoom	The zoom percentage for the window. Type INTEGER.

Execution

Mode	Mnemonic	Action
10	ZOOM	Change the zoom level of the specified window. The central point of the window is determined by the X and Y co-ordinates. Return: 1 if successful, else 0.

Syntax 12

StrVal = WINDOW(*Mode*, *Substitution, MyWindow*, [*MyWindowBranch*]);

The return type is STR.

Argument	Meaning
Substitution	One of the substitution strings used with the Link Open animation. Type STR.
MyWindow	(for substitution type #Mx) The window that contains the links. Type STR.
MyWindowBranch	(for substitution type #Mx) The branch (if any) of that window. Type STR.

Execution

Mode	Mnemonic	Action
21	GETSUBWINDOW	Return the name of a window associated with one of the following substitutions:

#P returns the name of the previous window opened.
#U returns the name of the User's primary window.
#I returns the first window opened when the project was started.
#Mx returns the name of one of the windows (*x*= 1 to 10) from the window's link tab.

22	GETSUBBRANCH	Return the name of a *Branch*associated with one of the above substitutions.

The window, *MyWindow*, that contains the links must be open.

Syntax 13

IntVal = WINDOW(*Mode*, *WinName, Branch*, *Refset*, *Layer*, *LayerMode*);

The return type is INTEGER.

Argument	Meaning
WinName	The name of a window. Type STR.
Branch	A branch. Type STR.
Refset	A number indicating the mode of the window. (0, 1, 2 or 3) Type INTEGER.
Layer	The number of the layer for the operation. (0 to 15) Type INTEGER.
Layermode	The mode for the operation. (0, 1 or 2) Type INTEGER.

Execution

Mode	Mnemonic	Action
24	LAYER	Change the status of the layer in the specified window, according to *LayerMode*: 0 Turned off (becomes invisible). 1 Turned on (becomes visible). 2 Toggled. Return: 1 if successful, else 0

Syntax 14

IntVal = WINDOW(*Mode*, *WinName, Branch* [, *Refset*[, *PrintMode*]]);

The return type is INTEGER.

Argument	Meaning
WinName	The name of a window. Type STR.
Branch	A Variables Tree *Branch*. Type STR.
Refset	A number indicating the mode of the window. (0, 1, 2 or 3) Type INTEGER.
PrintMode	The mode for the operation. (0 or 1) Type INTEGER.

Execution

Mode	Mnemonic	Action
25	HARDCOPY	Print a hard copy of the specified window on Windows' default printer. The window must be open or in the cache. If *PrintMode* is: 0 The window is printed without its border (default mode). 1 The window is printed with its border. Return: 1 if successful, else 0

Correct functioning of the HARDCOPY or PRINT modes may be affected by the operating system. If in doubt check the operation first before including in a project.

The *RefSet* parameter used in some of the above modes controls the window display:

0 Normal mode.
1 Reference Mode 1.
2 Reference Mode 2.
3 Test mode.

Syntax 15

IntVal = WINDOW(*Mode*, *WinName, Branch*, *Format*);

The return type is INTEGER.

Argument	Meaning
WinName	The name of a window. Type STR.
Branch	A Variables Tree Branch. Type STR.
Format	The format in which to save the window. Type STR.

Execution

Mode	Mnemonic	Action
26	SAVE	Save the named window. The format in which the window is saved is selected by the *Format* parameter: NATIVE: current format. BINARY: binary format. ASCII32: ASCII format. Return: 1 if successful, else 0

Syntax 16

IntVal = WINDOW(*Mode*, *WinName, Branch*, *TabIndex*);

The return type is INTEGER.

Argument	Meaning
WinName	The name of a window. Type STR.
Branch	A Variables Tree Branch. Type STR.
TabIndex	The index of the tab. Type INTEGER.

Execution

Mode	Mnemonic	Action
27	SELECTTAB	For a window using tabs, select the tab corresponding to the supplied index. Return: 1 if successful, else 0

Example

For an example, select the Example link above.

X-Z

XMLPATH

See Also Examples

Process XML formatted data according to XPath (XML Path Language) specifications.

WebVue support - Yes.

Mode	Mnemonic	Syntax
1	GET	1
2	GETSTR	2
3	GETDOUBLE	3
4	GETSINGLE	4
5	GETLONG	5
6	GETINT	6
7	GETBOOL	7
8	LOAD	8
9	UNLOAD	9
10	LOADFILE	10

Format of the Namespace when using XMLPATH with either the Alarm or Log viewer

The format of the namespace is as follows. Note that the string is case sensitive.

Branch/Mimic/ViewerId

Argument	Meaning
Branch	The branch with which the mimic is opened. If the mimic is not opened with a branch a null string should be used.
Mimic	The mimic file name.
ViewerID	The alarm or log viewer ID which can be found or changed using the Graphic Explorer.

Syntax 1

LongVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.
Xpath	Hierarchical path. Type STR or LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
1	GET	Returns a handle from a path.

LongVal is automatically allocated and reallocated if the verb is called cyclically. There is no need to do any allocation via ALLOCBUFFER. The *LongVal* handle is finally de-allocated when the UNLOAD mode is called.

Syntax 2

StrVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.

Xpath	Hierarchical path. Type STR or LONG.

The return type is STR.

Execution

Mode	Mnemonic	Action
2	GETSTR	Returns a string value from a path.

Syntax 3

DoubleVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.
Xpath	Hierarchical path. Type STR or LONG.

The return type is DOUBLE.

Execution

Mode	Mnemonic	Action
3	GETDOUBLE	Returns a double value from a path.

Syntax 4

SingleVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.
Xpath	Hierarchical path. Type STR or LONG.

The return type is SINGLE.

Execution

Mode	Mnemonic	Action
3	GETSINGLE	Returns a single value from a path.

Syntax 5

LongVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.
Xpath	Hierarchical path. Type STR or LONG.

The return type is LONG.

Execution

Mode	Mnemonic	Action
5	GETLONG	Returns a long value from a path.

Syntax 6

IntVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.

Xpath	Hierarchical path. Type STR or LONG.

The return type is INTEGER.

Execution

Mode	Mnemonic	Action
6	GETINT	Returns and integer value from a path.

Syntax 7

IntVal = XMLPATH(*Mode*, *Namespace*, *Xpath*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.
Xpath	Hierarchical path. Type STR or LONG.

The return type is INTEGER: 1 for a True string, 0 for a False string.

Execution

Mode	Mnemonic	Action
7	GETBOOL	Returns a boolean value (as an integer) from a path. 1 for <true> string. 0 for <false> string.

Syntax 8

IntVal = XMLPATH(*Mode*, *Namespace*, *XMLdata*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.
XMLdata	Content of the XML file. Type STR or LONG.

The return type is INTEGER: 1 for OK, -1 for XML file in error, -2 no XML file.

Execution

Mode	Mnemonic	Action
8	LOAD	Load and map the XML data into the context identified by a name-space.

Syntax 9

IntVal = XMLPATH(*Mode*, *Namespace*);

Argument	Meaning
Namespace	Identifier for the Xpath. Type STR or LONG.

The return type is INTEGER: 1 for OK, -1 for XML file in error, -2 no XML file.

Execution

Mode	Mnemonic	Action
9	UNLOAD	Unload and unmap XML data from a namespace.

Syntax 10

IntVal = XMLPATH(*Mode*, *Namespace*, *XMLfilename*);

Argument	Meaning

Namespace Identifier for the Xpath. Type STR or LONG.

XMLfilename The name of the XML file to load. Type STR or LONG.
If executed in a WebVue session context this instruction is processed by the computer that hosts the web back end and the referenced file is on that computer.

The return type is INTEGER: 1 for OK, -1 bad XML data, -2 file not found.

Execution

Mode	Mnemonic	Action
10	LOADFILE	Load and map XML data from a file.

Example

For an example, select the Examples link above.

Examples

Overview of the Examples

See Also

There are these kinds of example in the online help for SCADA Basic:

1. Small examples of code embedded in the Instruction topics.
2. Larger examples of code that are held as separate examples in this book.

Some of the larger examples were extracted from application projects that provided the context for running them.

Purpose and limits of use

The examples of all three kinds of project are designed to illustrate the SCADA BASIC commands. The larger examples may show the modes and syntaxes of a particular command.

Some examples provide reminders about practical application of the commands. However, they are not intended for running or to illustrate best practice for SCADA/HMI development or for programming. They may require particular contexts to be configured before execution can occur. For these reasons they should only be used for understanding, not as ready-made components for incorporating into applications.

How to find the examples

You can find examples by navigating or by searching the Help contents.

- Refer to any small examples that are in the topic for an instruction, usually at the end of that topic.
- Look up an instruction's name in this Help book or via the See Also link in the topic for the instruction.
- Use the help's Search tab to locate all examples of a SCADA BASIC instruction in the command topics and in the separate example topics. Include the word 'Example' and tick the option

Navigation is more precise, but searching has advantages. You can list all topics that mention a particular instruction (even where the example given is mainly about another instruction). You can also search for a combination of instructions, for example:

```
SendList NEAR ComboBox
```

Searching techniques are explained in the main Help book on How to Search the Help.

ALARMDISPLAY Example

Applies To

Example 1

In the following example, the window *AllAlarm* contains an Alarm Viewer animation identified as *AllDomain*. It assumes that an Alarm Viewer has been configured without the lower border, whose functionality has been replaced with command buttons to launch the various sub-programs.

```
SUB Main ()
  DIM Win As Str, ID AS Str;
  Win = "AllAlarm";
  ID = "AllDomain";
END SUB

SUB ToStart ()
  AlarmDisplay("Begin",Win,"",ID);
END SUB

SUB ToEnd ()
  AlarmDisplay("End",Win,"",ID);
END SUB

SUB PageUp ()
  AlarmDisplay("Before", Win, "", ID);
END SUB

SUB PageDown ()
  AlarmDisplay("After", Win, "", ID);
END SUB

SUB ListMode ()
  AlarmDisplay("Mode",Win,"",ID);
END SUB

' Display alarms of the previous hour
SUB daterange()
  DIM Hd1 as double;
  DIM Hd2 as double;
  DIM ret as integer;
  Hd2 = DateTimeValue();
  Hd1 = Hd2 - 3600000;
  ret = ALARMDISPLAY("DATERANGE", "alm1","","IDalarm1",Hd1,Hd2);
  print ( "ALARMDISPLAY mode DATERANGE ret=", ret);
END SUB

' Reset the filter on the date
SUB resetdaterange()
  DIM Hd1 as double;
  DIM Hd2 as double;
  DIM ret as integer;
  Hd2 = 0;
  Hd1 = 0;
  ret = ALARMDISPLAY("DATERANGE", "alm1","","IDalarm1",Hd1,Hd2);
  print ( "ALARMDISPLAY mode DATERANGE ret=", ret);
END SUB
```

Example 2

This example uses the instructions ADDSTRING, LINESELECT and XMLPATH:

```
SUB main()
END SUB
```

```
SUB subscribe()
DIM WinName as str;
DIM WinBranch as str;
DIM NameSpace as str;
DIM iRet as integer;

WinName = WINDOW("CURRENTNAME");
WinBranch = WINDOW("CURRENTBRANCH");
NameSpace = AddString(WinBranch,"/");
NameSpace = AddString(NameSpace,WinName);
NameSpace = AddString(NameSpace,"/alarm1");
@lineselect.Syno = WinName;
@lineselect.Branch = WinBranch;
@lineselect.AlarmSelected = "";
@lineselect.NameSpace = NameSpace;
iRet = AlarmDisplay("LINESELECT",WinName,WinBranch,"alarm1","P","","SelectLine","");
Print("Subscription with result: ",iRet);
END SUB

SUB SelectLine()
DIM WinName as str;
DIM WinBranch as str;
DIM NameSpace as str;
DIM AlarmSelected as str;
DIM iBuf as long;

WinName = WINDOW("CURRENTNAME");
WinBranch = WINDOW("CURRENTBRANCH");
NameSpace = AddString(WinBranch,"/");
NameSpace = AddString(NameSpace,WinName);

NameSpace = AddString(NameSpace,"/alarm1");
@lineselect.NameSpace = NameSpace;
'AlarmSelected = XMLPATH("GETSTR",NameSpace,"lineselect/variable");
iBuf = XMLPATH("GET",NameSpace,"lineselect/variable");
AlarmSelected = CGET_BUFFER(iBuf, 0, 255);
@lineselect.AlarmSelected = AlarmSelected ;
XMLPATH("UNLOAD",NameSpace);
END SUB
```

APPLICATION Examples

Applies To

Example 1

This program shows examples of starting an external application.

For each "\" in a Windows path, SCADA BASIC needs "\\".

```
'Mode ISLOADED or Mode 1 (syntax 1)

SUB ApplicationIsLoaded()
'Only runs if the application has been launched with APPLICATION LOAD
'Return code
DIM intReturn As Integer;
DIM strLink As Str;
'for each "\" in the Filename => SCADA BASIC needs "\\"
'for example:
'  "C:\WINDOWS\NOTEPAD.EXE" => "C:\\WINDOWS\\notepad.exe"
strLink = "C:\\WINDOWS\\notepad.exe";
intReturn = APPLICATION("ISLOADED",strLink);
If (intReturn == 1) Then
    PRINT("Program is already loaded.");
Else
    PRINT("Program is not already loaded.");
End If
END SUB

'Mode ACTIVATE or Mode 2 (syntax 1)

SUB ApplicationActivate()
'Return code
DIM intReturn As Integer;
DIM strLink As Str;
strLink = "C:\\WINDOWS\\notepad.exe";
intReturn = APPLICATION("ACTIVATE",strLink);
If (intReturn == 1) Then
    PRINT("Activation was successful.");
Else
    PRINT("Activation error of ", strLink, ".");
End If
END SUB

'Mode LOAD or Mode 3 (syntax 2)

SUB ApplicationLoad()
'Return code
DIM intReturn As Integer;
DIM strLink As Str;
DIM strArgs As Str;
DIM intWindowStyle As Integer;
strLink = "C:\\WINDOWS\\notepad.exe";
strArgs = "E:\\ScadaBasic\\USR\\APPLICATION\\p\\APPLICATION.SCB";
intWindowStyle = 1; 'Normal mode
intReturn = APPLICATION ("LOAD" , strLink , strArgs , intWindowStyle );
If (intReturn == 1) Then
    PRINT("load successful");
Else
    PRINT("load error");
End If
END SUB

'Mode UNLOAD or Mode 4 (syntax 1)
```

```
SUB ApplicationUnLoad()
'Return code
DIM intReturn As Integer;
DIM strLink As Str;
strLink = "C:\\WINDOWS\\notepad.exe";
intReturn = APPLICATION("UNLOAD",strLink);
If (intReturn == 1) Then
  PRINT("unload successful");
Else
  PRINT("unload error");
End If
END SUB

SUB LoadExcel ()
'example of how to launch the Excel application
DIM intReturn As Integer;
DIM strapplink As Str;
DIM strArgs As Str;
DIM intWindowStyle As Integer;
strapplink = "C:\\Program Files\\Microsoft Office\\Office10\\EXCEL.EXE";
If (Application ("IsLoaded", strapplink)) Then
' The application is already running
' => Activate the application
  If (Application ("Activate", strapplink) == 0) Then
    PRINT("activate error of ", strapplink);
  End If
Else
  'The application is not running, so LOAD the application
  intWindowStyle=5; 'Windows Minimizes not Activated
  strArgs="/E"; 'Open Excel without file opened by default
  'strArgs="/m"; 'Open Excel with a new file including just a Macro
  'strArgs="/r E:\\file1.xls"; 'Open the file e:\file.xls with Excel in Read Only
  'strArgs="/w E:\\file1.xls"; 'Open the file e:\file.xls with Excel in normal mode

  If (Application ("Load", strapplink, strArgs , intWindowStyle) == 0) Then
    PRINT ("load error of ", strapplink);
  End If
End If
END SUB
```

Example 2

This example starts the Excel application.

```
SUB LoadExcel ()
  DIM r As Integer;
  DIM s As Str;
  s = "C:\\EXCEL\\EXCEL.EXE"; 'note double backslashes
  If (Application ("IsLoaded", s)) Then
    'Application already loaded
    'Activate application :
    If (Application ("Activate", s) == 0) Then
      Print("Error in activation of ", s);
    End If
  Else
    'Application not loaded
    'Load application in a minimised window
    ' with the parameter "/E" :
    If (Application ("Load", s, "/E" , 5) == 0) Then
      Print ("Error when loading ", s);
    End If
  End If
END SUB

SUB UnloadExcel ()
  DIM r As Integer;
```

```
  DIM s As Str;
  s = "EXCEL";
  If (Application ("IsLoaded", s)) Then
    If (Application ("Unload", s) == 0) Then
      Print ("Error when unloading ", s);
    End If
  End If
END SUB
```

Example 3

This program exercises mode 3 (LOAD):

```
'Mode LOAD or Mode 3 (syntax 2)
SUB ApplicationLoadWithWorkingFolder()
'Return code
DIM intReturn As Integer;
DIM strLink As Str;
DIM strArgs As Str;
DIM intWindowStyle As Integer;
DIM strDir As Str;
strLink = "C:\\WINDOWS\\notepad.exe";
strArgs = "E:\\ScadaBasic\\USR\\APPLICATION\\p\\APPLICATION.SCB";
intWindowStyle = 1; 'Normal mode
strDir = "E:\\WORDING";
intReturn = APPLICATION ("LOAD" , strLink , strArgs , intWindowStyle, strDir );
If (intReturn == 1) Then
    PRINT("load successful");
Else
    PRINT("load error");
End If
END SUB
```

ASCIIFIELD Examples

Applies To

Example 1

This program exercises several modes of the ASCIIFIELD command.

```
'Retrieve, from an ASCII buffer, ASCII fields separated by a given character
'Variable Tree config:
'@STRING01 As Text
'@STR01 As Text
'@SEPA As Text
'@LEN01 As Register
'@INTEGER01 As Register
'@LONG01 As Register
'@DOUBLE01 As Register
'@COUNT01 As Register
'@INDEX01 As Register

'Retrieval of fields separated by a data character in the ASCII buffer

'Variable Tree prerequisites:
'@STRING01 of type   TEXT
'@STR01 of type TEXT
'@SEPA of type TEXT
'@LEN01 of type REGISTER
'@INTEGER01 of type REGISTER
'@LONG01 of type REGISTER
'@DOUBLE01 of type REGISTER
'@COUNT01 of type REGISTER
'@INDEX01 of type REGISTER
'For these examples you can use:
'@STRING01 = "123.34;my string, here;345;123456789;hello world\n"

'Mode LEN or Mode 1  (syntax 1)

SUB AsciiFieldLen1()'declare code
DIM lngbuffer1 As Long;
DIM strline As Str;
DIM intlengh As Integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT BUFFER(lngbuffer1, 0, strline);
@LEN01 = TOS(ASCIIFIELD("LEN",lngbuffer1));
FREE_BUFFER(lngbuffer1);
END SUB

SUB AsciiFieldLen2()
'declare code
DIM lngbuffer1 As Long;
DIM strline As Str;
DIM intlengh As Integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT_BUFFER(lngbuffer1, 0, strline);
'@SEPA is equal to";"
@LEN01 = TOS(ASCIIFIELD("LEN",lngbuffer1,1,@SEPA));
FREE_BUFFER(lngbuffer1);
END SUB

'Mode COUNT or Mode 2  (syntax 2)

SUB AsciiFieldCount()
'declare code
DIM lngbuffer1 As Long;
```

```
DIM strline As Str;
DIM intcount As Integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT_BUFFER(lngbuffer1, 0, strline);
@COUNT01 = TOS(ASCIIFIELD("COUNT", lngbuffer1,@SEPA ));
FREE_BUFFER(lngbuffer1);
END SUB

'Mode STR or Mode 3  (syntax 1)

SUB AsciiFieldStr()
'declare code
DIM lngbuffer1 As Long;
DIM strline As Str;
DIM strchar As Str;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT_BUFFER(lngbuffer1, 0, strline);
@STR01 = ASCIIFIELD("STR",lngbuffer1,TOI(@INDEX01),@SEPA);
FREE_BUFFER(lngbuffer1);
END SUB

'Mode INTEGER or Mode 4  (syntax 1)

SUB AsciiFieldInteger()
'declare code
DIM lngbuffer1 As Long;
DIM strline As Str;
DIM intInteger As Integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT_BUFFER(lngbuffer1, 0, strline);
@INTEGER01 = TOS(ASCIIFIELD("INTEGER",lngbuffer1,TOI(@INDEX01),@SEPA));
FREE_BUFFER(lngbuffer1);
END SUB

'Mode LONG or Mode 5  (syntax 1)

SUB AsciiFieldLong()
'declare code
DIM lngbuffer1 As Long;
DIM strline As Str;
DIM lngLong As Long;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT_BUFFER(lngbuffer1, 0, strline);
@LONG01 = TOS(ASCIIFIELD("LONG",lngbuffer1,TOI(@INDEX01),@SEPA));
FREE_BUFFER(lngbuffer1);
END SUB

'Mode DOUBLE or Mode 6  (syntax 1)

SUB AsciiFieldDouble()
'declare code
DIM lngbuffer1 As Long;
DIM strline As Str;
DIM dbldouble As double;
lngbuffer1 = ALLOC_BUFFER(110);
strline = @STRING01;
PUT_BUFFER(lngbuffer1, 0, strline);
@DOUBLE01 = TOS(ASCIIFIELD("DOUBLE",lngbuffer1,TOI(@INDEX01),@SEPA));
FREE_BUFFER(lngbuffer1);
END SUB
```

Example 2

This program displays ASCII fields in various formats.

```
SUB main()
DIM hdl as long;
DIM line as str;
DIM ch as str;
DIM i as integer;
DIM l as long;
DIM d as double;
DIM count as integer;
DIM length as integer;
hdl = ALLOC_BUFFER(110);
line = "123.34;my string, here;345;123456789;hello world\n";
PUT_BUFFER(hdl, 0, line);
length = ASCIIFIELD("LEN",hdl);
PRINT(length);
length = ASCIIFIELD("LEN",hdl,4,";");
PRINT(length);
PRINT("count = ", ASCIIFIELD("COUNT",hdl,";"));
ch = ASCIIFIELD("STR",hdl,2,";");
i = ASCIIFIELD("INTEGER",hdl,3,";");
l = ASCIIFIELD("LONG",hdl,4,";");
d = ASCIIFIELD("DOUBLE",hdl,1,";");
PRINT(ch);
PRINT(i);
PRINT(l);
PRINT(d);
PRINT(">",ASCIIFIELD("STR",hdl,5,";"),"<");
free_buffer(hdl);
END SUB
```

The program displays these lines in the Program Management debugging window:

```
110
9
count = 5
my string, here
345
123456789
123.34
>hello world<
```

ASSOCLABEL Example

Applies To

This program exercises the ASSOCLABEL command.

```
' Change the labels in the associated label AssocLabel1
SUB SetAssocLabel()
DIM l_bRet AS INTEGER;

'------------------------ First language --------------------
' State to 0
l_bRet = ASSOCLABEL ("SETLABEL", "AssocLabel1", 0, "Open" );
IF (l_bRet == 0) THEN
  PRINT( "ERROR SetLabel State to 0. Return = ", l_bRet);
END IF

' State to 1
l_bRet = ASSOCLABEL ("SETLABEL", "AssocLabel1", 1, "Close");
IF (l_bRet == 0) THEN
  PRINT( "ERROR SetLabel State to 1. Return = ", l_bRet);
END IF

' Transition to 0
l_bRet = ASSOCLABEL ("SETLABEL", "AssocLabel1", 2, "Transition to open");
IF (l_bRet == 0) THEN
  PRINT( "ERROR SetLabel Transition to 0. Return = ", l_bRet);
END IF

' Transition to 1
l_bRet = ASSOCLABEL ("SETLABEL", "AssocLabel1", 3, "Transition to closed");
IF (l_bRet == 0) THEN
  PRINT( "ERROR SetLabel Transition to 1. Return = ", l_bRet);
END IF

' Command to 0
l_bRet = ASSOCLABEL ("SETLABEL", "AssocLabel1", 4, "Forced to open");
IF (l_bRet == 0) THEN
  PRINT( "ERROR SetLabel Command to 0. Return = ", l_bRet);
END IF

' Command to 1
l_bRet = ASSOCLABEL ("SETLABEL", "AssocLabel1", 5, "Forced to closed");
IF (l_bRet == 0) THEN
  PRINT( "ERROR SetLabel Command to 1. Return = ", l_bRet);
END IF

'------------------------ Second language -------------------
l_bRet = ASSOCLABEL ( "SETLABEL", "AssocLabel1", 0, "Animation 0",     1 );
l_bRet = ASSOCLABEL ( "SETLABEL", "AssocLabel1", 1, "Animation 1",     1 );
l_bRet = ASSOCLABEL ( "SETLABEL", "AssocLabel1", 2, "Log change to 0", 1 );
l_bRet = ASSOCLABEL ( "SETLABEL", "AssocLabel1", 3, "Log change to 1", 1 );
l_bRet = ASSOCLABEL ( "SETLABEL", "AssocLabel1", 4, "Bit send 0",      1 );
l_bRet = ASSOCLABEL ( "SETLABEL", "AssocLabel1", 5, "Bit send 1",      1 );
END SUB
```

BACNET Examples

Applies To

This program exercises several modes of the BACNET command.

```
Sub Main()
End Sub

Sub TimeSynchronization()
  BACNET("TIME_SYNCHRONIZATION", "NETWORK01", "SYNCHRO");
End Sub

Sub ResetPriority()
  BACNET( "RESET_PRIORITY", "DEVICE1.ANALOG_VALUE_1.PRESENT_VALUE", 16, "RESULT2");
End Sub

Sub ResetAllPriorities()
  BACNET( "RESET_ALL_PRIORITIES", "DEVICE1.ANALOG_VALUE_1.PRESENT_VALUE", "RESULT");
End Sub

Sub StartNetwork()
  BACNET( "START_NETWORK", "NETWORK01", "RESULTSTARTNET");
End Sub

Sub StopNetwork()
  BACNET( "STOP_NETWORK", "NETWORK01", "RESULTSTOPNET");
End Sub

Sub StartDevice()
  BACNET( "START_DEVICE", "NETWORK01", "DEVICE1","RESULTSTARTDEV");
End Sub

Sub StopDevice()
  BACNET( "STOP_DEVICE");
End Sub

Sub WriteValue()
  BACNET( "WRITE_PRIORITY", "DEVICE1.ANALOG_VALUE_1.PRESENT_VALUE",59.65509,10,
"RESULT1");
End Sub
```

BUFTOFILE Example

Applies To

This program shows the use of the BUFTOFILE instruction.

```
SUB main()
'Declare variables
DIM lngbuffer1 as long;
DIM strline as Str;
DIM intresult as integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = "here,345;123456789 ,,t,\n";
PUT_BUFFER(lngbuffer1, 0, strline);
'Create file test1.txt in folder TP of current project
intresult = BUFTOFILE (lngbuffer1,"test1.txt");

'free the memory area
FREE_BUFFER(lngbuffer1);
END SUB

SUB buftofileusefull()
'Declare variables
DIM lngbuffer1 as long;
DIM strline as Str;
DIM strfilemode as Str;
DIM intresult as integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = "here,345;123456789 ,,t,\n";
PUT_BUFFER(lngbuffer1, 0, strline);
'strfilemode="WRITE";  'modes of BUFTOFILE - WRITE is commented out
strfilemode="APPEND";
'create file test1.txt in folder TP of current project
intresult = BUFTOFILE (lngbuffer1,"test1.txt","USEFULL_PART",strfilemode);
'free the memory area
FREE_BUFFER(lngbuffer1);
END SUB
```

The argument USEFULL_PART ensures that writing to the file stops as soon as a null character is encountered in the buffer.

CGET_BUFFER Example

Applies To

This program uses the CGET_BUFFER instruction.

```
SUB main()
'Declare variables
DIM lngbuffer1 as long;
DIM strline as Str;
DIM strResult as Str;
DIM intlength as integer;

'Create a buffer
lngbuffer1 = ALLOC_BUFFER(110);
strline = "123.34;a string,here;345;123456789;,t,\n";

'The string strline is copied to the buffer
PUT_BUFFER(lngbuffer1, 0, strline);

'Retrieve length of character string
intlength=LEN(strline);
strResult = CGET_BUFFER(lngbuffer1,0,intlength,0);
PRINT("The returned string is: ",strResult);
'This displays: "The returned string is: 123.34;a string, here;345;123456789;,t,"
FREE_BUFFER(lngbuffer1);
END SUB
```

CHECKLIST Example

Applies To

This example contains a set of functions to support a pair of Check-box List form controls using the CHECKLIST instruction.

```
SUB Main()
END SUB

'declare variables:
SUB cbxSel()
DIM lRet As Long; 'return
DIM sCbxText As Str; 'Text
DIM sCbxText1 As Str; 'Text
DIM sCbxData As Str; 'User Data
DIM sCbxData1 As Str; 'User Data

'retrieve content of selected item in 1st control:
SUB chkSel()
DIM lRet As Long;
DIM lRet2 As Long;
DIM schkText As Str;
DIM schkText1 As Str;
DIM schkData As Str;
DIM schkData1 As Str;
lRet = CHECKLIST ( "GETSELECTEDINDEX", "ctrl","","chk" );
CHECKLIST ( "SETSELECTEDINDEX", "ctrl","","chk1", lRet);
schkText = CHECKLIST ( "GETTEXT", "ctrl","","chk", lRet);
schkText1 = CHECKLIST ( "GETTEXT", "ctrl","","chk1", lRet);
schkData = CHECKLIST ( "GETUSERDATA", "ctrl","","chk", lRet);
schkData1 = CHECKLIST ( "GETUSERDATA", "ctrl","","chk1", lRet);
lRet2 = CHECKLIST ( "GETSTATE", "ctrl","","chk" , lRet);
CHECKLIST ( "SETSTATE", "ctrl","","chk1", lRet, lRet2);

'put its contents in a frame:
SET ( "chkText" , schkText);
SET("chkText1", schkText1);
SET("chkData", schkData);
SET("chkData1" , schkData1);
SET("chkCount" , CHECKLIST ( "COUNT", "ctrl","","chk") );
SET("chkCount1" , CHECKLIST ( "COUNT", "ctrl","","chk1"));
SENDLIST ("BLOC");

'concatenate & load contents into a form control of each kind:
COMBOBOX ( "LOAD", "ctrl","","cbx2", ADDSTRING("cbx",schkData));
LISTBOX ( "LOAD", "ctrl","","lbx2", ADDSTRING("lbx",schkData));
CHECKLIST ( "LOAD", "ctrl","","chk2", ADDSTRING("chk",schkData));
OPTIONLIST ( "LOAD", "ctrl","","opt2", ADDSTRING("opt",schkData));
TREEVIEW ( "LOAD", "ctrl","","tvw2", ADDSTRING("tvw",schkData));
END SUB

'retrieve content of selected item in 2nd control:
SUB chkSel1()
DIM lRet As Long;
DIM schkText As Str;
DIM schkText1 As Str;
DIM schkData As Str;
DIM schkData1 As Str;
lRet = CHECKLIST ( "GETSELECTEDINDEX", "ctrl","","chk1" );
CHECKLIST ( "SETSELECTEDINDEX", "ctrl","","chk", lRet);
schkText = CHECKLIST ( "GETTEXT", "ctrl","","chk", lRet);
schkText1 = CHECKLIST ( "GETTEXT", "ctrl","","chk1", lRet);
schkData = CHECKLIST ( "GETUSERDATA", "ctrl","","chk", lRet);
schkData1 = CHECKLIST ( "GETUSERDATA", "ctrl","","chk1", lRet);
lRet = CHECKLIST ( "COUNT", "ctrl","","chk");
```

```
lRet = CHECKLIST ( "COUNT", "ctrl","","chk1");

'put its contents in a frame:
SET ( "chkText" , schkText);
SET("chkText1", schkText1);
SET("chkData", schkData);
SET("chkData1" , schkData1);
SET("chkCount" , CHECKLIST ( "COUNT", "ctrl","","chk") );
SET("chkCount1" , CHECKLIST ( "COUNT", "ctrl","","chk1"));
SENDLIST ("BLOC");
END SUB

'select 2nd item:
SUB setSelchk()
DIM lRet As Long;
CHECKLIST ( "SETSELECTEDINDEX", "ctrl","","chk", 2);
END SUB
```

CHECKLIST Example

Applies To

This example contains a set of functions to support a pair of Check-box List form controls using the CHECKLIST instruction.

```
SUB Main()
END SUB

'declare variables:
SUB cbxSel()
DIM lRet As Long; 'return
DIM sCbxText As Str; 'Text
DIM sCbxText1 As Str; 'Text
DIM sCbxData As Str; 'User Data
DIM sCbxData1 As Str; 'User Data

'retrieve content of selected item in 1st control:
SUB chkSel()
DIM lRet As Long;
DIM lRet2 As Long;
DIM schkText As Str;
DIM schkText1 As Str;
DIM schkData As Str;
DIM schkData1 As Str;
lRet = CHECKLIST ( "GETSELECTEDINDEX", "ctrl","","chk" );
CHECKLIST ( "SETSELECTEDINDEX", "ctrl","","chk1", lRet);
schkText = CHECKLIST ( "GETTEXT", "ctrl","","chk", lRet);
schkText1 = CHECKLIST ( "GETTEXT", "ctrl","","chk1", lRet);
schkData = CHECKLIST ( "GETUSERDATA", "ctrl","","chk", lRet);
schkData1 = CHECKLIST ( "GETUSERDATA", "ctrl","","chk1", lRet);
lRet2 = CHECKLIST ( "GETSTATE", "ctrl","","chk" , lRet);
CHECKLIST ( "SETSTATE", "ctrl","","chk1", lRet, lRet2);

'put its contents in a frame:
SET ( "chkText" , schkText);
SET("chkText1", schkText1);
SET("chkData", schkData);
SET("chkData1" , schkData1);
SET("chkCount" , CHECKLIST ( "COUNT", "ctrl","","chk") );
SET("chkCount1" , CHECKLIST ( "COUNT", "ctrl","","chk1"));
SENDLIST ("BLOC");

'concatenate & load contents into a form control of each kind:
COMBOBOX ( "LOAD", "ctrl","","cbx2", ADDSTRING("cbx",schkData));
LISTBOX ( "LOAD", "ctrl","","lbx2", ADDSTRING("lbx",schkData));
CHECKLIST ( "LOAD", "ctrl","","chk2", ADDSTRING("chk",schkData));
OPTIONLIST ( "LOAD", "ctrl","","opt2", ADDSTRING("opt",schkData));
TREEVIEW ( "LOAD", "ctrl","","tvw2", ADDSTRING("tvw",schkData));
END SUB

'retrieve content of selected item in 2nd control:
SUB chkSel1()
DIM lRet As Long;
DIM schkText As Str;
DIM schkText1 As Str;
DIM schkData As Str;
DIM schkData1 As Str;
lRet = CHECKLIST ( "GETSELECTEDINDEX", "ctrl","","chk1" );
CHECKLIST ( "SETSELECTEDINDEX", "ctrl","","chk", lRet);
schkText = CHECKLIST ( "GETTEXT", "ctrl","","chk", lRet);
schkText1 = CHECKLIST ( "GETTEXT", "ctrl","","chk1", lRet);
schkData = CHECKLIST ( "GETUSERDATA", "ctrl","","chk", lRet);
schkData1 = CHECKLIST ( "GETUSERDATA", "ctrl","","chk1", lRet);
lRet = CHECKLIST ( "COUNT", "ctrl","","chk");
```

```
lRet = CHECKLIST ( "COUNT", "ctrl","","chk1");

'put its contents in a frame:
SET ( "chkText" , schkText);
SET("chkText1", schkText1);
SET("chkData", schkData);
SET("chkData1" , schkData1);
SET("chkCount" , CHECKLIST ( "COUNT", "ctrl","","chk") );
SET("chkCount1" , CHECKLIST ( "COUNT", "ctrl","","chk1"));
SENDLIST ("BLOC");
END SUB

'select 2nd item:
SUB setSelchk()
DIM lRet As Long;
CHECKLIST ( "SETSELECTEDINDEX", "ctrl","","chk", 2);
END SUB
```

CIMWAY Example

Applies To

This program shows several modes of the CIMWAY instruction.

```
SUB Main()
'------ The communication object is MODBUS.PLC1.WORD1
  DIM ret AS INTEGER;
  DIM period AS SINGLE;
  DIM errcount AS LONG;
  DIM net_stat AS INTEGER, node_stat AS INTEGER;
  DIM frame_stat AS INTEGER;

'------ Set scan rate to 500mSec
  ret = CIMWAY("SCANMDF","MODBUS.PLC1.WORD1",0.5);

'------ Test the network, node and frame Status
  net_stat = CIMWAY("STATUS","MODBUS");
  IF (net_stat ==1) THEN
    errcount = CIMWAY("ERROR","MODBUS");
    PRINT("Network out of service", errcount,"errors.");
  END IF
  node_stat = CIMWAY("STATUS","MODBUS.PLC1");
  IF (node_stat ==1) THEN
    errcount = CIMWAY("ERROR","MODBUS.PLC1");
    PRINT("Node out of service", errcount,"errors.");
  END IF
  frame_stat = CIMWAY("STATUS","MODBUS.PLC1,WORD1"); 'frame status
  IF (frame_stat ==1) THEN
    errcount = CIMWAY(5,"MODBUS.PLC1.WORD1");
    PRINT("Frame out of service", errcount,"errors.");
  END IF

'------ MODBUS connection to COM2
  ret = CIMWAY("CFG","MODBUS","MODIFY_PORT_NUMBER","abc");
  IF(ret !=0) THEN
    IF (ret == -1) THEN
    print("XBUS network do not exist");
    ELSE
      IF (ret ==-2) THEN
        print("Bad parameter");
      ELSE
        print("Unknown error ",ret );
      END IF
    END IF
  END IF

'------ Modification of the node address
  ret = CIMWAY("CFG","MODBUS.VIRTUALNODE","MODIFY_EQT_ADDRESS","10");
  IF (ret !=0) THEN
    IF (ret == -1) THEN
      print("MODBUS.VIRTUALNODE does not exist");
    ELSE
      IF (ret == -2) THEN
        print("Bad parameter");
      ELSE
        print("Unknown error");
      END IF
    END IF
  END IF

'------ Modify the address of the frame
  ret = CIMWAY("CFG","MODBUS.PLC1.WORD1","MEMORY_ADDRESS","10");
  IF (ret !=0) THEN
    IF (ret == -1) THEN
```

```
      print("MODBUS.PLC1,WORD1 does not exist");
    ELSE
      IF (ret == -2) THEN
        print("Bad parameter");
      ELSE
        print("Unknown error ",ret);
      END IF
    END IF
  END IF

'------ Start a communication object.
  print("Start MODBUS network");
  ret = CIMWAY ("START","XBUS");
  print(ret);
  print(Start node MODBUS.PLC1");
  ret = CIMWAY ("START","MODBUS.PLC1");
  print(ret);
  print(Start scanning frame MODBUS.PLC1.WORD1");
  ret = CIMWAY ("START","MODBUS.PLC1.WORD1");
  print(ret);

'------ Stop a communication object.
  print("Stop MODBUS network");
  ret = CIMWAY ("STOP","XBUS");
  print(ret);
  print(Stop node ModeBUS.PLC1");
  ret = CIMWAY ("STOP","MODBUS.PLC1");
  print(ret);
  print(Stop scanning frame MODBUS.PLC1.WORD1");
  ret = CIMWAY ("STOP","MODBUS.PLC1.WORD1");
  print(ret);

'------ Check reading after a write command
  print ("No check reading on frame MODBUS.PLC1.WORD1");
  ret = CIMWAY("READAFTERWRITE","MODBUS.AP1.MOT16","NO");
  print (ret);
END SUB
```

CMPSTRING Example

Applies To

This program uses the CMPSTRING instruction.

```
SUB Main()
'Declare variables
DIM strstring1 as Str;
DIM strstring2 as Str;
DIM intResult1 as integer;
DIM intResult2 as integer;
DIM intResult3 as integer;

strstring1="string1";
strstring2="string2";
intResult1 = CMPSTRING(strstring1,strstring2);
intResult2 = CMPSTRING(strstring2,strstring1);
intResult3 = CMPSTRING(strstring1,strstring1);
If (intResult1 <0 ) Then
    PRINT(strstring1," less than ",strstring2);
End If
If (intResult2 >0 )Then
    PRINT(strstring2," more than ",strstring1);
End If
If (intResult3 == 0 )Then
    PRINT(strstring1," same as ",strstring1);
End If
END SUB
```

COMBOBOX Example

Applies To

This example contains a set of functions to support a pair of Combo-box List form controls using the COMBOBOX instruction.

```
SUB Main()
END SUB

'declare variables:
SUB cbxSel()
DIM lRet As Long;
DIM sCbxText As Str;
DIM sCbxText1 As Str;
DIM sCbxData As Str;
DIM sCbxData1 As Str;

'retrieve contents of selected item in 1st control:
lRet = COMBOBOX ( "GETSELECTEDINDEX", "ctrl","","cbx" );
COMBOBOX ( "SETSELECTEDINDEX", "ctrl","","cbx1", lRet);
sCbxText = COMBOBOX ( "GETTEXT", "ctrl","","cbx", lRet);
sCbxText1 = COMBOBOX ( "GETTEXT", "ctrl","","cbx1", lRet);
sCbxData = COMBOBOX ( "GETUSERDATA", "ctrl","","cbx", lRet);
sCbxData1 = COMBOBOX ( "GETUSERDATA", "ctrl","","cbx1", lRet);

'prepare to put its contents in a frame:
SET ( "CbxText" , sCbxText);
SET("CbxText1", sCbxText1);
SET("CbxData", sCbxData);
SET("CbxData1" , sCbxData1);
SET("CbxCount" , COMBOBOX ( "COUNT", "ctrl","","cbx") );
SET("CbxCount1" , COMBOBOX ( "COUNT", "ctrl","","cbx1"));

'concatenate and load contents into a form control of each kind:
COMBOBOX ( "LOAD", "ctrl","","cbx2", ADDSTRING("cbx",sCbxData));
LISTBOX ( "LOAD", "ctrl","","lbx2", ADDSTRING("lbx",sCbxData));
CHECKLIST ( "LOAD", "ctrl","","chk2", ADDSTRING("chk",sCbxData));
OPTIONLIST ( "LOAD"a, "ctrl","","opt2", ADDSTRING("opt",sCbxData));
TREEVIEW ( "LOAD", "ctrl","","tvw2", ADDSTRING("tvw",sCbxData));
SENDLIST ("BLOC");
END SUB

'retrieve content of selected item in 2nd control:
SUB cbxSel1()
DIM lRet As Long;
DIM sCbxText As Str;
DIM sCbxText1 As Str;
DIM sCbxData As Str;
DIM sCbxData1 As Str;
lRet = COMBOBOX ( "GETSELECTEDINDEX", "ctrl","","cbx1" );
COMBOBOX ( "SETSELECTEDINDEX", "ctrl","","cbx", lRet);
sCbxText = COMBOBOX ( "GETTEXT", "ctrl","","cbx", lRet);
sCbxText1 = COMBOBOX ( "GETTEXT", "ctrl","","cbx1", lRet);
sCbxData = COMBOBOX ( "GETUSERDATA", "ctrl","","cbx", lRet);
sCbxData1 = COMBOBOX ( "GETUSERDATA", "ctrl","","cbx1", lRet);

'put its contents in a frame:
lRet = COMBOBOX ( "COUNT", "ctrl","","cbx1");
SET ( "CbxText" , sCbxText);
SET("CbxText1", sCbxText1);
SET("CbxData", sCbxData);
SET("CbxData1" , sCbxData1);
SET("CbxCount" , COMBOBOX ( "COUNT", "ctrl","","cbx") );
SET("CbxCount1" , COMBOBOX ( "COUNT", "ctrl","","cbx1"));
SENDLIST ("BLOC");
```

```
END SUB

'select 2nd item:
SUB setSelcbx()
DIM lRet As Long;
COMBOBOX ( "SETSELECTEDINDEX", "ctrl","","cbx", 2);
END SUB
```

CONVERT Example

Applies To

This example exercises modes of the CONVERT command and PRINTs the results to the Program Management debug pane.

```
SUB Main()
'call the functions
convertbintoa();
convertocttoa();
converthextoa();
convertbcdtoa();
convertatobin();
convertatooct();
convertatohex();
convertatobcd();
END SUB

'Mode BINTOA or Mode 1 (syntax 1)
SUB convertbintoa()
'Declare internal variables
DIM lngvalue as long;
DIM strReturn as Str;
lngvalue=20;
strReturn = CONVERT("BINTOA",lngvalue);
PRINT(lngvalue," to base 10 equals ",strReturn," to base 2");
END SUB

'Mode OCTTOA or Mode 2 (syntax 1)
SUB convertocttoa()
'Declare internal variables
DIM lngvalue as long;
DIM strReturn as Str;
lngvalue=20;
strReturn = CONVERT("OCTTOA",lngvalue);
PRINT(lngvalue," to base 10 equals ",strReturn," to base 8");
END SUB

'Mode HEXTOA or Mode 3 (syntax 1)
SUB converthextoa()
'Declare internal variables
DIM lngvalue as long;
DIM strReturn as Str;
lngvalue=20;
strReturn = CONVERT("HEXTOA",lngvalue);
PRINT(lngvalue," to base 10 equals ",strReturn," to base 16 (hexadecimal)");
END SUB

'Mode BCDTOA or Mode 4 (syntax 1)
SUB convertbcdtoa()
'Declare internal variables
DIM lngvalue as long;
DIM strReturn as Str;
lngvalue=20;
strReturn = CONVERT("BCDTOA",lngvalue);
PRINT(lngvalue," to base 10 equals ",strReturn," in BCD");
END SUB

'Mode ATOBIN or Mode 11 (syntax 2)
SUB convertatobin()
'Declare internal variables
DIM strvalue as Str;
DIM lngReturn as long;
strvalue="111111";
lngReturn = CONVERT("ATOBIN",strvalue);
```

```
PRINT(strvalue," to base 2 equals ",lngReturn," to base 10");
END SUB

'Mode ATOOCT or Mode 12 (syntax 2)
SUB convertatooct()
'Declare internal variables
DIM strvalue as Str;
DIM lngReturn as long;
strvalue="16";
lngReturn = CONVERT("ATOOCT",strvalue);
PRINT(strvalue," to base 8 equals ",lngReturn," to base 10");
END SUB

'Mode ATOHEX or Mode 13 (syntax 2)
SUB convertatohex()
'Declare internal variables
DIM strvalue as Str;
DIM lngReturn as long;
strvalue="D";
lngReturn = CONVERT("ATOHEX",strvalue);
PRINT(strvalue," to base 16 equals ",lngReturn," to base 10");
END SUB

'Mode ATOBCD or Mode 14 (syntax 2)
SUB convertatobcd()
'Declare internal variables
DIM strvalue as Str;
DIM lngReturn as long;
strvalue="D";
lngReturn = CONVERT("ATOBCD",strvalue);
PRINT(strvalue," in BCD equals ",lngReturn," to base 10");
END SUB
```

COPY_BUFFER Example

Applies To

This example shows use of the COPY_BUFFER command.

```
SUB Main()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strline as Str;
DIM strResult as Str;
DIM intResult as integer;
DIM intlength as integer;

'Create a buffer
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
strline = "123.34;a string,here;345;123456789;,t,\n";

'Put the string srtline into the buffer
PUT_BUFFER(lngbuffer1, 0, strline);

'Retrieve the length of the character string strline
intlength=LEN(strline);
intResult = COPY_BUFFER(lngbuffer2, lngbuffer1 ,0);
strResult = CGET_BUFFER(lngbuffer2,0,intlength,0);
PRINT("The returned string is: ",strResult);

'This displays:
'"The returned string is: 123.34;a string,here;345;123456789;,t,"
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB
```

CRONTAB Examples

Applies To

Example 1

This program creates or modifies events to run programs at various times.

```
SUB Main()
'-------- The 5th of August at 16.50.
  CRONTAB("ADDPROG","ONCE","05/08/04","16:50", "clicell", "","cront");
'-------- 26 minutes past each hour.
  CRONTAB("ADDPROG","EACH_HOUR","","26","clicell","", "cront");
'-------- Every day at 18.27
  CRONTAB("ADDPROG","EACH_DAY","","18:27","clicell","", "cront");
'-------- Every week on Sunday at 18.27
  CRONTAB("ADDPROG","EACH_WEEK","sun","18:28", "clicell","","cront");
'-------- The 27th of each month at 18.29
  CRONTAB("ADDPROG","EACH_MONTH","27","18:29", "clicell","","cront");
'-------- 30 minutes past each hour
  CRONTAB("ADDPROG","EACH_HOUR","","30","B1",0,0);
'-------- 30 minutes past each hour
  CRONTAB("ADDPROG","EACH_HOUR","","30", "SYSTEM.ALARM.NEWAL",2 ,3);
  CRONTAB("ADDPROG","EACH_HOUR","","30","B2",2,5);
END SUB
```

Example 2

This program sends the Scheduler configuration to all stations in the modification clients list.

```
DIM Handle As Long;
SUB SendSeq()
  Handle = alloc_buffer (1000);
  Set("EVTVAR", 0);
  SENDLIST(0);
  EVENT("ADDPROG", "EVTVAR", 1, "PGMCRON", "", "OnEvt");
  CRONTAB("NETWORKBROADCAST" , "LIST1" , Handle , "EVTVAR");
END SUB

SUB OnEvt() DIM Ret As Str;
 EVENT("DEL", "EVTVAR", 1, "PGMCRON", "", "OnEvt"); Ret = CGET_Buffer(Handle, 0, 255);
print(Ret); FREE_BUFFER(Handle); END SUB
```

CYCLIC Examples

Applies To

Example 1

This example exercises several modes of the CYCLIC command.

```
'Variables used:
'@ACTIVATION.STATE of type STATE

SUB prog1()
PRINT("The arguments passed are: ",GETARG("ARG1")," and ",GETARG("ARG2"));
END SUB

SUB prog2()
BEEP(1000,100);     '1000 Hz for 1/10 second
END SUB

'Mode ADDPROG or Mode 1 (syntax 1)

SUB cyclicaddprog()
DIM intResult as integer;
DIM intDelay as integer;
intDelay=5;
intResult = CYCLIC("ADDPROG",intDelay, "CYCLIC.SCB","","prog1", "Argument1,Argument2"
,"@ACTIVATION.STATE");
'intResult = CYCLIC("ADDPROG",intDelay, "CYCLIC.SCB","","prog2", ""
,"@ACTIVATION.STATE");
END SUB

'Mode DEL or Mode 5 (syntax 1)

SUB cyclicdel()
DIM intResult as integer;
DIM intDelay as integer;
intDelay=5;
intResult = CYCLIC("DEL",intDelay, "CYCLIC.SCB","","prog1", "Argument1,Argument2"
,"@ACTIVATION.STATE");
END SUB

'Mode DELALL or Mode 6 (syntax 2)

SUB cyclicdelall()
DIM intResult as integer;
DIM intDelay as integer;
intDelay=5;
intResult = CYCLIC("DELALL");
END SUB
```

Example 2

A program PRINTVAR.PVB contains a function PRT_MES:

```
SUB PRT_MES ()
LPrint(1,"Register =",mes1);  'output for Printer
                              'mes is a variable in the database.
END SUB
```

Another program DEF_CYCLE.PVB starts a cycle for activating PRINTVAR.PVB every 30 minutes:

```
SUB Main ()
If (Program("IS LOADED","PRINTVAR.PVB","") ==0) Then
  Program("PRELOAD","PRINTVAR.PVB","");
EndIf
CYCLIC("ADD",1800,"PRINTVAR.PVB","","PRT_MES");
END SUB
```

Once the program EF_CYCLE.PVB has run, the program PRINTVAR.PVB will run every 30 minutes.

Here is a function with delays to start and stop the cyclic functions can be included in a sub-program:

```
SUB MODIF_CYCLE(OldDly,NewDly)
CYCLIC("DEL",OldDly,"PRINTVAR.PVB","","PRT_MES");
CYCLIC("ADD",NewDly,"PRINTVAR.PVB","","PRT_MES");
END SUB
```

For instance, the function:

```
MODIF_CYCLE_MES (1800,60);
```

will be executed every minute.

DATETIME Examples

Applies To

Example 1

This example shows the use of the modes of the DATETIME command.

```
'Variables
'@DATETIME of type REGISTER
'@DATETIME.DAY of type REGISTER
'@DATETIME.MONTH of type REGISTER
'@DATETIME.YEAR of type REGISTER
'@DATETIME.HOUR of type REGISTER
'@DATETIME.MINUTE of type REGISTER
'@DATETIME.SECOND of type REGISTER
'@DATETIME.MILLISECOND of type REGISTER
'@DATETIME.WEEKDAY of type REGISTER
'@DATETIME.YEARDAY of type REGISTER
'@DATETIME.ISLEAPYEAR of type REGISTER
'A date-time is a whole number of seconds since 1st January 1980

'Mode DAY or Mode 1 (syntax 1)

SUB datetimeday()
DIM intResult as integer;
intResult = DATETIME ("DAY");
PRINT("Day = ",intResult);
@DATETIME.DAY = TOS (DATETIME ("DAY" , TOD(@DATETIME)) );
END SUB

'Mode MONTH or Mode 2 (syntax 1)

SUB datetimemonth()
DIM intResult as integer;
intResult = DATETIME ("MONTH");
@DATETIME.MONTH = TOS (DATETIME ("MONTH" , TOD(@DATETIME)) );
PRINT("Month = ",intResult);
END SUB

'Mode YEAR or Mode 3 (syntax 1)

SUB datetimeyear()
DIM intResult as integer;
intResult = DATETIME ("YEAR");
@DATETIME.YEAR= TOS (DATETIME ("YEAR" , TOD(@DATETIME)) );
PRINT("Year = ",intResult);
END SUB

'Mode HOUR or Mode 4 (syntax 1)

SUB datetimehour()
DIM intResult as integer;
intResult = DATETIME ("HOUR");
@DATETIME.HOUR = TOS (DATETIME ("HOUR" , TOD(@DATETIME)) );
PRINT("Hour = ",intResult);
END SUB

'Mode MINUTE or Mode 5 (syntax 1)

SUB datetimeminute()
DIM intResult as integer;
intResult = DATETIME ("MINUTE");
@DATETIME.MINUTE = TOS (DATETIME ("MINUTE" , TOD(@DATETIME)) );
PRINT("Minute(s) = ",intResult);
END SUB
```

```
'Mode SECOND or Mode 6 (syntax 1)

SUB datetimesecond()
DIM intResult as integer;
intResult = DATETIME ("SECOND");
@DATETIME.SECOND = TOS (DATETIME ("SECOND" , TOD(@DATETIME)) );
PRINT("Second(s) = ",intResult);
END SUB

'Mode MILLISECOND or Mode 7 (syntax 1)

SUB datetimemillisecond()
DIM intResult as integer;
intResult = DATETIME ("MILLISECOND");
@DATETIME.MILLISECOND = TOS (DATETIME ("MILLISECOND" , TOD(@DATETIME)) );
PRINT("Millisecond(s) = ",intResult);
END SUB

'Mode WEEKDAY or Mode 8 (syntax 1)

SUB datetimeweekday()
DIM intResult as integer;
intResult = DATETIME ("WEEKDAY");
@DATETIME.WEEKDAY = TOS (DATETIME ("WEEKDAY" , TOD(@DATETIME)) );
PRINT("Day of the week = ",intResult);
END SUB

'Mode YEARDAY or Mode 9 (syntax 1)

SUB datetimeyearday()
DIM intResult as integer;
intResult = DATETIME ("YEARDAY");
@DATETIME.YEARDAY = TOS (DATETIME ("YEARDAY" , TOD(@DATETIME)) );
PRINT("Day in the year = ",intResult);
END SUB

'Mode ISLEAPYEAR or Mode 10 (syntax 1)

SUB datetimeisleapyear()
DIM intResult as integer;
intResult = DATETIME ("ISLEAPYEAR");
@DATETIME.ISLEAPYEAR = TOS (DATETIME ("ISLEAPYEAR" , TOD(@DATETIME)) );
PRINT("Leap year if value =(1) else (0) value = ",intResult);
END SUB
```

Example 2

This program contains 2 modules. The first generates date-time pairs and the second PRINTs them in several formats.

```
SUB Main ()
DIM TD As Double;
TD = DateTimeValue();
PrintDateTime(TD);
TD = TD-3600000; ' Subtract 1 hour
PrintDateTime(TD);
TD = DateTimeValue(25, 12, 2003, 23, 59, 59);
PrintDateTime(TD); ' 25 December 2003
TD = DateTimeValue("01/01/2000", "");
PrintDateTime(TD); ' 01 January 2000
END SUB

SUB PrintDateTime(TD)
'------Prints a time and date in several formats.
PRINT ("td = ", td, " --> ");
```

```
PRINT (DateTimeString (td, "#D/#M/###Y - #h:#m:#s.##l"));
PRINT (DateTimeString (td));
PRINT (DateTimeString (td, "D")," \\ ",DateTimeString (td,"T"));
PRINT (DateTime ("Weekday", td), ", ",DateTime ("Yearday",td));
PRINT (DateTime("Day",td), "/",DateTime("Month",td),"/", DateTime ("Year", td));
PRINT (DateTime ("Hour", td), ":", DateTime ("Minute",td),":", DateTime ("Second", td),
".",DateTime ("Millisecond", td));
PRINT (DateTime ("IsLeapYear", td));
END SUB
```

DDE Example

Applies To

This example shows modes of the DDE command.

```
'Mode INITIATE or Mode 1 (syntax 1)

SUB ddeinitiate()
'Declare code return
DIM lngReturn as long;
DIM strServDdeName as Str;
DIM strTopicName as Str;
strServDdeName="EXCEL";
strTopicName="[FOLDER1.XLS]Sheet1";
lngReturn = DDE ("INITIATE",strServDdeName,strTopicName);
If(lngReturn==0)Then
  PRINT("Error when opening DDE channel");
End If
END SUB

'Mode TERMINATE or Mode 2 (syntax 2)

SUB ddeterminate()
'Declare code Return
DIM lngChannel as long;
DIM intResult as integer;
DIM strServDdeName as Str;
DIM strTopicName as Str;
strServDdeName="EXCEL";
strTopicName="[Folder1.XLS]Sheet1";
lngChannel = DDE ("INITIATE",strServDdeName,strTopicName);
If(lngChannel==0)Then
  PRINT("Error when opening DDE channel");
End If
intResult = DDE ("TERMINATE",lngChannel);
If(intResult==0)Then
  PRINT("Error when closing DDE channel");
End If
END SUB

'Mode TERMINATEALL or Mode 3 (syntax 3)

SUB ddeterminateall()
'Declare code Return
DIM lngChannel as long;
DIM intResult as integer;
intResult = DDE ("TERMINATEALL");
If(intResult==0)Then
  PRINT("Error when closing all DDE channels");
End If
END SUB

'Mode TIMEOUT or Mode 4 (syntax 4)

SUB ddetimeout()
'Declare code Return
DIM lngChannel as long;
DIM lngDdeTimeout as long;
DIM intResult as integer;
DIM strServDdeName as Str;
DIM strTopicName as Str;
strServDdeName="EXCEL";
strTopicName="[Folder1.XLS]Sheet1";
lngChannel = DDE ("INITIATE",strServDdeName,strTopicName);
If(lngChannel==0)Then
```

```
  PRINT("Error when opening DDE channel");
End If
lngDdeTimeout=500; ' in milliseconds =>  500ms
intResult = DDE ("TIMEOUT", lngChannel, lngDdeTimeout);
If(intResult==0)Then
  PRINT("Error when modifying DDE timeout channel");
End If
intResult = DDE ("TERMINATE",lngChannel);
If(intResult==0)Then
  PRINT("Error when closing DDE channel");
End If
END SUB

'Mode EXECUTE or Mode 5 (syntax 5)

SUB ddeexecute()
'Declare code Return
DIM lngChannel as long;
DIM intResult as integer;
DIM strDdeCmd as Str;
DIM strServDdeName as Str;
DIM strTopicName as Str;
strServDdeName="EXCEL";
'It is important to register the folder Folder2.xls
' before and in the Excel default folder
' so that Folder2.xls appears in the recent files list
strTopicName="[Folder2.XLS]Sheet1";
lngChannel = DDE ("INITIATE",strServDdeName,strTopicName);
If(lngChannel==0)Then
  PRINT("Error when opening DDE channel");
End If
strDdeCmd = "";
intResult = DDE ("EXECUTE", lngChannel, strDdeCmd);
If(intResult==0)Then
  PRINT("Error when executing the command ",strDdeCmd," on the DDE channel");
End If
intResult = DDE ("TERMINATE",lngChannel);
  If(intResult==0)Then
PRINT("Error when closing DDE channel");
End If
END SUB

'Mode REQUEST or Mode 6 (syntax 6)

SUB dderequest()
'Declare code Return
DIM lngChannel as long;
DIM intResult as integer;
DIM strDdeItem as Str;
DIM strServDdeName as Str;
DIM strTopicName as Str;
strServDdeName="EXCEL";
strTopicName="[Folder1.XLS]Sheet1";
lngChannel = DDE ("INITIATE",strServDdeName,strTopicName);
If(lngChannel==0)Then
  PRINT("Error when opening DDE channel");
End If
strDdeItem = "L1C1";'ligne nø1 et Colonne nø1 du Folder ouvert
intResult = DDE ("REQUEST", lngChannel, strDdeItem);
If(intResult==0)Then
  PRINT("Error when querying the item ",strDdeItem," on the DDE channel");
End If
intResult = DDE ("TERMINATE",lngChannel);
If(intResult==0)Then
  PRINT("Error when closing DDE channel");
```

```
End If
END SUB

'Mode POKE or Mode 7 (syntax 7)

SUB ddepoke()
'Declare code Return
DIM lngChannel as long;
DIM intResult as integer;
DIM strDdeItem as Str;
DIM strServDdeName as Str;
DIM strTopicName as Str;

'In Excel the line will be =EXCEL|[Folder2.xls]Sheet1!'L1C1'
strServDdeName="EXCEL";
strTopicName="[Folder1.XLS]Sheet1";
lngChannel = DDE ("INITIATE",strServDdeName,strTopicName);
If(lngChannel==0)Then
  PRINT("Error when opening DDE channel");
End If

strDdeItem = "L1C1";
strDdeValue = "test SV dde";
intResult = DDE ("POKE", lngChannel, strDdeItem,strDdeValue);
If(intResult==0)Then
  PRINT("Error after changing ",strDdeValue," of item "
    ,strDdeItem," on DDE channel");
End If

intResult = DDE ("TERMINATE",lngChannel);
If(intResult==0)Then
  PRINT("Error when closing DDE channel");
End If
END SUB
```

DDECONV Examples

Applies To

This example exercises modes of the DDECONV command.

```
'This program creates a DDE conversation in SV
'The conversation is called: SVEXCEL

'Mode START or Mode 0 (syntax 1)

SUB ddeconvstart()
'Declare return code
DIM intReturn as integer;
DIM strConvDdeName as Str;
strConvDdeName="SVEXCEL";
intReturn = DDECONV ("START",strConvDdeName);
If(intReturn==1)Then
    PRINT("Conversation name is invalid.");
End If
END SUB

'Mode STOP or Mode 1 (syntax 1)

SUB ddeconvstop()
'Declare return code
DIM intReturn as integer;
DIM strConvDdeName as Str;
strConvDdeName="SVEXCEL";
intReturn = DDECONV ("STOP",strConvDdeName);
If(intReturn==1)Then
    PRINT("Conversation name is invalid.");
End If
END SUB

'Mode STATUS or Mode 2 (syntax 1)

SUB ddeconvstatus()
'Declare return code
DIM intReturn as integer;
DIM strConvDdeName as Str;
strConvDdeName="SVEXCEL";
intReturn = DDECONV ("STATUS",strConvDdeName);
PRINT("Conversation status =",intReturn);
If(intReturn==1)Then
    PRINT("Conversation is absent.");
End If
END SUB

'Mode ON or Mode 3 (syntax 2)

SUB ddeconvon()
'Declare return code
DIM intReturn as integer;
intReturn = DDECONV ("ON");
If(intReturn==1)Then
    PRINT("Absent");
End If
END SUB

'Mode OFF or Mode 4 (syntax 2)

SUB ddeconvoff()
'Declare return code
DIM intReturn as integer;
intReturn = DDECONV ("OFF");
```

```
If(intReturn==1)Then
    PRINT("Absent");
End If
END SUB
```

Example 3

This SUB stops a particular DDE conversation if it is running, or starts it if it is not.

```
SUB ToggleDDE (ConversationName)
DIM Status As Integer;
Status = DDECONV("STATUS", ConversationName);
If (Status == -1) Then
Break(); 'Conversation does not exist
End If
If (Status ==1) Then
DDECONV("START", ConversationName);
Else
DDECONV("STOP", ConversationName);
End If
END SUB
```

DECLARE Examples

Applies To

Example 1

```
Declare Function GetPrivateProfileString LIB "kernel32" ALIAS "GetPrivateProfileStringA"
(lpApplicationName As Str, lpKeyName As Str, pcDefault As Str, lpReturnedString As Long,
nSize As Long, lpFileName As Str) As Long;'Private
Declare Function WritePrivateProfileString LIB "kernel32" ALIAS
"WritePrivateProfileStringA" (lpApplicationName As Str, lpKeyName As Str, lpString As
Str, lpFileName As Str) As Long;

Sub Main()
End Sub

'-------- Test function activated by mimic button
Sub Test()
  Dim lReturnHandle As long, lResWrite As Long, lResGet As Long;
  Dim cSectionName As Str, cKeyName As Str, cKeyValue As Str, cDefault As Str, cFileName
As Str;
```

```
'-------- Initialisation
  cFileName = "test.ini"; '-------- In the project's TH folder by default
  cSectionName ="SECTION";
  cKeyName ="10";
  cKeyValue ="EGGBACONCHIPS";

'-------- Try write
  lResWrite=WritePrivateProfileString (cSectionName, cKeyName, cKeyValue, cFileName);
  Print ("Return from write = ", lResWrite);

'-------- Try read
  lReturnHandle = Alloc_Buffer (255);
  Put_Buffer (lReturnHandle, 0, String (255, "0"));
  lResGet = GetPrivateProfileString (cSectionName, cKeyName , "Not Found", lReturnHandle,
255, cFileName);
  Print ("Return from get = ", lResGet);
  If (lResWrite != 0) Then
    Print (Cget_Buffer (lReturnHandle, 0, lResGet));
  End If
  Free_buffer (lReturnHandle);
End Sub
```

DIM Examples

Applies To

These examples declare large arrays of variables in an identical manner.

Example 1

```
SUB Main()
DIM t[5000] As STR;
DIM i As integer;
FOR (i =0; i<5000; i++)
  t[i] = toc(i);
  PRINT (t[i]);
NEXT
END SUB
```

The results are:

```
0
...
4085
4086
```

Example 2

```
SUB Main()
DIM t[4600] As STR;
DIM i As integer;
FOR (i =0; i<4600; i++)
  t[i] = toc(i);
  PRINT (t[i]);
NEXT
END SUB
```

The results are:

```
0
...
4598
4598
4599
```

EMAIL Example

Applies To

This example exercises the EMAIL command.

```
sub main ()
email("SEND", "EmailProfile01", "a.n.other@company.com", "", "", "EmailSender Subject
test", "EmailSender Message test");
end sub
```

where the file C\MAILCONFIG.XML has this mini configuration:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--Message Configuration File-->
<MessageConfig>
<MailGeneralSettings DefaultTemplateName="" DefaultProfileName=""
DefaultAttachmentEncoding="0" MaxAttachmentSize="1000"
ProhibitedAttachmentFileExtensions="exe,dll,vbs,js" />
<SmsGeneralSettings DefaultTemplateName="" DefaultProfileName="" />
<MessageTemplates />
<MailProfiles>
<MailProfile Name="EmailProfile01" Description="">
<MailAccounts>
<MailAccount Name="EmailAccount01" Description="" SequenceNumber="1">
<Sender Address="yourname@mycompany.com" DisplayName="" />
<ReplyTo Address="" />
<Server Type="0" Host="smtp.live.com" Port="25" DeliveryMethod="0" EncryptionMethod="1"
AuthenticationMode="1" UserName="yourname@hotmail.com" UserPassword="yourpassword" />
</MailAccount>
</MailAccounts>
</MailProfile>
</MailProfiles>
<SmsProfiles />
</MessageConfig>
```

Note

In the file content above, Microsoft's SMTP server is used. Other servers could equally be used, such as Google's Gmail with the Server Type line configured as follows:

```
<Server Type="0" Host="smtp.gmail.com" Port="587" EncryptionMethod="1"
AuthenticationMode="1" UserName=" yourname@gmail.com" UserPassword="yourpassword" />
```

EVENT Examples

Applies To

Example 1

This example exercises several modes of the EVENT command.

It calls modules 'test' and 'test2' of the Beep.svs program.

```
'Variables used:
'@STATE01 of type STATE
'@AUTHORISATION of type STATE

'Mode ADDPROG or Mode 1 (syntax 1)

SUB eventaddprog()
'Declare variable
DIM intResult as integer;

intResult = EVENT("ADDPROG","@STATE01","ALL>ALL","beep.svs","","test"
  ,"1000,100" ,"@AUTHORISATION" );
END SUB

'Mode DEL or Mode 5 (syntax 2)

SUB eventdel()
'Declare variables
DIM intResult as integer;
intResult = EVENT("DEL","@STATE01","ALL>ALL","beep.svs","","test");
END SUB

'Mode DELALL or Mode 6 (syntax 3)

SUB eventdelall()
'Declare variables
DIM intResult as integer;
intResult = EVENT("DELALL");
END SUB

'Mode ADDPROGS or Mode 11 (syntax 1)

SUB eventaddprogs()
'Declare variables
DIM intResult as integer;
intResult = EVENT("ADDPROGS","@STATE01","ALL>ALL","beep.svs","","test2"
  ,"1000,100" ,"@AUTHORISATION" );
END SUB

'Mode DELPROG or Mode 21 (syntax 4)

SUB eventdelprog()
'Declare variables
DIM intResult as integer;
intResult = EVENT("DELPROG","@STATE01","ALL>ALL","beep.svs","","test");
END SUB
```

This program (Beep.svs) is called by the one above.

```
SUB test()
DIM dblfrequency as double; 'in Hz
DIM dblduration as double;  ' in ms
dblfrequency =DVAL(GETARG("ARG1"));
dblduration = DVAL(GETARG("ARG2"));
BEEP (dblfrequency,dblduration);
END SUB
```

```
SUB test2()
PRINT("starting program test2");
END SUB
```

Example 2

When the routine DEF_EVN is executed in this example, the change of the bit 'DoorState' triggers the execution of the functions TRANS_0 and TRANS_1.

```
'Program "TEST_TR" contains two functions.
SUB TRANS_1( )
PRINT("Open The Door");
END SUB

SUB TRANS_0( )
PRINT("Shut That Door!");
END SUB

'Program "DEF_EVN" contains :
SUB MAIN( )
IF (PROGRAM("IS_LOADED","TEST_TR","")==0) THEN
PROGRAM("PRELOAD","TEST_TR","");
ENDIF
EVENT("ADDPROG","DoorState",0,"TEST_TR","","TRANS_0");
EVENT("ADDPROG","DoorState",1,"TEST_TR","","TRANS_1");
END SUB
```

EXCELTOBUF Example

Applies To

Example

```
'-------- Simple example of the use of ExcelToBuf
Sub ExcelToBuf1()
  Dim lHandle As Long;
  Dim cLineSepa As Str, cColSepa As Str;
  Dim iFirstRow As Integer, iFirstCol As Integer, iLastRow As Integer, iLastCol As
Integer;

  cLineSepa = ";";      ' Separators can only be a single char
  cColSepa = ",";
  iFirstCol = 1;        ' First and last column and row are optional
  iFirstRow = 1;
  iLastCol = 10;
  iLastRow = 2;

  lHandle = ExcelToBuf("Book1.xlsx", "Sheet1", cLineSepa, cColSepa, iFirstRow, iFirstCol,
iLastRow, iLastCol);
  BufToFile(lHandle, "Output.txt"); 'Dump to text file to check result

  Print("Buffer size was ", Seq_Buffer("LEN", lHandle));
  Free_Buffer(lHandle);End Sub
End Sub
```

EXPORT Examples

Applies To

Example

```
'-------- Generate the export "Trends" using default settings
Sub Generation()
  Export("GENERATE", "Trends");
End Sub

'-------- Generate the export "Trends" between specific dates
Sub GenerationDates()
  Dim StartDate as Double;
  Dim EndDate as Double;
  StartDate = DateTimeValue(8,1,2014,0,0,0);
  EndDate = DateTimeValue(9,1,2014,0,0,0);
  Export("GENERATE_DATES", "Trends", "", "", StartDate, EndDate, "");
End Sub

'-------- Generate the export "Trends" for a specific period
Sub GenerationPeriod()
  Dim ReferenceDate as Double;
  ReferenceDate = DateTimeValue();
  Export("GENERATE_PERIOD", "Trends", "", "", ReferenceDate, 1, 1, 2, "");
End Sub

'-------- Generate the export "Tendance" for a period derived from a Trend Viewer
Sub ExportTrend()
  Dim StartDate as double;
  Dim EndDate as double;
  StartDate = Trend("GETDATETIME", "Trends Export mimic", "", "Trend1", 1, 1);
  EndDate = Trend("GETDATETIME", "Trends Export mimic", "", "Trend1", 1, 2);
  Export("GENERATE_DATES", "Tendance", "", "", StartDate, EndDate, "");
End Sub
```

EXPORT_LOG Example

Applies To

This example exercises modes of the EXPORT_LOG command.

```
Dim cLineSeparator As  Str;
Dim cColSeparator As  Str;
Const ALARM_ON = 1, ALARM_OFF = 2, ALARM_ON_NACK = 4, ALARM_OFF_NOTACK = 8;
Const ALARM_ON_ACK = 16, ALARM_OFF_ACK = 32, ALARM_NOT_ACCESSIBLE = 64, ALARM_INHIBITED =
128;
Const ALARM_PROGRAM_MASKED = 256, ALARM_VARAIBLE_MASKED = 512, ALARM_USER_MASKED = 1024,
ALARM_EXPRESSION_MASKED = 2048;
Const ALARM_TAKEN_INTO_ACCOUNT = 4096, BIT_TO_0 = 8192, BIT_TO_1 = 16384, BIT_UNAVAILABLE
= 32768;
Const ACTION_COMMAND = 65536, ACTION_ACKNOWLEDGE = 131072, ACTION_LOGONOFF = 262144,
ACTION_EXECUTE_PROGRAM = 524288;
Const ACTION_MASK = 1048576;

Const NONE = 0, TOTAL_TRANSITION_COUNT = 1, COUNT_AT_0 = 2, COUNT_AT_1 = 4;
Const COUNT_AT_NS = 8, DUR_AT_0 = 16, DUR_AT_1 = 32, DUR_AT_NS = 64;

Const COUNT_AT_ALARM_OFF = 128, COUNT_AT_ALARM_ON = 256, COUNT_AT_ALARM_OFFACK = 512,
COUNT_AT_ALARM_OFFNOACK = 1024;
Const COUNT_AT_ALARM_ONACK = 2048, COUNT_AT_ALARM_ONNOACK = 4096,
COUNT_AT_ALARM_UNAVAILABLE = 8192, COUNT_AT_ALARM_MASKEDBYEXPRESSION = 16384;
Const COUNT_AT_ALARM_INHIBITED = 32768, COUNT_AT_ALARM_NOT_ACCESSIBLE = 65536,
COUNT_AT_ALARM_MASKEDBYPROGRAM = 131072, COUNT_AT_ALARM_MASKEDBYUSER = 262144;
Const COUNT_AT_ALARM_MASKEDBYVARIABLE = 524288;

Const DUR_AT_ALARM_OFF = 1048576, DUR_AT_ALARM_ON = 2097152, DUR_AT_ALARM_OFFACK =
4194304, DUR_AT_ALARM_OFFNOACK = 8388608;
Const DUR_AT_ALARM_ONACK = 16777216, DUR_AT_ALARM_ONNOACK = 33554432,
DUR_AT_ALARM_UNAVAILABLE = 67108864, DUR_AT_ALARM_MASKEDBYEXPRESSION = 134217728;
Const DUR_AT_ALARM_INHIBITED = 268435456, DUR_AT_ALARM_NOT_ACCESSIBLE = 536870912,
DUR_AT_ALARM_MASKEDBYPROGRAM = 1073741824, DUR_AT_ALARM_MASKEDBYUSER = 2147483648;
Const DURATION_AT_ALARM_MASKEDBYVARIABLE = 4294967296;

Sub Main ()
'-------- Initialise global variables
  cLineSeparator = Chr(13);
  cColSeparator = "'";
'-------- Set up event that is triggered when Export_trend is complete
  Event("ADDPROG", "SVB.ExportLogStatus", "S>S", "E.svb", "", "OnChangeOfStatus");
End Sub

'-------- Export_Log of records
Sub Export_Log_EX1()
  Dim dStartTime As  Double, dEndTime  As  Double;
  Dim iResult As  Integer;
  Dim llEvents As  LongLong;

  dEndTime = DateTimeValue();
  dStartTime = dEndtime - 3600000; ' Current time less 1 hour
  llEvents = ALARM_ON + ALARM_OFF + BIT_TO_0 + BIT_TO_1;
  iResult = Export_Log("GETRECORD", "Loglist01", dStartTime, dEndTime, llEvents, 0, 29,
"", "@SVB.ExportLogStatus", cLineSeparator, cColSeparator, 0, "#D/#M/#Y|#h:#m:#s|#E|#T",
"Date|Time|Event|Title");

'-------- Check return and, if a previous request is still running, cancel it and clear
buffer
  If(iResult != 0) Then
    Print("Result != 0, historic request not sent");
    If(iResult == -30) Then
      Export_Log("CANCEL");
      Export_Log("DISPOSE");
```

```
    End If
  End If
End Sub

'-------- Export_Log of statistics
Sub Export_Log_EX2()
  Dim dStartTime As  Double, dEndTime As  Double;
  Dim iResult As  Integer;
  Dim llEvents As  LongLong , llStatFlag As  LongLong ;

  dEndTime = DateTimeValue(); dStartTime = dEndtime - 3600000; ' Current time less 1 hour
  llEvents = ALARM_ON + ALARM_OFF + BIT_TO_0 + BIT_TO_1;
  llStatFlag = DUR_AT_0 + DUR_AT_1 + DUR_AT_NS;
  iResult = Export_Log("GETSTATISTIC", "Loglist01", dStartTime, dEndTime, llEvents, 0,
29, "", "@SVB.ExportLogStatus", cLineSeparator, cColSeparator, 0, 0, llStatFlag, 0, 1,
0);

'-------- Check return and, if a previous request is still running, cancel it and clear
buffer
  If(iResult != 0) Then
    Print("Result != 0, historic request not sent");
    If(iResult == -30) Then
      Export_Log("CANCEL");
      Export_Log("DISPOSE");
    End If
  End If
End Sub

'-------- On change of status variable process if complete else clean up
SubOnChangeOfStatus()
  Dim sStatus As  Single;
  Dim lBufferHandle As  Long;
  Dim iResult As  Integer, iLineNumber As  Integer;

  sStatus = SVB.ExportLogStatus;
  If(sStatus==0 || sStatus==4 || sStatus ==5) Then  ' Export complete or reached limits
    Print("Export_Log complete - status is ", sStatus);
    lBufferHandle = Alloc_Buffer(1024);
    iResult = 1;
    iLineNumber = 1;
    While(iResult > 0)
      iResult = Export_Log("READBUFFER", lBufferHandle);
      If (iResult> 0) Then
        BufToExcel(lBufferHandle, cLineSeparator, cColSeparator, "export.xlsx", "ex1",
"APPEND", iLineNumber );
        iLineNumber = iLineNumber + iResult;
        Print("Exported ",iResult, " lines to Excel");
      End If
    Delay(1); ' Allow program thread to break in case of overlong processing time
    Wend
    Print("Total lines = ",iLineNumber);
    Export_Log("DISPOSE");
    Free_Buffer(lBufferHandle);
  Else
    If(sStatus==1) Then  ' Export running - do nothing
      Print("Export_Log running");
    Else  ' Export failed - cancel and clean up
      Print("Export_Log failed - status is ", sStatus);
      Export_Trend("CANCEL");
      Export_Trend("DISPOSE");
    End If

  End If
End Sub
```

EXPORT_TREND Example

Applies To

This example exercises modes of the EXPORT_TREND command.

```
Dim cLineSeparator As  Str;
Dim cColSeparator As  Str;
Const MIN = 1, MAX = 2, AVE = 4, SUM = 8;
Const DEVIATION = 16, FIRST = 32, LAST = 64, COUNTER = 128;
Const MINDATE = 256, MAXDATE = 512, FIRSTDATE = 1024, LASTDATE = 2048;
Const WEIGHTED = 4096;

Sub Main ()
'-------- Initialise global variables
  cLineSeparator = Chr(13);
  cColSeparator = "'";
'-------- Set up event that is triggered when Export_trend is complete
  Event("ADDPROG", "SVB.Status", "S>S", "E.svb", "", "OnChangeOfStatus");
End Sub

'-------- Export_Trend of raw values
Sub Export_Trend_EX1()
  Dim dStartTime As  Double, dEndTime As  Double;
  Dim iResult As  Integer;

  dEndTime = DateTimeValue();
  dStartTime = dEndtime - 3600000; ' Current time less 1 hour
  iResult = Export_Trend("GETRAW", "SVB.Register01", "", "Unit01", dStartTime, dEndTime,
"SVB.Status", cLineSeparator, cColSeparator, 0);

'-------- Check return and, if a previous request is still running, cancel it and clear
buffer
  If(iResult != 0) Then
    Print("Result != 0, historic request not sent");
    If(iResult == -30) Then
      EXPORT_TREND("CANCEL");
      EXPORT_TREND("DISPOSE");
    End If
  End If
End Sub

'-------- Export_Trend of sampled values
Sub Export_Trend_EX2()
  Dim dStartTime As  Double, dEndTime As  Double;
  Dim iResult As  Integer;
  dEndTime = DateTimeValue();
  dStartTime = dEndtime - 3600000; ' Current time less 1 hour
  iResult = Export_Trend("GETSAMPLE", "SVB.Register01,SVB.Register02", "", "Unit01",
dStartTime, dEndTime, "SVB.Status", cLineSeparator, cColSeparator, 0, 0, 1, 1);

'-------- Check return and, if a previous request is still running, cancel it and clear
buffer
  If(iResult != 0) Then
    Print("Result != 0, historic request not sent");
    If(iResult == -30) Then
      Export_Trend("CANCEL");
      Export_Trend("DISPOSE");
    End If
  End If
End Sub

'-------- Export_Trend of statistical values
Sub Export_Trend_EX3()
```

```
  Dim dStartTime As  Double, dEndTime As  Double, dStatFlag As  Double;
  Dim iResult As  Integer;

  dStatFlag = MIN + MAX + AVE + SUM + DEVIATION + FIRST + LAST + COUNTER + WEIGHTED; '
Add/delete stats as required
  dEndTime = DateTimeValue();
  dStartTime = dEndtime - 3600000; ' Current time less 1 hour
  iResult = Export_Trend("GETSTATISTIC", "SVB.Register01,SVB.Register02", "", "Unit01",
dStartTime, dEndTime, "SVB.Status", cLineSeparator, cColSeparator, 0, dStatFlag );

'-------- Check return and, if a previous request is still running, cancel it and clear
buffer
  If(iResult != 0) Then
    Print("Result != 0, historic request not sent");
    If(iResult == -30) Then
      Export_Trend("CANCEL");
      Export_Trend("DISPOSE");
    End If
  End If
End Sub

'-------- Export_Trend of aggregated values
Sub Export_Trend_EX4()
  Dim dStartTime As  Double, dEndTime As  Double, dStatFlag As  Double;
  Dim iResult As  Integer;

  dStatFlag = MIN + MAX + AVE + SUM; ' Add/delete stats as required
  dEndTime = DateTimeValue();
  dStartTime = dEndtime - 3600000; ' Current time less 1 hour
  iResult = Export_Trend("GETAGGREGATED", "SVB.Register01,SVB.Register02", "", "Unit01",
dStartTime, dEndTime, "SVB.Status", cLineSeparator, cColSeparator, 0, 0, dStatFlag, 1,
1);
'-------- Check return and, if a previous request is still running, cancel it and clear
buffer
  If(iResult != 0) Then
    Print("Result != 0, historic request not sent");
    If(iResult == -30) Then
      Export_Trend("CANCEL");
      Export_Trend("DISPOSE");
    End If
  End If
End Sub

'-------- On change of status variable process if complete else clean up
SubOnChangeOfStatus()
  Dim sStatus As  Single;
  Dim lBufferHandles  Long;
  Dim iResult As  Integer, iLineNumber As  Integer;

  sStatus = SVB.Status;
  If(sStatus==0 || sStatus==4 || sStatus ==5) Then  ' Export complete or reached limits
    Print("Trend_Export complete - status is ", sStatus);
    lBufferHandle = Alloc_Buffer(1024);
    iResult = 1;
    iLineNumber = 1;
    While(iResult > 0)
      iResult = Export_Trend("READBUFFER", lBufferHandle);
      If (iResult> 0) Then
        BufToExcel(lBufferHandle, cLineSeparator, cColSeparator, "export.xlsx", "ex1",
"APPEND", iLineNumber );
        iLineNumber = iLineNumber + iResult;
        Print("Exported ",iResult, " lines to Excel");
      End If
    Wend
    Print("Total lines = ",iLineNumber);
```

```
    Export_Trend("DISPOSE");
    Free Buffer(lBufferHandle);
  Else
    If(sStatus==1) Then  ' Export running - do nothing
      Print("Trend_Export running");
    Else  ' Export failed - cancel and clean up
      Print("Trend_Export failed - status is ", sStatus);
      Export_Trend("CANCEL");
      Export_Trend("DISPOSE");
    End If
  End If
End Sub
```

EXPRESSION Example

Applies To

This example exercises several modes of the EXPRESSION command.

```
'Variables used:
'@STATE01 of type STATE
'@STATE02 of type STATE
'@AUTHORISATION of type STATE
'@REGISTER01 of type REGISTER
'@REGISTER02 of type REGISTER

'contents of a file model:
'EXPRM,"MODEL 01","STATE01 or STATE02",,,0,0,"expression model no01"
'EXPRM,"MODEL 02","REGISTER01 + REGISTER02",,,0,0,"expression model no02"
'contents of a file model:
'EXPRV,@STATE02,NOT @STATE01 ,,,0,0,comment expression01
'EXPRV,@REGISTER02,@REGISTER01 + 21,,,0,0, register variables

'Mode IMPORT_FILE_TEMP or Mode 0 (syntax 2)

SUB expressionimportfiletemp()
'Declare variables
DIM intResult as integer;
DIM strString01 as STR;
strString01 = "file model expressions.txt";
intResult = EXPRESSION("IMPORT_FILE_TEMP",strString01);
END SUB

'Mode IMPORT_BUFFER_TEMP or Mode 1 (syntax 1)

SUB expressionimportbuffertemp()
'Declare variables
DIM lngBuffer as long;
DIM intResult as integer;
DIM strString01 as STR;
strString01 = "EXPRM,MODEL 01,STATE01 or STATE02,,,0,0,expression model no01";
lngBuffer = ALLOC_BUFFER(1000);

'Put the strString01 into the buffer
PUT_BUFFER(lngbuffer,0,strString01);
intResult = EXPRESSION("IMPORT_BUFFER_TEMP",lngBuffer);
FREE_BUFFER(lngBuffer);
END SUB

'Mode IMPORT_FILE_ONVAR or Mode 2 (syntax 2)

SUB expressionimportfileonvar()
'Declare variables
DIM intResult as integer;
DIM strString01 as STR;
strString01 = "file expressions.txt";
intResult = EXPRESSION("IMPORT_FILE_ONVAR",strString01);
END SUB

'Mode IMPORT_BUFFER_ONVAR or Mode 3 (syntax 1)

SUB expressionimportbufferonvar()
'Declare variables
DIM lngBuffer as long;
DIM intResult as integer;
DIM strString01 as STR;
strString01 = "EXPRV,@STATE02,NOT @STATE01 ,,,0,0,comment expression01";
lngBuffer = ALLOC_BUFFER(1000);
```

```
'Put the string strString01 into the buffer
PUT BUFFER(lngbuffer,0,strString01);
intResult = EXPRESSION("IMPORT_BUFFER_ONVAR",lngBuffer);
FREE_BUFFER(lngBuffer);
END SUB
```

FCLOSE and FOPEN Example

Applies To

This example exercises the file opening and closing commands.

To run, it needs a text file "util.txt" to be stored in the TP folder of the project.

```
'This program uses a file "util.txt" that is stored in the TP folder of the project
SUB Main()
DIM intResult as integer;
DIM intEOF as integer;
DIM StrAccess as Str;
DIM StrFilename as Str;
DIM StrLine as Str;
StrFilename = "util.txt"; 'name of the file to open in the TP folder

'For a full path put "\\" instead of "\" => "C:\\SV\\util.txt" '
StrAccess = "r"; 'opening in  Read mode

' opening the file intResult = FOPEN(StrFilename , StrAccess);
If (intResult ==1) Then
    PRINT("Opening of file ", StrFilename," succeeded");
End If

'closing the file intResult = FCLOSE(StrFilename);
If (intResult ==1) Then
    PRINT("Closure of file ", StrFilename," succeeded");
End If
END
```

FCOPY Example

Applies To

This example exercises the FCOPY command.

```
'-------- Copy file from source to destination
Sub FCOPY_EX1()
  Dim cSource As Str, cDestination As Str;
  Dim iReturn As Integer;
  cSource = "MyFile.dat";
  cDestination = "NewFile.dat";
  iReturn = Fcopy(cSource, cDestination);
End Sub

'-------- Copy file from source to destination overwriting destination if it already
exists
Sub FCOPY_EX2()
  Dim cSource As Str, cDestination As Str;
  Dim iReturn As Integer;
  cSource = "MyFile.dat";
  cDestination = "NewFile.dat";
  iReturn = Fcopy(cSource, cDestination, 1);
End Sub
```

FMOVE Example

Applies To

This example exercises the FMOVE command.

```
'-------- Move file from source to destination
Sub FMOVE_EX1()
  Dim cSource As Str, cDestination As Str;
  Dim iReturn As Integer;
  cSource = "MyFile.dat";
  cDestination = "NewFile.dat";
  iReturn = FMOVE(cSource, cDestination);
End Sub

'-------- Move file from source to destination overwriting destination if it already
exists
Sub FMOVE_EX2()
  Dim cSource As Str, cDestination As Str;
  Dim iReturn As Integer;
  cSource = "MyFile.dat";
  cDestination = "NewFile.dat";
  iReturn = FMOVE(cSource, cDestination, 1);
End Sub
```

FORMAT Example

Applies To

This example shows how the FORMAT command displays various text strings.

```
SUB MAIN ()
  DIM count as integer;
  DIM i as integer;
  DIM ch97 as integer;
  DIM ch122 as integer;
  DIM s as str;
  DIM fp as double;

  count = -9234;
  i = 1;
  ch97 = 97;
  ch122 = 122;
  s="Hello world!";
  fp = 251.7366;

'-------- Display integer
  PRINT("Display integer");
  PRINT(FORMAT("Decimal: %d", count)); 'Decimal: -9234
  PRINT(FORMAT("Decimal: %i", count)); 'Decimal: -9234
  PRINT(FORMAT("Justified: %.6d", count)); 'Justified: -009234
  PRINT(FORMAT("Unsigned: %u", count)); ' Unsigned: 4294958062
  PRINT(Format("Short integer: %.2d", i)); 'Decimal: 01

'-------- Display hexadecimal
  PRINT("Display hexadecimal");
  PRINT(FORMAT("Hex: %Xh", count)); 'Hex: FFFFDBEEh
  PRINT(FORMAT("hex: 0x%x", count)); 'hex: 0xffffdbee

'-------- Display octal
  PRINT("Display octal");
  PRINT(FORMAT("Octal: %o", count)); 'Octal: 37777755756

'-------- Display characters
  PRINT("Display characters");
  PRINT(FORMAT("Character code 97: %c", ch97)); 'a
  PRINT(FORMAT("Character code 97: %C", ch97)); 'a
  PRINT(FORMAT("Character code 122: %c", ch122)); 'z

'-------- Display string / text
  PRINT("Display string / text");
  PRINT(FORMAT("%s", s)); 'Hello world!
  PRINT(FORMAT("%.9s", s)); 'Hello wor

'-------- Display real number
  PRINT("Real numbers:");
  PRINT(FORMAT("%f", fp)); '251.736600
  PRINT(FORMAT("%.2f ",fp)); '251.74
  PRINT(FORMAT("%e", fp)); '2.517366e+002
  PRINT(FORMAT("%E", fp)); '2.517366E+002
END SUB
```

FORMULA Example

Applies To

This example exercises modes of the FORMULA command.

It needs a text file "importFormula.txt" to be in the TP folder, formatted like FORMULA.DAT.

For a full path use "\\" instead of "\", eg "C:\\SV\\util.txt".

Always put a return character at the end of each line in FORMULA.DAT, else the formula is ignored.

```
'Variables used in the formulas:
'@BRCH1.REGISTER01 of type REGISTER
'@BRCH2.REGISTER01 of type REGISTER
'@BRCH3.REGISTER01 of type REGISTER
'@BRCH1.REGISTER02 of type REGISTER
'@BRCH2.REGISTER02 of type REGISTER
'@BRCH3.REGISTER02 of type REGISTER
'@BRCH1.REGISTER03 of type REGISTER
'@BRCH2.REGISTER03 of type REGISTER
'@BRCH3.REGISTER03 of type REGISTER

'Mode ADD or Mode 1 (syntax 1)

SUB formulaadd()
'Declare variables
DIM lngbuffer1 as long;
DIM intResult as integer;
DIM strFilename as Str;
strFilename = "Formula.txt";
lngbuffer1 = filetobuf(strFilename);
intResult = FORMULA("ADD",lngbuffer1);
FREE_BUFFER(lngbuffer1);
END SUB

'Mode ENABLE or Mode 2 (syntax 2)

SUB formulaenable()
'Declare variables
DIM intResult as integer;
intResult = FORMULA("ENABLE", "formula01", "BRCH1");
END SUB

'Mode DISABLE or Mode 3 (syntax 2)

SUB formuladisable()
'Declare variables
DIM intResult as integer;
intResult = FORMULA("DISABLE", "formula01", "BRCH1");
END SUB

'Mode DEL or Mode 4 (syntax 2)

SUB formuladel()
'Declare variables
DIM intResult as integer;
intResult = FORMULA("DEL", "formula01", "BRCH1");
END SUB

'Mode DELALL or Mode 4 (syntax 2)

SUB formuladelall()
'Declare variables
```

```
DIM intResult as integer;
intResult = FORMULA("DELALL");
END SUB

'Mode DELALL or Mode 4 (syntax 2)

SUB formuladelall2()
'Declare variables
DIM intResult as integer;
'intResult = FORMULA("DELALL",3); 'Delete computational formulas
intResult = FORMULA("DELALL",14);
END SUB
```

FPUTC Example

Applies To

This example exercises modes of the FPUTC command.

For a full path use "\\" instead of "\", e.g. "C:\\SV\\tool.txt".

To run, it needs a text file "tool.txt" to be stored in the TP folder of the project.

```
SUB Main()
DIM intResult as INTEGER;
DIM intEOF as INTEGER;
DIM StrAccess as STR;
DIM StrFilename as STR;
DIM StrChar1 as STR;

StrFilename = "doc1.txt"; 'name of the file to open in the TP folder
StrAccess = "w+"; '  opening in Write mode

' opening of the file
intResult = FOPEN(StrFilename , StrAccess);
If (intResult ==1) Then
    PRINT("Opening the file ", StrFilename," succeeded");
End If
StrChar1= "A";
intResult = FPUTC(StrFilename,StrChar1);
If (intResult ==1) Then
    PRINT("Writing succeeded.");
End If

'Closing the file
intResult = FCLOSE(StrFilename);
If (intResult ==1) Then
    PRINT("Closing of the file ", StrFilename," succeeded");
End If
END SUB
```

FPUTS Example

Applies To

This example exercises modes of the FPUTS command.

For a full path use "\\" instead of "\", eg "C:\\SV\\util.txt".

To run, it needs a text file "tool.txt" to be stored in the TP folder of the project.

```
SUB main()
DIM intResult as integer;
DIM intEOF as integer;
DIM StrAccess as Str;
DIM StrFilename as Str;
DIM StrChar1 as Str;
StrFilename = "tool.txt";
'name of the file to open in the TP folder

StrAccess = "w+"; '  opening in Write mode

' opening the file
intResult = FOPEN(StrFilename , StrAccess);
If (intResult ==1) Then
    PRINT("Opening of the file ", StrFilename," succeeded");
End If

StrChar1= "Hello, World!";
intResult = FPUTS(StrFilename,StrChar1);
If (intResult ==1) Then
    PRINT("Writing succeeded");
End If

'Closing of the file
intResult = FCLOSE(StrFilename);
If (intResult ==1) Then
    PRINT("Closing of the file ", StrFilename," succeeded");
End If
END SUB
```

FSEEK Example

Applies To

This example exercises of modes of the FSEEK command.

For a full path use "\\" instead of "\".

To run, it needs a text file "tool.txt" to be stored in the TP folder of the project.

```
SUB Main()
DIM intResult as INTEGER;
DIM intEOF as INTEGER;
DIM StrAccess as STR;
DIM StrFilename as STR;
DIM STRLine as STR;

StrFilename = "tool.txt"; 'name of file to open in TP folder
StrAccess = "r"; '  opening in  Read mode

' opening the file
intResult = FOPEN(StrFilename , StrAccess);
If (intResult ==1) Then
    PRINT("Opening of the file ", StrFilename," succeeded");
End If
PRINT(FGETC (StrFilename,2));
FSEEK(StrFilename,3,0);
PRINT(FGETC (StrFilename,2));

' closing the file
intResult = FCLOSE(StrFilename);
If (intResult ==1) Then
    PRINT("Closing of file ", StrFilename," succeeded");
End If
END SUB
```

FTP Example

Applies To

This program copies a file from and then to a file transfer site.

```
Sub Down ()
 Dim cFTPsource As Str, cLocalDestn As Str;
 Dim cName As Str, cPass As Str, cStatusVar As Str;
 Dim iReturn As Integer;

 cFTPsource = "ftp://ftp.xxxxxxxx.com/test.txt";
 cLocalDestn = "test.txt";
 cName = "xyz123";
 cPass = "abc890";
 cStatusVar = "FTPSTATUS";

 iReturn = FTP("DOWNLOAD", cFTPsource, cLocalDestn, 1, cName, cPass, cStatusVar);
 Print ("Return from FTP download is "iReturn);
End Sub

Sub Up ()
 Dim cFTPdestn As Str, cLocalSource As Str;
 Dim cName As Str, cPass As Str, cStatusVar As Str;
 Dim iReturn As Integer;

 cFTPdestn = "ftp://ftp.xxxxxxxx.com/test.txt";
 cLocalSource = "test.txt";
 cName = "xyz123";
 cPass = "abc890";
 cStatusVar = "FTPSTATUS";

 iReturn = FTP("UPLOAD", cLocalSource, cFTPdestn, 1, cName, cPass, cStatusVar);
 Print ("Return from FTP upload is "iReturn);
End Sub
```

GETARG Example

Applies To

Example 1

This example uses each mode of the GETARG command.

```
'variables
'TEXT01 type text
'TEXT02 type TEXT
'TEXT03 type TEXT
'TEXT04 type TEXT

SUB Main()
'How the example works: a branch must be specified
getargmainbranch();
END SUB

'Mode MAINBRANCH or Mode 0 (syntax 1)

'retrieve the branch of the calling program
SUB getargmainbranch()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("MAINBRANCH");
@text01 = strReturn;
END SUB

'Mode ARG1 or Mode 1 (syntax 1)

SUB getargarg1()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG1");
@text01 = strReturn;
END SUB

'Mode ARG2 or Mode 2 (syntax 1)

SUB getargarg2()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG2");
@text01 = strReturn;
END SUB

'Mode ARG3 or Mode 3 (syntax 1)

SUB getargarg3()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG3");
@text01 = strReturn;
END SUB

'Mode ARG4 or Mode 4 (syntax 1)

SUB getargarg4()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG4");
@text01 = strReturn;
END SUB

'Mode ARG5 or Mode 5 (syntax 1)
```

```
SUB getargarg5()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG5");
@text01 = strReturn;
END SUB

'Mode ARG6 or Mode 6 (syntax 1)

SUB getargarg6()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG6");
@text01 = strReturn;
END SUB

'Mode ARG7 or Mode 7 (syntax 1)

SUB getargarg7()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG7");
@text01 = strReturn;
END SUB

'Mode ARG8 or Mode 8 (syntax 1)

SUB getargarg8()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG8");
@text01 = strReturn;
END SUB

'Mode SOURCE or Mode 10 (syntax 1)

SUB getargsource()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("SOURCE");
@text01 = strReturn;
END SUB

'Mode PROGRAM or Mode 11 (syntax 1)

SUB getargprogram()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("PROGRAM");
@text01 = strReturn;
END SUB

'Mode BRANCH or Mode 12 (syntax 1)

SUB getargbranch()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("BRANCH");
@text01 = strReturn;
END SUB

'Mode FUNCTION or Mode 13 (syntax 1)

SUB getargfunction()
```

```
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("FUNCTION");
@text01 = strReturn;
END SUB

'Mode WINDOW or Mode 14 (syntax 1)

SUB getargwindow()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("WINDOW");
@text01 = strReturn;
END SUB

'Mode WBRANCH or Mode 15 (syntax 1)

SUB getargwbranch()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("WBRANCH");
@text01 = strReturn;
END SUB

'Mode IDENTIFIER or Mode 16 (syntax 1)

SUB getargidentifier()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("IDENTIFIER");
@text01 = strReturn;
END SUB

'Mode VARNAME or Mode 17 (syntax 1)

SUB getargvarname()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("VARNAME");
@text01 = strReturn;
END SUB

'Mode VARVALUE or Mode 18 (syntax 1)

SUB getargvarvalue()
'Declare the return code
DIM sngReturn as Single;
sngReturn = GETARG("VARVALUE");
@text02 = TOC(sngReturn);
END SUB

'Mode VARSTATUS or Mode 19 (syntax 1)
SUB getargvarstatus()
'Declare the return code
DIM Current_Region as integer;
Current_Region = GETARG("VARSTATUS");
END SUB

END SUB

'Mode KEYTYPE or Mode 20 (syntax 1)

SUB getargkeytype()
'Declare the return code
DIM strReturn as STR;
```

```
strReturn = GETARG("KEYTYPE");
@text04 = strReturn;
'in the example: SC => SHIFT+CTRL
END SUB

'Mode KEYCODE or Mode 21 (syntax 1)

SUB getargkeycode()
'Declare the return code
DIM intReturn as integer;
intReturn = GETARG("KEYCODE");
PRINT(intReturn);
@text03 = TOC(intReturn);
'In the example: 3
END SUB

'Mode CRONTYPE or Mode 22 (syntax 1)

SUB getargcrontype()
'Declare the return code
DIM intReturn as integer;
intReturn = GETARG("CRONTYPE");
PRINT(intReturn);
@text01 = TOC(intReturn);
END SUB

'Mode CRONDATE or Mode 23 (syntax 1)

SUB getargcrondate()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("CRONDATE");
PRINT(strReturn);
@text02 = strReturn;
END SUB

'Mode CRONTIME or Mode 24 (syntax 1)

SUB getargcrontime()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("CRONTIME");
PRINT(strReturn);
@text03 = strReturn;
END SUB

SUB getargcron()
DIM strReturn as STR;
DIM intReturn as integer;
intReturn = GETARG("CRONTYPE");
PRINT(intReturn);
@text01 = TOC(intReturn);
strReturn = GETARG("CRONDATE");
PRINT(strReturn);
@text02 = strReturn;
strReturn = GETARG("CRONTIME");
PRINT(strReturn);
@text03 = strReturn;
END SUB


'Mode ARG9 or Mode 25 (syntax 1)

SUB getargarg9()
'Declare the return code
```

```
DIM strReturn as STR;
strReturn = GETARG("ARG9");
@text01 = strReturn;
END SUB

'Mode ARG10 or Mode 26 (syntax 1)

SUB getargarg10()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG10");
@text01 = strReturn;
END SUB

'Mode ARG11 or Mode 27 (syntax 1)

SUB getargarg11()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG11");
@text01 = strReturn;
END SUB

'Mode ARG12 or Mode 28 (syntax 1)

SUB getargarg12()
'Declare the return code
DIM strReturn as STR;
strReturn = GETARG("ARG12");
@text01 = strReturn;
END SUB
```

Example 2

This example displays the calling arguments of the GETARG instruction:

```
SUB func()
 ' Display of all the calling arguments
 PRINT("MAINBRANCH = \"",GETARG("MAINBRANCH"),"\"");
 PRINT("ARG1 = \"",GETARG("ARG1"),"\"");
 PRINT("ARG2 = \"",GETARG("ARG2"),"\"");
 PRINT("ARG3 = \"",GETARG("ARG3"),"\"");
 PRINT("ARG4 = \"",GETARG("ARG4"),"\"");
 PRINT("ARG5 = \"",GETARG("ARG5"),"\"");
 PRINT("ARG6 = \"",GETARG("ARG6"),"\"");
 PRINT("ARG7 = \"",GETARG("ARG7"),"\"");
 PRINT("ARG8 = \"",GETARG("ARG8"),"\"");
 PRINT("SOURCE = \"",GETARG("SOURCE"),"\"");
 PRINT("PROGRAM = \"",GETARG("PROGRAM"),"\"");
 PRINT("BRANCH = \"",GETARG("BRANCH"),"\"");
 PRINT("FUNCTION = \"",GETARG("FUNCTION"),"\"");
 PRINT("WINDOW = \"",GETARG("WINDOW"),"\"");
 PRINT("WBRANCH = \"",GETARG("WBRANCH"),"\"");
 PRINT("IDENTIFIER = \"",GETARG("IDENTIFIER"),"\"");
 PRINT("VARNAME = \"",GETARG("VARNAME"),"\"");
 PRINT("VARVALUE = \"",GETARG("VARVALUE"),"\"");
 PRINT("VARSTATUS = \"",GETARG("VARSTATUS"),"\"");
 PRINT("KEYTYPE = \"",GETARG("KEYTYPE"),"\"");
 PRINT("KEYCODE = \"",GETARG("KEYCODE"),"\"");
 PRINT("CRONTYPE = \"",GETARG("CRONTYPE"),"\"");
 PRINT("CRONDATE = \"",GETARG("CRONDATE"),"\"");
 PRINT("CRONTIME = \"",GETARG("CRONTIME"),"\"");
END SUB
```

HARDCOPY Example

Applies To

This program exercises several modes of the HARDCOPY command.

```
'Variables:
'@ACTIVATION.STATE of Type STATE

'Mode SCREEN or Mode 1 (syntax 1)

SUB hardcopyscreen()
DIM intResult as integer;
intResult = HARDCOPY("SCREEN");
END SUB

'Mode OPTION (mode 2) (syntax 1)

SUB hardcopyoption()
DIM intResult as integer;
intResult = HARDCOPY("OPTION",1,1);  'Fit to the page
'intResult = HARDCOPY("OPTION",1,2); 'Full use of page
'intResult = HARDCOPY("OPTION",1,3); 'Scale factor mode
'intResult = HARDCOPY("OPTION",2,2); 'Reduced by 50% X
'intResult = HARDCOPY("OPTION",3,2); 'Reduced by 50% Y
'intResult = HARDCOPY("OPTION",5,"Hardcopy colour");  'Box title
'intResult = HARDCOPY("OPTION",6,"Copying in progress"); 'Message
'intResult = HARDCOPY("OPTION",9,"Hardcopy done"); 'Button text
'intResult = HARDCOPY("SCREEN");
END SUB

'Mode PREVIOUSWINDOW (mode 4) (syntax 1)

SUB hardcopypreviouswindow()
DIM intResult as integer;
intResult = HARDCOPY("PREVIOUSWINDOW");
END SUB
```

For mode 4, the window that has lost focus must not be hidden or covered by a window with focus.

HISTORY Example

Applies To

Example 1

This program "EXTRACT1" extracts historic data to a text file:

```
SUB Main ()
  DIM ret as long;
  DIM hd as double;
  ' Loading of "EXTRACT2" if not loaded :
  If (Program ("Is_loaded", "EXTRACT2", "") == 0) Then
    Program ("Preload", "EXTRACT2", "");
End if;
```

```
' Programing of execution of function "EndExtract"
' of the program "EXTRACT2" when bit switches to 1
' "BitExtract" :
Event ( "ADD", "BitExtract", 1, "EXTRACT2", "","EndExtract");
' Export of the trend variable "Valve1" in the
' file "C:\EXTRACT.TXT". The request for export
' is only valid for values archived in the current
' hour. The internal variables "BitExtract"
' (bit) and "ResultExtract" (register) must exist.
hd = DateTimeValue ();
ret = HISTORY ( "GetTrend", "Valve1", hd -3600000,hd, "C:\\EXTRACT.TXT", 0,
    "#D/#M/###Y,#h:#m:#s.##l, #V", "BitExtract", "ResultExtract" );
If (ret == 0) Then
Print ("Error on GetTrend: ");
Event ("DEL", "EndExtract", 1);
Exit SUB;
End If
END SUB

'Program "EXTRACT2"
SUB EndExtract ()
Print ("Export of variable Valve1 completed:");
If (ResultExtract < 0) Then
Print ("Error code = ", ResultExtract);
Else
Print (ResultExtract, "value(s) exported.");
End If
' Cancellation of the release :
Event ("DEL", "BitExtract", 1);
END SUB
```

Example of file "C:\EXTRACT.TXT" after export:

```
23/03/1994, 14:01:03.130, ?
23/03/1994, 14:04:45.520, 12.3
23/03/1994, 14:07:11.890, 11.9
23/03/1994, 14:12:52.460, 11.7
23/03/1994, 14:19:02.780, 11.4
23/03/1994, 14:25:59.340, 11
23/03/1994, 14:40:44.520, 11.2
23/03/1994, 14:56:31.200, 11.5
```

The maximum number of values that can be exported at one time is 4,000.

To export more than 4000 values the HISTORY instruction can be repeated, thus appending the results to the file.

Example 2

These programs import historic data:

```
'Program "IMPORT1"
' Set up event for change of the Count variable
SUB Main()
Event("DELALL");
Event("ADD", "IMPORT.COUNT.MV1", "ALL>S", "Import.pvb", "","CountMV1");
END SUB
```

```
' Import Example 1. FORMAT and VARNAME keywords are
' contained in the import file
SUB LoadMV1()
HISTORY("IMPORTTREND","MV1.DAT", "IMPORT.STATUS.MV1","IMPORT.COUNT.MV1");
END SUB
```

```
' Function to run when count variable changes
SUB CountMV1 ()
If (@IMPORT.COUNT.MV1 == -1) Then
@IMPORT.MESSAGE = "Starting import of MV1";
Else
If (@IMPORT.COUNT.MV1 == 0) Then
@IMPORT.MESSAGE = "No import - check error messages";
Else
@IMPORT.MESSAGE = Addstring("Import succesfull - number of records imported = ",
      TOC(@IMPORT.COUNT.MV1));
End If
End If
END SUB
```

Import Example 2.

The FORMAT and VARNAME keywords are contained in the instruction.

```
SUB LoadMV2()
HISTORY("IMPORTTREND","MV2.DAT", "IMPORT.STATUS.MV2","IMPORT.COUNT.MV2",
    "FORMAT,0,0,5", "VARNAME,MV2");
END SUB
```

Example of Import file for LoadMV1:

```
FORMAT,0,0,5
VARNAME,MV1

25/09/2000,00:00:00, 7.64
25/09/2000,00:10:00, 7.87
25/09/2000,00:20:00, 8.12
25/09/2000,00:30:00, 8.40
25/09/2000,00:40:00, 8.69
25/09/2000,00:50:00, 8.99
```

Example of Import file for LoadMV2:

The FORMAT and VARNAME keywords are contained in the instruction.

```
25/09/2000,00:00:00, 7.64
25/09/2000,00:10:00, 7.87
25/09/2000,00:20:00, 8.12
25/09/2000,00:30:00, 8.40
25/09/2000,00:40:00, 8.69
25/09/2000,00:50:00, 8.99
25/09/2000,01:00:00, 9.30
25/09/2000,01:10:00, 9.62
```

LAN (Network Configuration) Example

Applies To

This example exercises several modes of the LAN command.

The network structure is as follows:

- An association of two servers.
- The number and name of the of the association is: 1 and ASSOC01 respectively
- The number of the local server station is: 12
- The number of the remote server station is: 11

```
SUB Main()
langetlists();
END SUB

'Mode CONNECT or Mode 1 (syntax 1)

SUB lanconnect()
DIM iResult as Integer;
iResult = LAN("CONNECT","SERVER11S0");
END SUB

'Mode SET_SERVER_MODE or Mode 5 (syntax 5)

'A function to enable the local server station of the association (ASSOC01)
'to be forced into the active state
SUB Force_Active()
  LAN("SET_SERVER_MODE", "ASSOC01", SYSTEM.STATION_NAME, "SET_SERVER_ACTIF");
END SUB

'A function to enable the local server station of the association (ASSO10)
'to be forced into the passive state
SUB Force_Passive()
  LAN("SET_SERVER_MODE", "ASSOC01", SYSTEM.STATION_NAME, "SET_SERVER_PASSIF");
END SUB
```

Example of modes 13, 14, 15 and 16

```
' StartConnections()
' -----------------
' Function for starting a connection with a remote station (passed in a parameter)
' ARG1 = name of the remote station
'
SUB StartConnections()
  DIM l_Ret AS INTEGER;
  l_Ret = LAN( "START_CONNECTIONS", GETARG("ARG1"));
  PRINT(@TIME, " StartConnections "" ", GETARG("ARG1"), " l_Ret = ", l_Ret);
END SUB
```

```
' StopConnections()
' -----------------
' Function for stopping a connection with a remote station (passed in a parameter)
' ARG1 = name of the remote station
'
SUB StopConnections()
  DIM l_Ret AS INTEGER;
  l_Ret = LAN( "STOP_CONNECTIONS", GETARG("ARG1"));
  PRINT(@TIME, " StopConnections "" ", GETARG("ARG1"), " l_Ret = ", l_Ret);
END SUB
```

```
' StartConnection()
' -----------------
```

```
' Function for starting a connection with a remote station (passed in a parameter)
' ARG1 = name of the remote station
' ARG2 = name of the remote connection
'
SUB StartConnection()
  DIM l_Ret AS INTEGER;
  l_Ret = LAN( "START_CONNECTION", GETARG("ARG1"), GETARG("ARG2"));
  PRINT(@TIME, " StartConnection Remote station ", GETARG("ARG1"), " Remote connection
",GETARG("ARG2"), " l_Ret = ", l_Ret);
END SUB
```

```
' StopConnection()
' -----------------
' Function for stopping a connection with a remote station (passed in a parameter)
' ARG1 = name of the remote station
' ARG2 = name of the remote connection
'
SUB StopConnection()
  DIM l_Ret AS INTEGER;
  l_Ret = LAN( "STOP_CONNECTION", GETARG("ARG1"), GETARG("ARG2"));
  PRINT(@TIME, " StopConnection Remote station ", GETARG("ARG1"), " Remote connection
",GETARG("ARG2"), " l_Ret = ", l_Ret);
END SUB
```

LANGUAGE Example

Applies To

This example exercises several modes of the LANGUAGE command.

You must activate the bilingual mode in the menu Configure.Application Explorer.Doc settings.Languages.Enable bilingual text strings.

```
'Mode GET or Mode 0 (syntax 1)
SUB languageget()
DIM intResult as integer;
intResult = LANGUAGE("Get");
PRINT("The current language is:",intResult);
END SUB

'Mode SET_L1 or Mode 1 (syntax 2)

SUB languagesetL1()
DIM intResult as integer;
intResult = LANGUAGE("SET_L1",1);
END SUB

'Mode SET_L2 or Mode 2 (syntax 2)

SUB languagesetL2()
DIM intResult as integer;
intResult = LANGUAGE("SET_L2",1);
END SUB

'Mode TOGGLE or Mode 3 (syntax 2)

SUB languagetoggle()
DIM intResult as integer;
intResult = LANGUAGE("TOGGLE",1);
END SUB
```

LISTBOX Example

Applies To

This example contains a set of functions to support a pair of List-box form controls using the LISTBOX instruction.

```
SUB Main()
END SUB

'declare variables:
SUB lbxSel()
DIM lRet As Long;
DIM sLbxText As Str;
DIM sLbxText1 As Str;
DIM sLbxData As Str;
DIM sLbxData1 As Str;

'retrieve content of selected item in 1st control:
lRet = LISTBOX ( "GETSELECTEDINDEX", "ctrl","","lbx" );
LISTBOX ( "SETSELECTEDINDEX", "ctrl","","lbx1", lRet);
sLbxText = LISTBOX ( "GETTEXT", "ctrl","","lbx", lRet);
sLbxText1 = LISTBOX ( "GETTEXT", "ctrl","","lbx1", lRet);
sLbxData = LISTBOX ( "GETUSERDATA", "ctrl","","lbx", lRet);
sLbxData1 = LISTBOX ( "GETUSERDATA", "ctrl","","lbx1", lRet);

'put its contents in a frame:
SET ( "LbxText" , sLbxText);
SET("LbxText1", sLbxText1);
SET("LbxData", sLbxData);
SET("LbxData1" , sLbxData1);
SET("LbxCount" , LISTBOX ( "COUNT", "ctrl","","lbx") );
SET("LbxCount1" , LISTBOX ( "COUNT", "ctrl","","lbx1"));
SENDLIST ("BLOC");

'concatenate and load contents into a form control of each kind:
COMBOBOX ( "LOAD", "ctrl","","cbx2", ADDSTRING("cbx",sLbxData));
LISTBOX ( "LOAD", "ctrl","","lbx2", ADDSTRING("lbx",sLbxData));
CHECKLIST ( "LOAD", "ctrl","","chk2", ADDSTRING("chk",sLbxData));
OPTIONLIST ( "LOAD", "ctrl","","opt2", ADDSTRING("opt",sLbxData));
TREEVIEW ( "LOAD", "ctrl","","tvw2", ADDSTRING("tvw",sLbxData));
END SUB

'retrieve content of selected item in 2nd control:
SUB lbxSel1()
DIM lRet As Long;
DIM sLbxText As Str;
DIM sLbxText1 As Str;
DIM sLbxData As Str;
DIM sLbxData1 As Str;
lRet = LISTBOX ( "GETSELECTEDINDEX", "ctrl","","lbx1" );
LISTBOX ( "SETSELECTEDINDEX", "ctrl","","lbx", lRet);
sLbxText = LISTBOX ( "GETTEXT", "ctrl","","lbx", lRet);
sLbxText1 = LISTBOX ( "GETTEXT", "ctrl","","lbx1", lRet);
sLbxData = LISTBOX ( "GETUSERDATA", "ctrl","","lbx", lRet);
sLbxData1 = LISTBOX ( "GETUSERDATA", "ctrl","","lbx1", lRet);
lRet = LISTBOX ( "COUNT", "ctrl","","lbx");
lRet = LISTBOX ( "COUNT", "ctrl","","lbx1");

'put its contents in a frame:
SET ( "LbxText" , sLbxText);
SET("LbxText1", sLbxText1);
SET("LbxData", sLbxData);
SET("LbxData1" , sLbxData1);
SET("LbxCount" , LISTBOX ( "COUNT", "ctrl","","lbx") );
SET("LbxCount1" , LISTBOX ( "COUNT", "ctrl","","lbx1"));
```

```
SENDLIST ("BLOC");
END SUB

'select 2nd item:
SUB setSellbx()
DIM lRet As Long;
LISTBOX ( "SETSELECTEDINDEX", "ctrl","","lbx", 2);
END SUB
```

LOGDISPLAY Example

Applies To

This program exercises several modes of the LOGDISPLAY instruction. It assumes that in the project, there is a mimic called MENU with a Log Viewer animation whose identifier is REPLOG01.

To find the Log Viewer's identifier, select Edit mode, then select the menu Display.Properties List.(Name).

Configuration

```
'  Create a domain DOM01
'  Create a nature NAT01
'Variables used:
'  @PRIORITY.MIN of type REGISTER
'  @PRIORITY.MAX of type REGISTER
'  @VALIDATE of type STATE
'  @CODERETURN of type STATE
'  @RETURN of type TEXT
```

Mode 0 - BEGIN (syntax 1)

```
SUB logdisplaybegin()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("BEGIN","MENU","","REPLOG01");
END SUB
```

Mode 1 - BEFORE (syntax 1)

```
SUB logdisplaybefore()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("BEFORE","MENU","","REPLOG01");
END SUB
```

Mode 3 - AFTER (syntax 1)

```
SUB logdisplayafter()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("AFTER","MENU","","REPLOG01");
END SUB
```

Mode 4 - END(syntax 1)

```
SUB logdisplayend()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("END","MENU","","REPLOG01");
END SUB
```

Mode 9 - DOMAIN (syntax 2)

```
SUB logdisplaydomain()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("DOMAIN","MENU","","REPLOG01","DOM01");
END SUB

SUB logdisplaydomainnofilter()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("DOMAIN","MENU","","REPLOG01","");
END SUB
```

Mode 5 - Nature (syntax 2)

```
SUB logdisplayNature()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("Nature","MENU","","REPLOG01","NAT01");
END SUB

SUB logdisplayNaturenofilter()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("Nature","MENU","","REPLOG01","");
END SUB
```

Mode 12 - SETDATETIME (syntax 3)

```
SUB logdisplaysetdatetime()
'Declare return code
DIM intReturn as integer;
DIM dblStartTime as double;
DIM dblEndTime as double;
dblStartTime = DATETIMEVALUE("21/06/2002", "17:05:12:654");
dblEndTime = DATETIMEVALUE("21/06/2002", "17:10:12:654");
intReturn = LOGDISPLAY("SETDATETIME","MENU","","REPLOG01",dblStartTime,dblEndTime) ;
END SUB
```

Mode 11 - PRINTALL (syntax 1)

```
SUB logdisplayprintall()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("PRINTALL","MENU","","REPLOG01");
END SUB
```

Mode 13 - FILTER (syntax 4)

```
SUB logdisplayfilter()
'Declare return code
DIM intReturn as integer;
DIM strFilter as Str;
strFilter = "=(#t BEG DOM01)";
intReturn = LOGDISPLAY("FILTER","MENU","","REPLOG01",strFilter);
END SUB
```

'Mode 14 - PRINT_SELECTED (syntax 1)

```
SUB logdisplayprintselected()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("PRINT_SELECTED","MENU","","REPLOG01");
END SUB
```

Mode 15 - PRINT_DISPLAY (syntax 1)

```
SUB logdisplayprintdisplay()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("PRINT_DISPLAY","MENU","","REPLOG01");
END SUB
```

Mode 16 - FIRST (syntax 1)

```
SUB logdisplayfirst()
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("FIRST","MENU","","REPLOG01");
END SUB
```

Mode 17 - LAST (syntax 1)

```
SUB logdisplaylast()
```

```
'Declare return code
DIM intReturn as integer;
intReturn = LOGDISPLAY("LAST","MENU","","REPLOG01");
END SUB
```

Mode 23 - LINESELECT (syntax 9)

```
SUB OnLineSel()
LOGDISPLAY("LINESELECT", "log","","log1","main","","linesel" ,""); END SUB
SUB LineSel()
DIM lHandle As Long;
DIM i As Integer;
DIM sPath As Str;
PRINT ("LINESELECT");
lHandle = XMLPATH ( "GET", "/log/log1", "lineselect/variable" );
PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
lHandle = XMLPATH ( "GET", "/log/log1", "lineselect/x" );
PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
lHandle = XMLPATH ( "GET", "/log/log1", "lineselect/y" );
PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
lHandle = XMLPATH ( "GET", "/log/log1", "lineselect/selected" );
PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
PRINT (XMLPATH ( "GETDOUBLE", "/log/log1", "lineselect/time" )); FOR (i=0;i<8;i=i+1)
 sPath = FORMAT ("lineselect.element[%d].value", i+1);
 lHandle = XMLPATH ( "GET", "/log/log1", sPath );
 PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
NEXT
XMLPATH ( "UNLOAD", "/log/log1");
END SUB
```

Mode 25 - GETSORT (syntax 11)

```
SUB GetSort()
DIM lHandle As Long;
LOGDISPLAY ( "GETSORT", "log","","log1");
lHandle = XMLPATH ( "GET", "/log/log1", "getsort/column" );
PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
lHandle = XMLPATH ( "GET", "/log/log1", "getsort/sort" );
PRINT (CGET_BUFFER ( lHandle, 0, 255 ));
XMLPATH ( "UNLOAD", "/log/log1");
END SUB
```

LOGICAL Examples

Applies To

Example 1

This example exercises several modes of the LOGICAL command.

```
'Mode NOT or Mode 0 (syntax 1)
SUB logicalnot()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue As INTEGER;

lngValue = 1; '00.....0001
lngReturn = LOGICAL("NOT",lngValue );
PRINT("NOT(",lngValue,")= ",lngReturn);

'Display NOT(1)= -2  '11.....1110
END SUB

'Mode AND or Mode 1 (syntax 2)

SUB logicaland()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue1 As INTEGER;
DIM lngValue2 As INTEGER;

lngValue1 = 1;  '00.....0001
lngValue2 = 3;  '00.....0011

'AND=  00.....0001
lngReturn = LOGICAL("AND",lngValue1,lngValue2);
PRINT(lngValue1," AND ",lngValue2," = ",lngReturn);

'Display "1 AND 3 = 1"
END SUB

'Mode XOR or Mode 2 (syntax 2)

SUB logicalxor()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue1 As INTEGER;
DIM lngValue2 As INTEGER;

lngValue1 = 1;  '00.....0001
lngValue2 = 3;  '00.....0011

'XOR=  00.....0010
lngReturn = LOGICAL("XOR",lngValue1,lngValue2);
PRINT(lngValue1," XOR ",lngValue2," = ",lngReturn);
'Display "1 XOR 3 = 2"
END SUB

'Mode OR or Mode 3 (syntax 2)

SUB logicalor()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue1 As INTEGER;
DIM lngValue2 As INTEGER;

lngValue1 = 1;  '00.....0001
lngValue2 = 3;  '00.....0011
```

```
'OR=  00.....0011
lngReturn = LOGICAL("OR",lngValue1,lngValue2);
PRINT(lngValue1," OR ",lngValue2," = ",lngReturn);
'Display "1 OR 3 = 3"
END SUB

'Mode SHIFT_LEFT or Mode 4 (syntax 2)
SUB logicalshiftleft()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue1 As INTEGER;
DIM lngValue2 As INTEGER;

lngValue1 = 1;  '00.....0001
lngValue2 = 3;  'shift by 3 digits
 'SHIFT_LEFT = 00.....1000
lngReturn = LOGICAL("SHIFT_LEFT",lngValue1,lngValue2);
PRINT(lngValue1," SHIFT_LEFT ",lngValue2," = ",lngReturn);
'Display "1 SHIFT_LEFT 3 = 8"
END SUB

'Mode SHIFT_RIGHT or Mode 5 (syntax 2)

SUB logicalshiftright()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue1 As INTEGER;
DIM lngValue2 As INTEGER;
lngValue1 = 8;  '00.....1000
lngValue2 = 3;  'shift b 3 digits
'SHIFT_RIGHT = 00.....0001
lngReturn = LOGICAL("SHIFT_RIGHT",lngValue1,lngValue2);
PRINT(lngValue1," SHIFT_RIGHT ",lngValue2," = ",lngReturn);
'Display "8 SHIFT_RIGHT 3 = 1"
END SUB

'Mode MODULO or Mode 6 (syntax 2)

SUB logicalmodulo()
'Declare variables
DIM lngReturn As LONG;
DIM lngValue1 As INTEGER;
DIM lngValue2 As INTEGER;
lngValue1 = 14;
lngValue2 = 12;
'Calculate modulo: 14 = 12 + 2
'decimal 14 equals 2 modulo 12
lngReturn = LOGICAL("MODULO",lngValue1,lngValue2);
PRINT(lngValue1," = ",lngReturn," (mod ",lngValue2,")");
'Display "14 = 2 (mod 12)"

END SUB
```

Example 2

This program shows the use of the LOGICAL instruction.

```
SUB main()
DIM x As LONG;
DIM y As LONG;
x = 622;
y = 217;
PRINT ("x = ", x);
PRINT ("y = ", y);
PRINT (" not(x) =", logical("not", x));
```

```
PRINT (" x and y =", logical("and", x, y));
PRINT (" x xor y =", logical("xor", x, y));
PRINT (" x or y =", logical("or", x, y));
PRINT (" x << 2 =", logical("shift_left", x, 2));
PRINT (" x >> 2 =", logical("shift_right", x, 2));
PRINT (" x modulus 7 =", logical("modulo", x, 7));
END SUB
```

The program displays:

```
x = 622
y = 217
not(x) =-623
x and y =72
x xor y =695
x or y =767
x << 2 =2488
x >> 2 =155
x modulus 7
```

MULTIMEDIA Example

Applies To

This example exercises modes of the MULTIMEDIA command.

It uses:

- a mimic: MENU
- a wave file Alarm.wav in the TP folder
- an avi file demo.avi in the TP folder

It returns "true" or "false" according to the CD drive's capabilities.

```
SUB capabilityeject()
PRINT(MULTIMEDIA("SEND","cdaudio function can eject"));
END SUB

'Begins recording. Print for debugging.
SUB startrec()
PRINT(MULTIMEDIA("SEND","open new type waveaudio alias mysnd"));
PRINT(MULTIMEDIA("SEND","record mysnd"));
END SUB

'To play a wave sound
'The .wav file must be present in the TP folder of this project
SUB playwav()
MULTIMEDIA ("SEND","Play Alarm.wav");
END SUB

'Opening an animation sequence with video
SUB play()
IF (Window("IS_OPEN","MENU","")) THEN
  MULTIMEDIA ("OPENW","movie","demo.avi","MENU", "",70,10,320,240);
ELSE
  MULTIMEDIA ("OPENW","movie","demo.avi","Projection");
END IF
  MULTIMEDIA ("SEND","play movie");
  MULTIMEDIA ("SEND","set movie time format frames");
END SUB
```

OPTIONLIST Example

Applies To

This example contains a set of functions to support a pair of Option-button list form controls using the OPTIONLIST instruction.

```
SUB Main()
END SUB

'declare variables:
SUB optSel()
DIM lRet As Long;
DIM soptText As Str;
DIM soptText1 As Str;
DIM soptData As Str;
DIM soptData1 As Str;
DIM lRet2 As Long;

'retrieve content of selected item in 1st control:
lRet = OPTIONLIST ( "GETSELECTEDINDEX", "ctrl","","opt" );
OPTIONLIST ( "SETSELECTEDINDEX", "ctrl","","opt1", lRet);
soptText = OPTIONLIST ( "GETTEXT", "ctrl","","opt", lRet);
soptText1 = OPTIONLIST ( "GETTEXT", "ctrl","","opt1", lRet);
soptData = OPTIONLIST ( "GETUSERDATA", "ctrl","","opt", lRet);
soptData1 = OPTIONLIST ( "GETUSERDATA", "ctrl","","opt1", lRet);

'put its contents in a frame:
SET ( "optText" , soptText);
SET("optText1", soptText1);
SET("optData", soptData);
SET("optData1" , soptData1);
SET("optCount" , OPTIONLIST ( "COUNT", "ctrl","","opt") );
SET("optCount1" , OPTIONLIST ( "COUNT", "ctrl","","opt1"));
SENDLIST ("BLOC");

'check the latest change
' 1 Selection has changed; 2 Option has changed
PRINT ( OPTIONLIST ( "GETEVENTTYPE", "ctrl","","opt") );
If ( OPTIONLIST ( "GETEVENTTYPE", "ctrl","","opt") == 2 )
Then
      lRet2 = OPTIONLIST ( "GETSTATE", "ctrl","","opt", lRet);
      OPTIONLIST ( "SETSTATE", "ctrl","","opt1", lRet, lRet2);
End If

'concatenate and load contents into a form control of each kind:
COMBOBOX ( "LOAD", "ctrl","","cbx2", ADDSTRING("cbx",soptData));
LISTBOX ( "LOAD", "ctrl","","lbx2", ADDSTRING("lbx",soptData));
CHECKLIST ( "LOAD", "ctrl","","chk2", ADDSTRING("chk",soptData));
OPTIONLIST ( "LOAD", "ctrl","","opt2", ADDSTRING("opt",soptData));
TREEVIEW ( "LOAD", "ctrl","","tvw2", ADDSTRING("tvw",soptData));
END SUB

'retrieve content of selected item in 2nd control:
SUB optSel1()
DIM lRet As Long;
DIM soptText As Str;
DIM soptText1 As Str;
DIM soptData As Str;
DIM soptData1 As Str;
lRet = OPTIONLIST ( "GETSELECTEDINDEX", "ctrl","","opt1" );

OPTIONLIST ( "SETSELECTEDINDEX", "ctrl","","opt", lRet);
soptText = OPTIONLIST ( "GETTEXT", "ctrl","","opt", lRet);
soptText1 = OPTIONLIST ( "GETTEXT", "ctrl","","opt1", lRet);
soptData = OPTIONLIST ( "GETUSERDATA", "ctrl","","opt", lRet);
```

```
soptData1 = OPTIONLIST ( "GETUSERDATA", "ctrl","","opt1", lRet);
lRet = OPTIONLIST ( "COUNT", "ctrl","","opt");
lRet = OPTIONLIST ( "COUNT", "ctrl","","opt1");

'put its contents in a frame:
SET ( "optText" , soptText);
SET("optText1", soptText1);
SET("optData", soptData);
SET("optData1" , soptData1);
SET("optCount" , OPTIONLIST ( "COUNT", "ctrl","","opt") );
SET("optCount1" , OPTIONLIST ( "COUNT", "ctrl","","opt1"));
SENDLIST ("BLOC");
END SUB

'select 2nd item:
SUB setSelopt()
DIM lRet As Long;
OPTIONLIST ( "SETSELECTEDINDEX", "ctrl","","opt", 2);
END SUB
```

POPULATION Example

Applies To

Create a population filter and display its definition.

Example 1 using Seq_Buffer

```
SUB EX1 ()
  DIM HB As Long;

  '--------Create buffer
  HB = Alloc_Buffer(4096);

  '-------Define population in buffer using Seq_Buffer
  Seq_Buffer("PUT_LINE",HB,"POPULATION,23,POP1,,,0,1,\"\"");
  Seq_Buffer("PUT_LINE",HB,"POPDEF,POP1,TATT1,IN,ZONE1");
  Seq_Buffer("PUT_LINE",HB,"POPDEF,POP1,TATT1,IN,ZONE2");
  Seq_Buffer("PUT_LINE",HB,"POPDEF,POP1,TYPE,OUT,4");

  '--------Create population
  Population("CREATE", HB);
  Population("DUMP", "POP1"); ' Dump in program debug area to verify
  Free_Buffer(HB);
END SUB
```

Example 2 using Population mode ADDLINE

```
SUB EX2 ()
  DIM HB As Long;

  '--------Create buffer
  HB = Alloc_Buffer(4096);

  '--------Define population in buffer using ADDLINE
  Population("ADDLINE",HB,"POPULATION,23,POP2,,,0,1,\"\"");
  Population("ADDLINE",HB,"POPDEF,POP2,TATT1,IN,ZONE1");
  Population("ADDLINE",HB,"POPDEF,POP2,TYPE,OUT,4");

  '--------Create population
  Population("CREATE", HB);
  Population("DUMP", "POP2"); ' Dump in program debug area to verify
  Free_Buffer(HB);
END SUB
```

PRINTER Examples

Applies To

This program exercises several modes of the PRINTER command.

```
const printer_no_primary = 1;
const printer_no_secondary = 2;

'Mode OFF or Mode 0 (syntax 1)

SUB printeroff()
DIM intResult as integer;
intResult = PRINTER("OFF",printer_no_primary);
END SUB

'Mode ON or Mode 1 (syntax 1)

SUB printeron()
DIM intResult as integer;
intResult = PRINTER("ON",printer_no_primary);
END SUB

'Mode CLEAR or Mode 2 (syntax 1)

SUB printerclear()
DIM intResult as integer;
intResult = PRINTER("CLEAR",printer_no_primary);
END SUB

'Mode GETNB or Mode 3 (syntax 1)

SUB printergetnb()
DIM intResult as integer;
intResult = PRINTER("GETNB",printer_no_primary);
PRINT("Result of GETNB is: ",intResult);
END SUB

'Mode STATUS or Mode 4 (syntax 1)

SUB printerstatus()
DIM intResult as integer;
intResult = PRINTER("STATUS",printer_no_primary);
PRINT("Result of STATUS is: ",intResult);
END SUB

'Mode SELECT or Mode 5 (syntax 2)

SUB printerselect()
DIM intResult as integer;
intResult = PRINTER("SELECT",printer_no_primary,printer_no_secondary);
END SUB

'Mode FLUSH or Mode 6 (syntax 1)

SUB printerflush()
DIM intResult as integer;
intResult = PRINTER("FLUSH",printer_no_primary);
END SUB
```

PROGRAM Example

Applies To

This program exercises several modes of the PROGRAM command.

```
'This program uses another program called BEEP.SCB
'that has a function main() and a fonction beepfunction1()
'Mode STOP or Mode 0 (syntax 1)

SUB programstop()
DIM intResult as integer;
intResult = PROGRAM("STOP","beep.SCB","");
END SUB

'Mode START or Mode 1 (syntax 1)

SUB programstart()
DIM intResult as integer;
intResult = PROGRAM("START","beep.SCB","");
END SUB

'Mode IS_LOADED or Mode 2 (syntax 1)

SUB programisloaded()
DIM intResult as integer;
intResult = PROGRAM("IS_LOADED","beep.SCB","");
IF(intResult) Then
  PRINT("The program is loaded");
END IF
END SUB

'Mode PRELOAD or Mode 3 (syntax 1)

SUB programpreload()
DIM intResult as integer;
intResult = PROGRAM("PRELOAD","beep.SCB","");
END SUB

'Mode UNLOAD or Mode 4 (syntax 1)

SUB programunload()
DIM intResult as integer;
intResult = PROGRAM("UNLOAD","beep.SCB","");
END SUB

'Mode EXECUTE or Mode 5 (syntax 1)

SUB programexecute()
DIM intResult as integer;
intResult = PROGRAM("EXECUTE","beep.SCB","");
END SUB

'Mode FUNCTION or Mode 6 (syntax 2)

SUB programfunction()
DIM intResult as integer;
intResult = PROGRAM("FUNCTION","beep.SCB","","beepfunction1");
END SUB

'Mode IS_FUNCTION or Mode 7 (syntax 2)

SUB programisfunction()
DIM intResult as integer;
intResult = PROGRAM("IS_FUNCTION","beep.SCB","","beepfunction1");
IF(intResult) Then
```

```
  PRINT("The function exists");
END IF
END SUB
```

RECIPE Examples

Applies To

Example 1

This module uses the READ and CREATE modes of the RECIPE instruction.

```
SUB CopyRecipe (OldID, NewID)
  DIM Handle As Long;
  DIM OK As Integer;
  DIM Title As STR;
  Const BufSize 1024;
  Handle = Alloc_Buffer (BufSize);
  Title = AddString("New Recipe ", System("DATE"));
  If (Recipe("READ", OldID, Handle, BufSize-1) ==1) THEN
    Recipe("CREATE", Handle, NewID, Title, 1,"Auto");
    OK = 1;
  Else
    OK = 0;
  End If
  Free Buffer(Handle);
  Return(OK);
END SUB
```

Example 2

This example sends a recipe and triggers an event. By detecting that event, it retrieves the contents.

```
DIM handle As Long ;
SUB SendRec()
  handle = ALLOC_BUFFER ( 1000);
  SET ( "EVTVAR2 " , 0 );
  SENDLIST ( 0 );
  EVENT ("ADDPROG", "EVTVAR2", 1, "PGMRECIPE", "", "onEvt2");
  RECIPE ( "NETWORKBROADCAST" , "REC1" , "LIST2" , handle , "EVTVAR2", 1);
END SUB
SUB onEvt2 ()
  DIM ret As Str;
  EVENT ("DEL", "", 1, "EVTVAR2", "", "onEvt2");
  ret = CGET_BUFFER ( handle , 0 ,255);
  print ( ret );
  FREE_BUFFER ( handle );
END SUB
```

REGION Example

Applies To

This example configures two regions so that subsequent windows opened by animation or SCADA BASIC program will appear in region 2:

```
SUB Main()
'Declare variables
DIM Result As INTEGER;
DIM NoOfRegions  As INTEGER;
DIM WhichRegion  As INTEGER;
DIM SubMode As INTEGER;

'set the number of regions
Result = REGION("SETSYSREGION", 2);
NoOfRegions  = REGION("GETSYSREGION");
PRINT ("Number of regions = ", NoOfRegions);

'set the submode for opening windows from animations
SubMode = REGION("GETSELECTIONMODE");
PRINT ("Current submode = ", SubMode);

'set the region for opening windows from animations
Result = REGION("SETSELECTION", 2, 1);
WhichRegion  = REGION("GETSELECTION");
PRINT ("Open by animation in region = ", WhichRegion);

'set the region for opening windows from SCADA BASIC
Result = REGION("SETREGION", 2);
WhichRegion  = REGION("GETREGION");
PRINT ("Open from SCADA BASIC in region = ", WhichRegion);
END SUB
```

If no errors occur, the results are:

```
Number of regions = 2
Current submode = 1
Open by animation in region = 2
Open from SCADA BASIC in region = 2
```

RETURN Example

Applies To

In this example, the Main module passes values to the Factorial module and prints the returned values.

```
SUB Main ()
DIM J As Integer;
For (J=1; J<=10; J++)
  Print("Factorial ", J, " = ", Factorial(J));
Next J;
END SUB

SUB Factorial (Input)
DIM I As Integer;
DIM Result As Long;
Result = 1;
If (Input < 1) Then
  Return (0);
End If
For ( I = 1; I <=Input; I++)
  Result = Result*TOL(I);
Next I;
Return(Result);
END SUB
```

SELECTOR Example

Applies To

Example 1

This example exercises several modes of the SELECTOR command,

It uses variables that start with "WORKS", whose 2nd branch is "BLOC" and that end with the code: WORKS.BLOC.*[0-9]$.

It requires a mimic "cell" to be loaded, with a grid control whose index is "SELECTOR 1".

```
SUB filling()
DIM lngbuffer12 as long;
lngbuffer12 = ALLOC_BUFFER (4096);
'The file "SELECTOR .txt" holds a description of fields with
'  separators as shown here: ; for columns and \n for lines.
lngbuffer12 = FILETOBUF ("SELECTOR .txt");
SELECTOR ("PUTARRAY","cell","","SELECTOR 1",lngbuffer12,5,0,10,3,";\n");
free_buffer(lngbuffer12);
END SUB

'Mode GETNBLINE or Mode 1 (syntax 1)

SUB SELECTOR getnbline()
'Retrieve the number of lines in view
PRINT ("Number of lines = ",SELECTOR ("GETNBLINE","cell","","SELECTOR 1"));
END SUB

'Mode GETNBCOL or Mode 2 (syntax 1)

SUB SELECTOR getnbcol()
'Retrieve the number of columns of the animation
PRINT ("Number of columns = ",SELECTOR ("GETNBCOL","cell","","SELECTOR 1"));
END SUB

'Mode GETCELL or Mode 3 (syntax 2)

SUB SELECTOR getcell()
'Retrieve the contents of cell x=3 y=1 (base 0)
PRINT ("CELL(3,1) = ",SELECTOR ("GETCELL","cell","","SELECTOR 1",3,1));
END SUB

'Mode GETLINE or Mode 4 (syntax 3)

SUB SELECTOR getline()
'Declare variables
DIM lngBuffer1 as long;
DIM intResult as integer;
lngBuffer1 = ALLOC_BUFFER (4096);

'Retrieve in buffer hdl line 3 - fields separated by: ;
intResult = SELECTOR ("GETLINE","cell","","SELECTOR 1",3,lngBuffer1,";");
PRINT("GETLINE 3     ",CGET_BUFFER(lngBuffer1,0,200));
free_buffer(lngBuffer1);
END SUB

'Mode GETCOL or Mode 5 (syntax 4)

SUB SELECTOR getcol()
'Declare variables
DIM lngBuffer1 as long;
DIM intResult as integer;
lngBuffer1 = ALLOC_BUFFER (4096);

'Retrieve column 1 (base 0) on retrieve the fields
```

```
'separated by: ; in buffer of hdl
intResult = SELECTOR ("GETCOL","cell","","SELECTOR 1",1,lngBuffer1,";");
PRINT("GETCOL 1    ",CGET_BUFFER(lngBuffer1,0,200));
free_buffer(lngBuffer1);
END SUB

'Mode GETARRAY or Mode 6 (syntax 5)

SUB SELECTOR getarray()
'Declare variables
DIM lngBuffer1 as long;
DIM intResult as integer;
lngBuffer1 = ALLOC_BUFFER (4096);
'Retrieve the subset of grid contents between cell (2,0) and cell (4,2)
' - retrieval of fields separated by ; and : in buffer hdl
PRINT (">",SELECTOR ("GETARRAY","cell","","SELECTOR 1",lngBuffer1,2,0,4,2,";:"),"<");
PRINT ("GETARRAY 2,0-4,2    ",CGET_BUFFER(lngBuffer1,0,200));
free_buffer(lngBuffer1);
END SUB

'Mode PUTCELL or Mode 7 (syntax 6)

SUB SELECTOR putcell()
'Declare variables
DIM intResult as integer;
'Writing to a cell
PRINT ("PUTCELL(3,2): ",SELECTOR ("PUTCELL","cell","","SELECTOR 1",0,0,",t, test"));
END SUB

'Mode PUTLINE or Mode 8 (syntax 3)

SUB SELECTOR putline()
'Declare variables
DIM lngBuffer1 as long;
DIM intResult as integer;
lngBuffer1 = ALLOC_BUFFER (4096);
'Retrieve in hdl line 3, fields separated by ;
intResult = SELECTOR ("GETLINE","cell","","SELECTOR 1",3,lngBuffer1,";");
'Filling line 5 with contentsf hdl
' ie the previous line 3 in this example
PRINT (SELECTOR ("PUTLINE","cell","","SELECTOR 1",5,lngbuffer1,";"));
PRINT("PUTLINE 3 -> 5 ",CGET_BUFFER(lngBuffer1,0,200));
free_buffer(lngBuffer1);
END SUB

'Mode PUTCOL or Mode 9 (syntax 4)

SUB SELECTOR putcol()
'Declare variables
DIM lngBuffer1 as long;
DIM intResult as integer;
lngBuffer1 = ALLOC_BUFFER (4096);
'Retrieve column 1 (base 0)
' - retrieval of fields separated by ; and : in buffer hdl
intResult = SELECTOR ("GETCOL","cell","","SELECTOR 1",1,lngBuffer1,";");
'filling of column 3 with contents of hdl
'ie column 1
PRINT (SELECTOR ("PUTCOL","cell","","SELECTOR 1",3,lngbuffer1,";"));
PRINT("PUTCOL 1 -> 3 ",CGET_BUFFER(lngBuffer1,0,200));
free_buffer(lngBuffer1);
END SUB

'Mode PUTARRAY or Mode 10 (syntax 5)

SUB SELECTOR putarray()
```

```
'Declare variables
DIM lngBuffer1 as long;
DIM intResult as integer;
lngBuffer1 = ALLOC_BUFFER (4096);
'Retrieve the subset of grid contents between (2,0) and cell (4,2)
' - retrieve fields separated by ; and : in buffer hdl
PRINT (">",SELECTOR ("GETARRAY","cell","","SELECTOR 1",lngBuffer1,0,0,2,2,";\n"),"<");
'filling column 3 with contents of hdl
'ie column 1
intResult = SELECTOR ("PUTARRAY","cell","","SELECTOR 1",lngBuffer1,2,0,4,2,";\n");
PRINT("PUTARRAY 2,0-4,2 ",CGET_BUFFER(lngBuffer1,0,200));
free_buffer(lngBuffer1);
END SUB

'Mode SCROLL or Mode 11 (syntax 7)

'example of control of scrolling of the animation by forcing by program
SUB tobegin ()
SELECTOR ("SCROLL","cell","","SELECTOR 1","TOBEGIN");
updatepcsel ();
END SUB

SUB pageup ()
SELECTOR ("SCROLL","cell","","SELECTOR 1","PAGEUP");
updatepcsel ();
END SUB

SUB lineup ()
SELECTOR ("SCROLL","cell","","SELECTOR 1","LINEUP");
updatepcsel ();
END SUB

SUB linedown ()
SELECTOR ("SCROLL","cell","","SELECTOR 1","LINEDOWN");
updatepcsel ();
END SUB

SUB pagedown ()
SELECTOR ("SCROLL","cell","","SELECTOR 1","PAGEDOWN");
updatepcsel ();
END SUB

SUB toend ()
SELECTOR ("SCROLL","cell","","SELECTOR 1","TOEND");
updatepcsel ();
END SUB

'to go to line line ll directly
SUB toline (ll)
SELECTOR ("SCROLL","cell","","SELECTOR 1","TOLINE",ll);
updatepcsel ();
END SUB

'Mode GETLISTART or Mode 12 (syntax 1)

SUB SELECTOR getlistart()
'Retrieve the 1st visible line
PRINT ("The forst line displayed is = ",SELECTOR ("GETLISTART","cell","","SELECTOR 1"));
END SUB

'Mode GETNBLINEMAX or Mode 13 (syntax 1)

SUB SELECTOR getnblinemax()
PRINT ("The total number of lines shown in the memory grid is = ",SELECTOR
("GETNBLINEMAX","cell","","SELECTOR 1"));
```

```
END SUB

'Mode VARIABLE or Mode 14 (syntax 9)

SUB SELECTOR variable()
DIM intResult as integer;
'Showing all variables of type TEXT
intResult = SELECTOR ("VARIABLE","cell","","SELECTOR 1",".*",8);
END SUB

'Mode VARMODE or Mode 15 (syntax 10)

SUB SELECTOR varmode()
DIM intResult as integer;
'Showing all variables of type TEXT
intResult = SELECTOR ("VARMODE","cell","","SELECTOR 1",2);
END SUB

'Mode SELECTLINE or Mode 16 (syntax 1)

SUB SELECTOR selectline()
'Retrieve the 1st visible line
PRINT ("The selected line is : ",SELECTOR ("SELECTLINE","cell","","SELECTOR 1"));
END SUB

'Mode SELECTCOL or Mode 17 (syntax 1)

SUB SELECTOR selectcol()
'Retrieve the 1st visible line
PRINT ("The selected column is : ",SELECTOR ("SELECTCOL","cell","","SELECTOR 1"));
END SUB

'Mode SELECTMODE or Mode 18 (syntax 7)

SUB modeoff() ' mode without selection
SELECTOR ("SELECTMODE","cell","","SELECTOR 1","OFF");
END SUB

SUB modemono() ' single selection mode
SELECTOR ("SELECTMODE","cell","","SELECTOR 1","MONO");
END SUB

SUB modemulti() ' multi-selection mode
SELECTOR ("SELECTMODE","cell","","SELECTOR 1","MULTI");
END SUB

SUB modeinput() ' direct selection mode
SELECTOR ("SELECTMODE","cell","","SELECTOR 1","INPUT");
END SUB

'Mode SFIRSTCELL or Mode 19 (syntax 1)

SUB SELECTOR sfirstcell()
'Find the first cell selected from the start of the grid (0,0).
PRINT ("The first cell selected is : ",SELECTOR ("SFIRSTCELL","cell","","SELECTOR 1"));
END SUB

'Mode SNEXTCELL or Mode 20 (syntax 1)

SUB SELECTOR snextcell()
'Find the next cell selected from the last cell found
PRINT ("The next cell selected is : ",SELECTOR ("SNEXTCELL","cell","","SELECTOR 1"));
END SUB

'Mode ALLSELOFF or Mode 21 (syntax 1)
```

```
SUB SELECTOR allseloff()
SELECTOR ("ALLSELOFF","cell","","SELECTOR 1");
END SUB

'Mode CLICKCELL or Mode 22 (syntax 2)

SUB SELECTOR clickcell()
'Retrieve of contents of cell x=3 y=1 (base 0)
PRINT ("Select cell (3,1) = ",SELECTOR ("CLICKCELL","cell","","SELECTOR 1",3,1));
END SUB

'Mode GETSEL or Mode 23 (syntax 2)

SUB SELECTOR getsel()
'Enables you to know whether cell x=3 y=1 (base 0)is selected
PRINT ("Selection of cell (3,1) = ",SELECTOR ("GETSEL","cell","","SELECTOR 1",3,1));
END SUB

'Mode SELPROG or Mode 24 (syntax 11)

SUB SELECTOR selprog()
'The main function of program beep.SCB is executed on selection
SELECTOR ("SELPROG","cell","","SELECTOR 1","beep.SCB","","");
END SUB

'Mode INPUTPROG or Mode 25 (syntax 11)
SUB SELECTOR inputprog()
'The main function of program beep.SCB is executed on
' confirmation of dialog box
SELECTOR ("INPUTPROG","cell","","SELECTOR 1","beep.SCB","","");
END SUB

'Mode CLEAR or Mode 26 (syntax 1)

SUB SELECTOR clear()
'Enables purging of the grid. Cells are re-initialised.
PRINT ("Purge grid ",SELECTOR ("CLEAR","cell","","SELECTOR 1"));
END SUB

'Mode SETNBLINE or Mode 27 (syntax 7)

SUB SELECTOR setnbline()
'Moves on the grid to line 10
SELECTOR ("SETNBLINE","cell","","SELECTOR 1",10);
END SUB

'Mode GETLASTCELL or Mode 28 (syntax 1)

SUB SELECTOR getlastcell()
'Returns the number of lines used in the grid
PRINT ("Number of lines used ",SELECTOR ("GETLASTCELL","cell","","SELECTOR 1"));
END SUB

'Mode SORT or Mode 27 (syntax 7)

SUB SELECTOR sort()
'Sorting column 0 of the grid into ascending order
SELECTOR ("SORT","cell","","SELECTOR 1",0,0);
END SUB
```

Example 2

The following example displays information from a file containing data recorded in one of the disk units for a Trend Viewer. The format of the file is as follows.

```
940211142058, 244
940211142112, 237
940211142113, 230
940211142127, 223
940211142127, 216
940211142141, 209
940211142141, 202
940211142155, 195
```

The program displays data in a mimic called "cell" that contains a grid identified as "SELECTOR 1".

```
' Program SEL.PVB
' Date 21-Mar-93
' Author RJM

SUB Main ()
END SUB

' Load data into Grid. Note the use of \n as a line delimiter.
SUB LoadGrid ()
DIM hdl2 As Long;
hdl2 = Filetobuf ("trend.dat");
SELECTOR ("PUTARRAY", "cell", "", "SELECTOR 1",hdl2,0,0, 35, 1, ", \n");
Free_Buffer(hdl2);
END SUB

' Provides scroll facility.
SUB PageUp ()
SELECTOR ("SCROLL", "cell", "", "SELECTOR 1", "PAGEUP");
UPDATE ();
END SUB

SUB PageDown ()
SELECTOR ("SCROLL", "cell", "", "SELECTOR 1","PAGEDOWN");
UPDATE ();
END SUB

SUB LineUp ()
SELECTOR ("SCROLL", "cell", "","SELECTOR 1","LINEUP");
UPDATE ();
END SUB

SUB LineDown ()
SELECTOR ("SCROLL", "cell", "", "SELECTOR 1","LINEDOWN");
UPDATE ();
END SUB

SUB ToStart ()
SELECTOR ("SCROLL", "cell", "", "SELECTOR 1","TOBEGIN");
UPDATE ();
END SUB

SUB ToEnd ()
SELECTOR ("SCROLL", "cell", "", "SELECTOR 1","TOEND");
UPDATE ();
END SUB

' Function to update a variable indicating the current position
' of the file within the Grid.
SUB Update ()
DIM LStart As Integer;
DIM NblMax As Integer;
DIM Nbl As Integer;
LStart = SELECTOR ("GETLISTART", "cell", "", "SELECTOR 1");
Nblmax = SELECTOR ("GETNBLINEMAX", "cell", "", "SELECTOR 1");
Nbl = SELECTOR ("GETNBLINE", "cell", "", "SELECTOR 1");
```

```
pcsel = TOS((LStart*100)/(Nblmax - nbl));
END SUB

SUB deselect()
SELECTOR ("ALLSELOFF", "cell", "", "SELECTOR 1");
END SUB

SUB modeoff() 'mode without selection
SELECTOR ("SELECTMODE", "cell", "", "SELECTOR 1","OFF");
END SUB

SUB modemono() 'mode mono selection
SELECTOR ("SELECTMODE", "cell","", "SELECTOR 1", "MONO");
END SUB

SUB modemulti() 'mode multi selection
SELECTOR ("SELECTMODE", "cell", "","SELECTOR 1","MULTI");
END SUB

SUB modeinput() 'direct entry mode
SELECTOR ("SELECTMODE", "cell", "", "SELECTOR 1", "INPUT");
END SUB
```

Example 3

```
‘Set the text and background color of a set of cells (from cell 0,0 to 2,2)
SUB PUTCOLOR()
  dim col as Long;
  dim red as Long;
  dim green as Long;
  dim blue as Long;
' Set the background color. Calculate the color using the values of the RGB components of
a color
' (each from 0 to 255)
  red = TOL(@VARCOL.RED);
  green = LOGICAL("SHIFT_LEFT", TOL(@VARCOL.GREEN), 8);
  blue = LOGICAL("SHIFT_LEFT", TOL(@VARCOL.BLUE), 16);
  col = red + green + blue;
  SELECTOR("PUTBACKCOLOR", "GRIDS", "", "GRIDNEW", 0, 0, 2,2, col);
' Set the text color (opposite color of the background).
' Calculate the color using the values of the RGB components of the color (each from 0 to
255)
  red = 255 - TOL(@VARCOL.RED);
  green = LOGICAL("SHIFT_LEFT", 255 - TOL(@VARCOL.GREEN), 8);
  blue = LOGICAL("SHIFT_LEFT", 255 - TOL(@VARCOL.BLUE), 16);
  col = red + green + blue;
  SELECTOR("PUTTEXTCOLOR", "GRIDS", "", "GRIDNEW", 0, 0, 2,2, col);
END SUB

' Read the background color of a cell (0, 0)
SUB GETCOLOR()
  dim col as long;
  col = SELECTOR("GETCELLBACKCOLOR", "GRIDS", "", "GRIDNEW", 0, 0);
' Retrieve the RGB components of the color
  @VARCOL.RED = TOS(LOGICAL("AND", col, 255));
  @VARCOL.GREEN = TOS(LOGICAL("AND", LOGICAL("SHIFT_RIGHT", col, 8), 255));
  @VARCOL.BLUE = TOS(LOGICAL("AND", LOGICAL("SHIFT_RIGHT", col, 16), 255));
END SUB

' Retrieve the variable name of the row currently selected (GRID in "Variable tracking"
mode).
SUB GETVARIABLENAME()
  @TEXT1= SELECTOR("GETVARNAME", "GRIDS", "", "GRIDNEW", SELECTOR("SELECTLINE", "GRIDS",
"", "GRIDNEW"));
END SUB
```

SEQ_BUFFER Example

Applies To

Example 1

This example exercises several modes of the SEQ_BUFFER command.

```
'Mode CLEAR or Mode 1 (syntax 1)

SUB seqbufferclear()
'Declare variables
DIM lngbuffer1 as long;
DIM strline as Str;
DIM strResult as Str;
DIM intResult as integer;

lngbuffer1 = ALLOC_BUFFER(110);
strline = "123.34;a string,here;345;123456789;,t,\n";
PUT_BUFFER(lngbuffer1, 0, strline);
'Retrieve the buffer contents to display
strResult = CGET BUFFER(lngbuffer1,0,30,0);
PRINT("The retrieved string is: ",strResult);
'All entries in the buffer are not shown
intResult = SEQ_BUFFER("CLEAR",lngbuffer1) ;
'Retrieve the buffer contents to display
strResult = CGET_BUFFER(lngbuffer1,0,30,0);
PRINT("The retrieved string is: ",strResult);
FREE_BUFFER(lngbuffer1);
END SUB

'Mode PUT_LINE or Mode 2 (syntax 2)

SUB seqbufferputline()
'Declare variables
DIM lngbuffer1 as long;
DIM strline as Str;
DIM strResult as Str;
DIM intResult as integer;
lngbuffer1 = ALLOC_BUFFER(110);
strline = "123.34;a string here;345;123456789;,t,\n";
intResult = SEQ_BUFFER("PUT_LINE",lngbuffer1,strline) ;
'Retrieve the buffer contents to display
strResult = CGET BUFFER(lngbuffer1,0,30,0);
PRINT("The retrieved string is: ",strResult);
FREE_BUFFER(lngbuffer1);
END SUB

'Mode BEGIN or Mode 3 (syntax 3)

SUB seqbufferbegin()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve of the buffer in a type Str if necessary
```

```
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB

'Mode END or Mode 4 (syntax 3)

SUB seqbufferend()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("END",lngbuffer1);
SEQ_BUFFER ( "SEEKFIELD" ,-1, lngbuffer1 , "\n");
SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
PRINT(strResult);
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB

'Mode SEEKFIELD or Mode 7 (syntax 5)

SUB seqbufferfields()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve of buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
  SEQ_BUFFER ( "PREVFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB

'Mode INSERTFIELD or Mode 8 (syntax 6)

SUB seqbufferinsertfield()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM lngbuffer3 as long;
DIM strResult as Str;
```

```
DIM strline as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer3 = ALLOC_BUFFER(110);
lngbuffer3 = ALLOC_BUFFER(110);
strline = "123.34;a string, here;345;123456789;,t,\n";
PUT_BUFFER(lngbuffer3, 0, strline);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ( "INSERTFIELD" ,2, lngbuffer1 , "\n", lngbuffer3) ;
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
FREE_BUFFER(lngbuffer3);
END SUB

'Mode CRLFTOCR or Mode 9 (syntax 3)

SUB seqbuffercrltocr()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
intResult = SEQ_BUFFER ("CRLFTOCR",lngbuffer1);
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB

'Mode CRTOCRLF or Mode 10 (syntax 3)

SUB seqbuffercrtocrlf()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
intResult = SEQ_BUFFER ("CRTOCRLF",lngbuffer1);
```

```
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB

'Mode LEN or Mode 11 (syntax 3)

SUB seqbufferlen()
'Declare variables
DIM lngbuffer1 as long;
DIM strResult as Str;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("LEN",lngbuffer1);
PRINT(intResult);
FREE BUFFER(lngbuffer1);
END SUB

'Mode ASCIITOANSI or Mode 12 (syntax 3)

SUB seqbufferasciiansii()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
intResult = SEQ_BUFFER (12,lngbuffer1);
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve of buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB

'Mode ANSITOASCII or Mode 13 (syntax 3)

SUB seqbufferansiiascii()
'Declare variables
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM strResult as Str;
DIM intResult as Integer;
DIM i as integer;
lngbuffer1 = ALLOC_BUFFER(110);
lngbuffer2 = ALLOC_BUFFER(110);
lngbuffer1 = FILETOBUF("tool.txt");
intResult = SEQ_BUFFER ("BEGIN",lngbuffer1);
```

```
intResult = SEQ_BUFFER (13,lngbuffer1);
i = 1;
While ( i != 0)
  'Read lines
  i = SEQ_BUFFER ( "NEXTFIELD" , lngbuffer1 , "\n", lngbuffer2);
  'Retrieve buffer as type Str if necessary
  strResult = CGET_BUFFER ( lngbuffer2  , 0 , 255 );
  PRINT(strResult);
Wend
FREE_BUFFER(lngbuffer1);
FREE_BUFFER(lngbuffer2);
END SUB
```

Example 2

Here are examples of the SEQ_BUFFER command's modes REPLACEFIELD and INSERTFIELD:

```
SEQ_BUFFER("Clear",hbuf1); 'clear the buffer
Put_Buffer(hbuf1, 0, "123,xyz"); 'insert a string of two fields
PRINT(CGET_BUFFER(hbuf1,0,40,0)); 'shows: 123,xyz
Seq_Buffer("Begin",hbuf1); 'locate pointer at start of buffer
res=SEQ_BUFFER("REPLACEFIELD", 2, hbuf1, ",", "456"); 'insert 456 as field 2
PRINT(CGET_BUFFER(hbuf1,0,40,0)); 'shows: 123,456
res=SEQ_BUFFER("REPLACEFIELD", 2, hbuf1, ",", "456,789"); 'retain: 456, insert: 789
PRINT(CGET_BUFFER(hbuf1,0,40,0)); 'shows: 123,456,789
```

REPLACEFIELD operation:

To replace the specified field "oField_i" with your new field "nField_i", you can use:

```
SEQ_BUFFER("REPLACEFIELD", i, Handle, ",", "nField_i");
```

INSERTFIELD operation:

To insert your new field "nField_i" before the specified field "oField_i", you must use:

```
SEQ_BUFFER("INSERTFIELD", i, Handle, ",", "nField_i, oField_i");
```

To write texts to the buffer:

```
SUB Main()
DIM hb As Long; Handle for the buffer
Alloc_Buffer(4096);
Seq_Buffer("PUT_LINE",hb,"POPULATION,23,ID1,0,0,Free_comment");
Seq_Buffer("PUT_LINE",hb,"POPDEF,ID1,TATT1,IN, DOM1");
Seq_Buffer("PUT_LINE",hb,"POPDEF,ID1,TATT1,IN, DOM2");
Seq_Buffer("PUT_LINE",hb,"POPDEF,ID1,TYPE,OUT,4" );
Population("CREATE",hb );
Population("DUMP","ID1") ;'For verification
Station_Filter("APPLY",2,"ID1");
Free_buffer(hb) ;
END SUB
```

This program loops through a buffer, reading a line then displaying the values in it.

```
DIM Hdl As Long; ' Contains the string to process
DIM Hdl2 As Long; ' Contains the current line
DIM Hdl3 As Long; ' Contains the current value
```

```
SUB Main ()
hdl = 0;
hdl2 = 0;
hdl3 = 0;
END SUB

SUB ReadFields()
DIM i As Integer;
DIM j As Integer;
DIM p As Str;
DIM sep As Str;
DIM sep2 As Str;
```

```
If ( hdl == 0 ) Then
 hdl = ALLOC BUFFER ( 16000 );
End If
sep = "\n";
sep2 = ",";
If ( hdl2 == 0 ) Then
 hdl2 = ALLOC_BUFFER ( 1000 );
End If
If ( hdl3 == 0 ) Then
 hdl3 = ALLOC_BUFFER ( 1000 );
End If
SEQ_BUFFER ( "BEGIN" , hdl ); ' Start position
i = 1;
While ( i != 0) ' Read the lines
 i = SEQ_BUFFER ( "NEXTFIELD" , hdl , sep , hdl2, 1000);
 p = CGET_BUFFER ( hdl2 , 0 , 255 );
 If ( i == 0 ) Then
   Break;
 End If
 j = SEQ_BUFFER ( "BEGIN" , hdl2 );
 j = 1;
 While ( j != 0)
   j = SEQ_BUFFER ( "NEXTFIELD" , hdl2 , sep2 , hdl3 , 1000 );
   p = CGET BUFFER ( hdl3 , 0 , 255 );
   PRINT ( p ) ;
 Wend
Wend
END SUB
```

STATION_FILTER Example

Applies To

This example exercises modes of the STATION_FILTER command.

```
SUB Main()
populationcreate();
station_filter("CLEAR",2);
'To check
station_filter("APPLY",2,"POP01");

'To apply to Log Viewers
station_filter("DUMP",2);
END SUB

'Mode CREATE or Mode 1 (syntax 1)

SUB populationcreate()
DIM strString1 as Str;
DIM lngBuffer as long ;

'Preparing the buffer
lngBuffer = ALLOC_BUFFER(4096) ; 'buffer handle
lngBuffer = FILETOBUF("popu1.dat");
population("CREATE",lngBuffer);

'Select variables for domains DOM1 and DOM2
'excluding text variables
free_buffer(lngBuffer) ;
END SUB

'Mode DUMP or Mode 3 (syntax 2)

SUB populationdump()
population("DUMP","POP01");
END SUB
```

SUBENDSUB Example

Applies To

This example shows a main SUB calling procedures. There are also calls among these and other procedures, with and without parameters.

```
SUB Main()
DIM intResult as integer;

procedure1();
procedure2("parameter1");
intResult = procedure4();
PRINT("Return from Procedure4 = ",intResult);
END SUB

SUB procedure1()
'procedure without parameters
PRINT("Procedure1");
END SUB

SUB procedure2(strParam1)
'procedure with one parameter
PRINT("Procedure2 parameter: ",strParam1);
procedure3("parameter1","parameter2");
END SUB

SUB procedure3(strParam1,strParam2)
'procedure with two parameters
PRINT("Procedure3 parameters: ",strParam1,strParam2);
END SUB

SUB procedure4()
'procedure without parameters
PRINT("Procedure4");
Return(3);
END SUB
```

The results are:

```
Procedure1
Procedure2 parameter: parameter1
Procedure3 parameters : parameter1 parameter2
Procedure4
Return from Procedure4 = 3
```

SVALA (Alarm Information) Examples

Applies To

Example 1

This example exercises the SVALA command.

Variables required:

- @ALAEND is a variable of type STATE
- @NB_ALARMS is a variable of type REGISTER

The program also requires these global variables to be declared:

- DIM hdbuffer As Long;
- DIM hparam As Long;

The module "Main" must be run first to trigger an event before module SvAla1 is run.

```
SUB Main()
EVENT("ADDPROG","@ALAEND","ALL>1","sval.SCB","","putfile");
END SUB

SUB SvAla1()
DIM GFilter As STR;
DIM GFormat As STR;
DIM GState1 As STR;
DIM GState2 As STR;
DIM GState3 As STR;
DIM GState4 As STR;
DIM GState5 As STR;
DIM GState6 As STR;
DIM Gminmax As STR;
DIM Gmin As STR;
DIM Gmax As STR;
DIM chParam As STR;
chParam = "";
hdbuffer = ALLOC BUFFER ( 48000 );
hparam = ALLOC_BUFFER ( 4000 );

'   LIMITATION ON LINES
chParam = ADDSTRING(chParam,"0" );
'No. of lines ( 0 - no limitations,
' if this is not the size of the buffer )
chParam = ADDSTRING(chParam , ",");
'field separator

' MODE taking into account MIN & MAX
GMinMax="0";
chParam = ADDSTRING(chParam , GMinMax);
' equivalent - check all Min and Max boxes
chParam = ADDSTRING(chParam , ",");
'field separator

' MIN LEVEL
Gmin = "0";
chParam = ADDSTRING(chParam , Gmin);
'Priority, min
chParam = ADDSTRING(chParam , ",");
'field separator

' MAX LEVEL
GMax="29";
chParam = ADDSTRING(chParam , GMax);
'Priority, max
```

```
chParam = ADDSTRING(chParam , ",");
'field separator

'     TYPES OF ALARM
GState1 = "1"; 'taken into account
chParam = ADDSTRING(chParam , GState1);
'taken into account present not ack
chParam = ADDSTRING(chParam , ",");
'field separator
GState2 = "1"; 'taken into account
chParam = ADDSTRING(chParam , GState2);
'present ack
chParam = ADDSTRING(chParam , ",");
'field separator
GState3="1"; 'taken into account
chParam = ADDSTRING(chParam , GState3);
'Hidden not ack
chParam = ADDSTRING(chParam , ",");
'field separator
GState4="1"; 'taken into account
chParam = ADDSTRING(chParam , GState4);
'Inactive
chParam = ADDSTRING(chParam , ",");
'field separator
GState5 = "1"; 'taken into account
chParam = ADDSTRING(chParam , GState5);
'Invalid
chParam = ADDSTRING(chParam , ",");
'field separator
GState6 = "0"; 'not taken into account
chParam = ADDSTRING(chParam , GState6);
'User masked
chParam = ADDSTRING(chParam , ",");
'field separator

'     FILTERING
'GFilter = "";
GFilter = "=(#A1 == DOM1)||(#A1 == DOM2)";
'GFilter = "=(#A1 == DOM1)";
chParam = ADDSTRING(chParam,GFilter);
'filter variables with Domain DOM1 or DOM2
chParam = ADDSTRING(chParam , ",");
'field separator

'     FORMAT
GFormat="#D/#M/#Y #h:#m:#s #t #@A1 #@A2";
chParam = ADDSTRING(chParam, GFormat );
'Format

'     MODIFYING THE BUFFER OF PARAMETERS
PUT_BUFFER(hparam,0,chParam);
PRINT("Extraced by SVALA :",SVALA(1,hdbuffer,hparam,"ALAEND",1,"NB_ALARMS"));
END SUB

SUB putfile()
BUFTOFILE (hdbuffer, "SvAlaResult","USEFULL_PART","Append");
FREE_BUFFER(hdbuffer);
FREE_BUFFER(hparam);
@ALAEND = 0;
END SUB
```

Example 2

This is another program to extract information on alarms. It uses the CYCLIC and EVENT commands to activate SVALA functions.

```
DIM hdl As LONG;
DIM hparam As LONG;
DIM hdl2 As LONG;
DIM hdl3 As LONG;
DIM GFilter As STR;
DIM GFormat As STR;
DIM Gstate1 As STR;
DIM Gstate2 As STR;
DIM Gstate3 As STR;
DIM Gstate4 As STR;
DIM Gstate5 As STR;
DIM Gstate6 As STR;
DIM Gminmax As STR;
DIM Gmin As STR;
DIM Gmax As STR;
DIM Lock As INTEGER;

SUB Main()
hdl = 0;
hdl2 = 0;
hdl3 = 0;
hparam = 0;
Lock = 0 ;
END SUB

SUB AlarmExtract()
If ( Lock == 1 ) Then
PRINT ( "request in progress " );
CYCLIC ( "DEL" , 2 , "SvAla" ,"", "Filter2" );
CYCLIC ( "ADDPROG" , 2 , "SvAla" ,"", "Filter2" );
Return;
End If
Lock = 1 ;
@STATE = 0;
@ANA = 0;
GFilter = "";
GFilter = ADDSTRING(GFilter, @FILTER);
GFormat = "";
GFormat = ADDSTRING(GFormat, @ALAFORMAT);
GMinMax = FORMAT ( "%d" , @MINMAX );
GMin1 = FORMAT ( "%d" , @MIN );
GMin2 = FORMAT ( "%d" , @MAX );
GState1 =FORMAT ( "%d" , @ETAT1 );
GState2 =FORMAT ( "%d" , @ETAT2 );
GState3 =FORMAT ( "%d" , @ETAT3 );
GState4 =FORMAT ( "%d" , @ETAT4 );
GState5 =FORMAT ( "%d" , @ETAT5 );
GState6 =FORMAT ( "%d" , @ETAT6 );
DoAlarmExtract();
END SUB

SUB DoAlarmExtract()
DIM i As Integer;
DIM chParam As Str;
chParam = "";
If ( hdl == 0 ) Then
hdl = ALLOC_BUFFER ( 16000 );
End If
If ( hparam == 0 ) Then
hparam = ALLOC_BUFFER ( 2000 );
End If

' Number of lines. 0 = no limitation
chParam = ADDSTRING(chParam,"0," ); '
' Level filter (0,1 or 2)
```

```
chParam = ADDSTRING(chParam , GMinMax
chParam = ADDSTRING(chParam , ",");
' Minimum priority
chParam = ADDSTRING(chParam , Gmin)
chParam = ADDSTRING(chParam , ",");
' Maximum priority
chParam = ADDSTRING(chParam , GMax);
chParam = ADDSTRING(chParam , ",");
' Alarms on and not acknowledged
chParam = ADDSTRING(chParam , GState1);
chParam = ADDSTRING(chParam , ",");
' Alarms on and acknowledged
chParam = ADDSTRING(chParam , GState2);
chParam = ADDSTRING(chParam , ",");
' Alarms off and not acknowledged
chParam = ADDSTRING(chParam , GState3);
chParam = ADDSTRING(chParam , ",");
' Alarms off
chParam = ADDSTRING(chParam , GState4);
chParam = ADDSTRING(chParam , ",");
' Invalid alarms
chParam = ADDSTRING(chParam , GState5);
chParam = ADDSTRING(chParam , ",");
' Alarms masked by the operator
chParam = ADDSTRING(chParam , GState6);
chParam = ADDSTRING(chParam , ",");
' Filter - Alarms of nature ELEC and starting with the name WORKSHOP
GFilter = "=(#t BEG WORKSHOP)&&(#A2 == ELEC)";
chParam = ADDSTRING(chParam,GFilter);
' Line format
chParam = ADDSTRING(chParam, GFormat );
PUT_BUFFER(hparam,0,chParam);
i=EVENT("ADDPROG","ETAT",1,"SVALA","","RcvEvt" );
i = SVALA ( "EXTRACT" , hdl, hparam , "ETAT" , 1 , "ANA" );
If ( i < 1 ) Then
PRINT (" i " , i );
End If
END SUB

SUB RcvEvt()
DIM i As Integer;
DIM j As Integer;
DIM p As Str;
DIM sep As Str;
DIM sep2 As Str;
DIM lig As Integer;
sep = "\n";
sep2 = ",";
If ( hdl2 == 0 ) Then
hdl2 = ALLOC_BUFFER ( 1000 );
End If
If ( hdl3 == 0 ) Then
hdl3 = ALLOC_BUFFER ( 1000 );
End If
SEQ_BUFFER ( "BEGIN" , hdl ); ' Start of buffer
i = 1;
While ( i != 0)
i = SEQ_BUFFER ( "NEXTFIELD" , hdl , sep , hdl2, 1000);
p = CGET_BUFFER ( hdl2 , 0 , 255 );
j = SEQ_BUFFER ( "BEGIN" , hdl2 );
j = 1;
While ( j != 0)
j = SEQ_BUFFER ( "NEXTFIELD" , hdl2 , sep2 , hdl3 , 1000 );
p = CGET BUFFER ( hdl3 , 0 , 255 );
Wend
```

```
Wend
Lock = 0 ;
i=EVENT("DEL","STATE",1);
END SUB
```

SVLOG Example

Applies To

This example shows the use of the SVLOG instruction to extract data logged in the previous hour from the Log Filter LIST1.

```
' Constants for log data type:

const VALTRANSEV = 1; 'Bit transition
const ALARMACQEV = 2; 'Alarm acknowledge
const WRITECMDEEV = 4; 'Send a command
const WRITECTRLEV = 8; 'Send a control value
const WRITETEXTEV = 16; 'Send text
const WRITERECIPESEV = 32; 'Send a recipe
const STARTEXP = 128; 'Log on
const STOPEXP = 256; 'Log off
const TRANSETAT_0 = 64; 'Bit transition to 0
const TRANSETAT_1 = 512; 'Bit transition to 1
const TRANSETAT_NS = 1024; ' Bit transition to NS
const TRANSETAT_ALLALL = 1600; 'All bit transitions
const TRANSALA_PRSACQ = 2048; 'Alarm On Ack
const TRANSALA_PRSNOACQ = 4096; 'Alarm On Nack
const TRANSALA_ABSACQ = 8192; 'Alarm Off
const TRANSALA_ABSNOACQ = 16384; 'Alarm Off Nack
const TRANSALA_NS = 32768; 'Alarm NS.
const TRANSALA_ACQOPERATOR = 65536; 'Operator ack
const TRANSALA_ALLALL = 63488; 'All alm transitions
const FORCAGEPROG = 131072; 'Run program
const TRANSALA_OFF = 262144; 'Alarm on
const TRANSALA_ON = 524288; 'Alarm off

' Global variable
DIM HandleDest As long; 'Buffer handle


SUB Main()
  DIM Ret as long;
  DIM HandleParam as long;
  DIM Hd1 as double;
  DIM Hd2 as double;
  DIM Mask as long;
  DIM ChParam as STR;
  EVENT ("ADDPROG", "Extract.Fin", 1, "LOG", "", "EndExtract");

  ' Create the filter parameters buffer

  HandleParam = alloc_buffer(1000);
  Mask=0;
  Mask = LOGICAL("OR",Mask, WRITECTRLEV);
  Mask = LOGICAL("OR",Mask, TRANSETAT_ALLALL);
  Hd1 = DateTimeValue();
  Hd2 = Hd1 - 3600000;
  chParam = addstring(chParam,"LIST1");
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, format("%.0f",Hd2));
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, format("%.0f",Hd1));
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, "0"); ' Max lines
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, "0"); ' Direction
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, TOC(Mask)); ' Mask
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, "0"); ' Min priority
```

```
  chParam = addstring(chParam,",");
  chParam = addstring(chParam, "15"); ' Max priority
  chParam = addstring(chParam,",");
  ' Selection filter
  chParam = addstring(chParam, "=(#A1 == DOM1)||(#A1 == DOM2)");
  chParam = addstring(chParam,",");
  ' Output format
  chParam = addstring(chParam, "#h:#m:#s #E #O #T #t #C #c #d #n");
  put_buffer(HandleParam,0,chParam);

  ' Create the output buffer
  ' ----------------------------------------------------
  HandleDest = alloc_buffer(20000);
  ' Execute the extract function
  ' ----------------------------------------------------
  Ret = SvLog("Extract", HandleDest, HandleParam,
"Extract.Fin","Extract.Status","Extract.Lignes", "Extract.HDeb","Extract.HFin");
  If (Ret == 0) then
  PRINT ("Error in extract function");
  Event ("DelProg", "EndExtract", 1);
  End if

  ' Free buffers
  Free Buffer(HandleParam);
END SUB

' Function to be executed when extraction is complete

SUB EndExtract()
  DIM NomApp as STR;
  DIM Ret as INTEGER;
  PRINT ("Extract complete");
  If (Extract.Status == 0) then
    PRINT ("All lines extracted");
 Else
   PRINT ("Buffer full - some data may be missing");
 End if
 PRINT ("The first archive date is ", DateTimeString(Extract.HDeb));
 PRINT ("The last archive date is ", DateTimeString (Extract.HFin));

 ' Convert the buffer to a file
 BUFTOFILE(HandleDest,"C:\\EXTRACT.TXT");
 FREE_BUFFER(HandleDest);

 ' Start WordPad to display the extracted information
 NomApp="C:\\WORDPAD.EXE C:\\EXTRACT.TXT";
 Ret=APPLICATION("LOAD",NomApp);
END SUB
```

SVSQL Database Example

Applies To

This example uses a mimic (SQLTEST) containing two arrays (TITLE and DATA) in which the results of executing a SQL command are placed.

The User enters the name of the database in the text variable @DATABASE.NAME and the SQL command in the variable @DATABASE.SQL. The variable @DATABASE.ERROR is used to report the status and error situation to the User.

```
DIM HBase as long; 'Handle for database connection
DIM RetSQL as long; 'Return from SVSQL command

SUB Main ()
  HBase = 0;
END SUB

SUB Connect()
@DATABASE.ERROR = ADDSTRING("Connecting to database", UCase(@DATABASE.NAME));
Delay(0.1);
ClearArray();
If (HBase > 0) Then
    @DATABASE.ERROR = "Database already connected";
Return(0);
Else
 If (CmpString(@DATABASE.NAME, "") ==0) Then
   @DATABASE.ERROR = "Please enter database name";
     RETURN(0);
 Else
     HBase = SVSQL("CONNECT",@DATABASE.NAME);
 End If
End If
If (HBase > 0) then
   @DATABASE.ERROR = "Database connected OK";
Else
   @DATABASE.ERROR = "Database failed to connect";
End If
Print(Hbase);
END SUB
```

```
SUB Disconnect()
 Print (Hbase);
 @DATABASE.ERROR = AddString("Disconnecting database " , UCase(@DATABASE.NAME));
 Delay(0.1);
 If (HBase <= 0) Then
 @DATABASE.ERROR = "Database not connected";
   RETURN(0);
 End if
 RetSQL = SVSQL("DISCONNECT",HBase);
 If (RetSQL == -1) Then
   @DATABASE.ERROR = "Error disconnecting database";
   RETURN(0);
 Else
   If (RetSQL == -2) Then
     @DATABASE.ERROR = "Bad database handle";
     RETURN(0);
   Else
     @DATABASE.ERROR = "Disconnected OK";
     Hbase = 0;
     ClearArray();
   End If
 End If
END SUB
```

```
SUB ExecuteSQL()
 DIM HBufErr as long; ' Handle of error buffer
 DIM HField as long; ' Handle of SQL buffer
 DIM NbTableCol As Integer;
 DIM NbCol As Long;
 DIM i As Integer;
 DIM j As Integer;
 Const NbMaxRow = 15;
 Const NbMaxCol = 5;
```

```
 @DATABASE.ERROR = "Executing query";
 Delay(0.1);
 If (HBase <= 0) Then
   @DATABASE.ERROR = "Database not connected";
   RETURN(0);
 End If

' Initialise buffers

 HBufErr = alloc_buffer(255);
 HField = alloc_buffer(255);
 ClearArray();

' Search for information

 RetSQL = SVSQL("EXECUTE", HBase, @DATABASE.SQL);
 If (RetSQL < 0) Then
   RetSQL = SVSQL("ERROR", HBase , HBufErr);
   @DATABASE.ERROR = Cget_Buffer(HBufErr, 0, RetSQL, 1);
   RETURN(0);
 Else
   NbCol = SVSQL("NUMCOL", HBase);
 End if
 If (NbCol > NbMaxCol) then
   NbTableCol = NbMaxCol;
 Else
   NbTableCol = TOI(NbCol);
 End if

' Get column names from table

 For (i=1 ; i<=NbTableCol; i++)
   RetSQL = SVSQL("GETCOLNAME", HBase , I, HField);
   If (RetSQL < 0) then
     RetSQL = SVSQL("ERROR", HBase , HBufErr);
     @DATABASE.ERROR = Cget_Buffer(HBufErr, 0, RetSQL, 1);
     RETURN(0);
   End If
   SELECTOR("PUTCELL", "SQLTEST", "", "TITLE", 0, i-1, Cget_Buffer(HField, 0, RetSQL,
1));
 Next i;

 For (i=NbTableCol+1 ; i<=5; i++)
 SELECTOR("PUTCELL", "SQLTEST", "", "TITLE", 0, i-1, "---");
 Next i;

' Retrieve values and write to table

 For(i=1; i<=NbMaxRow; i++)
   RetSQL = SVSQL("FETCH", HBase);
   If (RetSQL != 0) Then
     BREAK();
   End If
   DELAY(0.1);
   For (j=1 ; j<=NbTableCol; j++)
```

```
     RetSQL = SVSQL("GETCOL", HBase , j, HField);
     SELECTOR("PUTCELL", "SQLTEST", "", "DATA", i-1, j-1, Cget Buffer(HField, 0, RetSQL,
1));
   Next j;
 Next i;
 @DATABASE.ERROR = "Query complete";
END SUB

SUB ClearArray()
 SELECTOR("CLEAR", "SQLTEST", "", "TITLE");
 SELECTOR("CLEAR", "SQLTEST", "", "DATA");
END SUB
```

SVTREND Examples

Applies To

Example 1

This program extracts trend (Historic) data for a list of variables, between two dates.

```
SUB GetTrend()
DIM d_Date1 As Double;
DIM d_Date2 As Double;
DIM d_Period As Double;
DIM l_Res As Long;
DIM s_Misc As Str;
gn_Ind = 0;
d_Date1 = DATETIMEVALUE("16/02/2003", "08:00:00:000");
d_Date2 = DATETIMEVALUE("17/02/2003", "21:00:00:000");
If ( l_Result == 0 ) Then
 l_Result = ALLOC_BUFFER ( 8000 );
End If
 d_Period = 3600;
 s_Div = " ,,@STATE,@CURVAR";
 l_Res = SVTREND ( "GETTREND" , "STATE1,STATE2,MES1,MES2,MES3,MES4,MES5,MES6",
  "TREND",d_Date1, d_Date2, d_Period, 2, l_Result, s_Misc );
END SUB

SUB GetNext()
DIM l_Res As Long;
DIM s As Str;
gn_Ind++;
PRINT ("Pass", gn_Ind );
PRINT ("CURVAR" , @CURVAR );
s = CGET_BUFFER(l_Result , 0 , 1 );
If (CMPSTRING (s, "") != 0) Then
PRINT ("WRITE");
BUFTOFILE(l_Result, TOC(gn_Ind), "USEFULL_PART" );
@STATE = 0;
l_Res = SVTREND ( "GETNEXTBUFFER" , l_Result );
End If
END SUB
```

When the buffer is full, the state variable *LogVar* is set to 1.

That allows an event to be triggered (see the verb EVENT) so as to restart the extraction of the remaining data. When there are no more data to extract, *Handle_Result* becomes null.

When *DisplayMode* is 1, the data are extracted using the list of variables in the buffer *Handle_Listvar* to the format defined by *Format*.

Data example 1

This shows the contents of the buffer, using a sampling period of 1 second in Mode 1. There were four variable names in the buffer pointed to by *ListHandle*.

StrVar = STATE1:

```
20030226145546675,0
20030226145547675,0
20030226145548675,0
20030226145549675,1
20030226145550675,0
20030226145551675,1
```

StrVar = STATE2:

```
20030226145546675, 1
20030226145547675, 1
20030226145548675, 0
20030226145549675, 1
```

```
20030226145550675, 0
20030226145551675, 0
```

StrVar = ANA1:

```
20030226145546675, -0.927184
20030226145547675, -0.927184
20030226145548675, -0.898794
20030226145549675, -0.882948
20030226145550675, -0.882948
20030226145551675, -0.882948
```

StrVar = ANA2:

```
20030226145546675, 64.99774
20030226145547675, 65.007904
20030226145548675, 65.018062
20030226145549675, 65.028214
20030226145550675, 65.038361
20030226145551675, 65.038361
```

When *DisplayMode* is 2, the sampled data for each variable will be contained on the same line preceded by its time and date stamp.

Data example 2

The sampling period was 1000mS (1 second). There were four variable names in the buffer pointed to by *ListHandle*.

```
20030226145546675,0,1,-0.927184,64.99774
20030226145547675,0,1,-0.927184,65.007904
20030226145548675,0,0,-0.898794,65.018062
20030226145549675,1,1,-0.882948,65.028214
20030226145550675,0,0,-0.882948,65.038361
20030226145551675,1,0,-0.882948, 65.038361
```

Example 2

This example calculates mean, minimum and maximum temperatures.

REGISTERS.DAT is a file to hold the names of register variables to be processed.

```
const MAX_VARIABLES = 1;
const BUFSIZE = 10000; ' size of buffer to take the data
DIM hDest AS LONG;
DIM hNames AS LONG;
DIM hBranch AS LONG;
DIM hVariables AS LONG;
DIM nb_variables AS INTEGER;
DIM nbPoints [8] AS SINGLE;
DIM SumValue [8] AS SINGLE;
DIM MinValue [8] AS SINGLE;
DIM MaxValue [8] AS SINGLE;
DIM AveValue [8] AS SINGLE;
DIM flagEnd AS INTEGER;
DIM filesource AS STR;
DIM lock AS INTEGER;

SUB main ()
hDest = alloc_buffer (BUFSIZE);
filesource = "REGISTERS.DAT";
lock = 0;
END SUB

'===============  RequestSvTrEND ===============
' Function : Prepare the parameters for processing
' Call : By User
' Branch : None
' Argument : None
'
```

```
SUB RequestSvTrEND ()
IF (lock == 0) THEN
  lock = 1;
  ' allocate the buffers
  hNames = filetobuf (filesource);
  hBranch = filetobuf (filesource);
  hVariables = alloc_buffer (1024);
  SEQ_BUFFER ("BEGIN", hNames);
  SEQ_BUFFER ("BEGIN", hBranch);
  EVENT ("ADDPROG", "RESULT.DONE", 1, "RESULTS.TRT", "", "compute");
  @RESULT.DONE = 0;
  processing ();
END IF
END SUB

'===============  Processing  ===============
' Function : Prepare list of variables to process
' Call: Sub RequestSvTrend
' Branch : None
' Argument : None
'
SUB processing ()
DIM ch AS STR;
DIM i AS INTEGER;
DIM offset AS INTEGER;
DIM StartTime AS DOUBLE;
DIM EndTime AS DOUBLE;
DIM selection AS DOUBLE;
selection = 0;
StartTime= DATETIMEVALUE (@RESULT.START.DATE, @RESULT.START.TIME);
EndTime   = DATETIMEVALUE (@RESULT.END.DATE, @RESULT.END.TIME);
i = 0;
offset = 0;
nb_variables = 0;
' composition of a buffer containing N variables
WHILE (i++ < MAX_VARIABLES)
  ch = SEQ_BUFFER ("NEXTFIELD", hNames, "\N", "STR");
  IF (CMPSTRING (ch, "")) THEN
    PUT_BUFFER (hVariables, offset, format ("%s,", ch));
    offset = offset + LEN (ch) + 1;
    nb_variables++;
  ELSE
    break;
  END IF
WEND
ch = SEQ_BUFFER ("NEXTFIELD", hNames, "\N", "STR");
flagEnd = (CMPSTRING (ch, "") == 0);
ch = SEQ_BUFFER ("PREVFIELD", hNames, "\N", "STR");
' remove last separator
PUT_BUFFER (hVariables, offset, "");
SVTREND ("GETTREND", hVariables, "", StartDate, EndTime, selection, 2, hDest,
",,RESULT.DONE,");
END SUB
'
'===============  Compute  ===============
'
' Function : Process the resulting buffer
' Call Event : RESULT.DONE
' Branch : None
' Argument : None
'
SUB compute ()
DIM ch AS STR;
DIM i AS INTEGER;
DIM hResult AS LONG;
```

```
DIM value AS SINGLE;
DIM BufLine AS LONG;
' Are there data?
IF (CMPSTRING (CGET_BUFFER (hDest, 0, 1), "")) THEN
  hResult = SEQ_BUFFER ("CRTOCRLF", hDest, "NEW_BUFFER");
  SEQ_BUFFER ("BEGIN", hResult);
  SEQ_BUFFER ("SEEKFIELD", 1, hResult, ",");
  FOR (i = 0; i < nb_variables; i++)
    value = sval (SEQ_BUFFER ("NEXTFIELD", hResult, ",", "STR"));
    nbPoints [i] = 0; ' number of points found
    SumValue [i] = 0; ' sum of values
    MinValue [i] = value; ' minimum of values
    MaxValue [i] = value; ' maximum of values
    AveValue [i] = 0; ' mean
  NEXT
  BufLine = alloc_buffer (500);
  SEQ_BUFFER ("BEGIN", hResult);
  WHILE (SEQ_BUFFER ("NEXTFIELD", hResult, "\N", BufLine))
    CalcStatistics (BufLine);
  WEND
  FREE_BUFFER (hResult);
  WHILE (svtrEND ("GETNEXTBUFFER", hDest))
    hResult = SEQ_BUFFER ("CRTOCRLF", hDest, "NEW_BUFFER");
    SEQ BUFFER ("BEGIN", hResult);
    WHILE (SEQ_BUFFER ("NEXTFIELD", hResult, "\N", BufLine))
      CalcStatistics (BufLine);
    WEND
    FREE_BUFFER (hResult);
  WEND
  FREE_BUFFER (BufLine);
  FOR (i = 0; i < nb_variables; i++)
  ' compute the mean
    IF (nbPoints [i] != 0) THEN
      AveValue [i] = SumValue [i] / nbPoints [i];
    END IF
    ' retrieve the variable name (branch)
    ch = SEQ_BUFFER ("NEXTFIELD", hBranch, "\N", "STR");
    ' assign values in database
    ?ch.MIN = MinValue [i];
    ?ch.MAX = MaxValue [i];
    ?ch.MOY = AveValue [i];
  NEXT
END IF

' processing ended?
IF (flagend) THEN
  lock = 0;
  FREE_BUFFER (hNames);
  FREE_BUFFER (hBranch);
  FREE_BUFFER (hVariables);
  EVENT ("DEL", "RESULT.DONE", 1, "RESULTS.TRT", "", "compute");
  ' close the schedule WINDOW at start and end
  WINDOW ("CLOSE", "RESULTS", "");
  ' open the window if mean temperatures, min and max
  WINDOW ("OPEN", "MEANS", "");
ELSE
  processing ();
END IF
END SUB
'
'==============  CalcStatistics  ==============
' Function : Compute the sum
' Call :  Sub Process
' Branch : None
' Arg : Buffer containing the line to process
```

```
'
SUB CalcStatistics ( handle )
DIM ch AS STR;
DIM i AS INTEGER;
DIM flag AS INTEGER;
DIM value AS SINGLE;
i    = 0;
flag = 1;
SEQ_BUFFER ("BEGIN", handle);
SEQ_BUFFER ("SEEKFIELD", 1, handle, ",");
WHILE (flag)
  ch = SEQ_BUFFER ("NEXTFIELD", handle, ",", "STR");
  flag = CMPSTRING (ch, "");
  IF ((flag) && CMPSTRING (ch, "?")) THEN
    ' number of valid points found
    nbPoints [i]++;
    ' retrieve the value at a point
    value = sval (ch);
    ' do the sum
    SumValue [i] = SumValue [i] + value;
    ' retrieve the min
    IF (value < MinValue [i]) THEN
      MinValue [i] = value;
    END IF
    ' retrieve the max
    IF (value > MaxValue [i]) THEN
      MaxValue [i] = value;
    END IF
  END IF
  i++;
WEND
END SUB
```

SYSTEM Examples

Applies To

Example 1

This example exercises several modes of the SYSTEM command.

It requires creation of a User "user1" with a password "pass1".

```
'Create a control variable of type State: @STATE01
'Mode SETDATE or Mode 1 (syntax 1)

SUB systemsetdate()
'Declare variables
DIM intResult as integer;
'change the system date
intResult = SYSTEM("SETDATE", 28, 06, 2002);
END SUB

'Mode SETTIME or Mode 2 (syntax 2)

SUB systemsettime()
'Declare variables
DIM intResult as integer;
'change the time of the system
intResult = SYSTEM("SETTIME", 21, 03, 00);
END SUB

'Mode GETDATE or Mode 3 (syntax 3)

SUB systemgetdate()
'Declare variables
DIM strResult as Str;
'Retrieve the system date
strResult = SYSTEM("GETDATE");
PRINT("GETDATE : ",strResult);
END SUB

'Mode GETTIME or Mode 4 (syntax 3)

SUB systemgettime()
'Declare variables
DIM strResult as Str;
'Retrieve the system time
strResult = SYSTEM("GETTIME");
PRINT("GETTIME : ",strResult);
END SUB

'Mode SETREGION or Mode 5 (syntax 4)

SUB systemsetregion()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("SETREGION",1);
PRINT("SETREGION : ",intResult);
END SUB

'Mode SETSYSREGION or Mode 6 (syntax 4)

SUB systemsetsysregion()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("SETSYSREGION",1);
PRINT("SETSYSREGION : ",intResult);
END SUB
```

```
'Mode GETREGION or Mode 7 (syntax 7)

SUB systemgetregion()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("GETREGION");
PRINT("GETREGION : ",intResult);
END SUB

'Mode GETSYSREGION or Mode 8 (syntax 7)

SUB systemgetsysregion()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("GETSYSREGION");
PRINT("GETSYSREGION : ",intResult);
END SUB

'Mode SYSTEM or Mode 10 (syntax 5)

SUB systemsystem()
'Declare variables
DIM intResult as integer;
'starting Windows Calculator
intResult = SYSTEM("SYSTEM","%SystemRoot%\\System32\\calc.exe");
PRINT("SYSTEM : ",intResult);
END SUB

'Mode LOGIN or Mode 11 (syntax 6)

SUB systemlogin()
'Declare variables
DIM intResult as integer;
'starting Windows Calculator
intResult = SYSTEM("LOGIN","user1", "pass1");
PRINT("LOGIN : ",intResult);
END SUB

'Mode LOGOUT or Mode 12 (syntax 7)

SUB systemlogout()
'Declare variables
DIM intResult as integer;
'starting Windows Calculator
intResult = SYSTEM("LOGOUT");
PRINT("LOGOUT : ",intResult);
END SUB

'Mode SETDATETIME or Mode 13 (syntax 8)

SUB systemsetdatetime()
'Declare variables
DIM intResult as integer;
'starting Windows Calculator
intResult = SYSTEM("SETDATETIME","28/06/02","21:38:00");
PRINT("SETDATETIME : ",intResult);
END SUB

'Mode LANGUAGE or Mode 14 (syntax 7)

SUB systemlanguage()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("LANGUAGE");
```

```
PRINT("LANGUAGE : ",intResult);
END SUB

'Mode USER or Mode 15 (syntax 9)

SUB systemuser()
'Declare variables
DIM strResult as Str;
strResult = SYSTEM("USER","NAME");
PRINT("USER NAME : ",strResult);
END SUB

'Mode USER or Mode 15 (syntax 10)

SUB systemuser2()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("USER","RIGHTS1");
PRINT("USER RIGHTS1 : ",intResult);
PRINT("USER RIGHTS2 : ",SYSTEM("USER","RIGHTS2"));
PRINT("USER RIGHTS3 : ",SYSTEM("USER","RIGHTS3"));
PRINT("USER RIGHTS4 : ",SYSTEM("USER","RIGHTS4"));
PRINT("USER RIGHTS5 : ",SYSTEM("USER","RIGHTS5"));
PRINT("USER RIGHTS6 : ",SYSTEM("USER","RIGHTS6"));
END SUB

'Mode EXIT or Mode 16 (syntax 7)

SUB systemxit()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("EXIT");
PRINT("EXIT : ",intResult);
END SUB

'Mode REPAINT or Mode 17 (syntax 7)

SUB systemrepaint()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("REPAINT");
PRINT("REPAINT : ",intResult);
END SUB

'Mode PASSWORD or Mode 21 (syntax 11)

SUB systempassword()
'Declare variables
DIM strResult as Str;
strResult = SYSTEM("PASSWORD","user1");
PRINT("PASSWORD : ",strResult);
END SUB

'Mode USERNAME or Mode 22 (syntax 11)

SUB systemusername()
'Declare variables
DIM strResult as Str;
strResult = SYSTEM("USERNAME","pass1");
PRINT("USERNAME : ",strResult);
END SUB

'Mode MKDIR or Mode 23 (syntax 12)

SUB systemmkdir()
```

```
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("MKDIR","e:\\test1");
PRINT("MKDIR : ",intResult);
END SUB

'Mode RMDIR or Mode 24 (syntax 12)

SUB systemrmdir()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("RMDIR","e:\\test1");
PRINT("RMDIR : ",intResult);
END SUB

'Mode OPERATORMODE or Mode 25 (syntax 14)

SUB systemoperatormode()
'Declare variables
DIM intResult as integer;
'in a Log Viewer
'validate the current User's logon
'intResult = SYSTEM("OPERATORMODE", 1);
PRINT("OPERATORMODE : ",intResult);
@STATE01=1;
@STATE01=0;
'in a Log Viewer
'prevent current User logging on
intResult = SYSTEM("OPERATORMODE", 0);
'PRINT("OPERATORMODE : ",intResult);
@STATE01=0;
@STATE01=1;
END SUB

'Mode GETDRIVETYPE or Mode 26 (syntax 13)

SUB systemgetdrivetype()
'Declare variables
DIM intResult as integer;
intResult = SYSTEM("GETDRIVETYPE", "C:");
PRINT("GETDRIVETYPE : ",intResult);
END SUB

'Mode GETDISKSIZE or Mode 27 (syntax 13)

SUB systemgetdisksize()
'Declare variables
DIM lngResult as long;
lngResult = SYSTEM("GETDISKSIZE", "C:");
PRINT("GETDISKSIZE : ",lngResult);
END SUB

'Mode GETDISKFREESPACE or Mode 28 (syntax 13)

SUB systemgetdiskfreespace()
'Declare variables
DIM lngResult as long;
lngResult = SYSTEM("GETDISKFREESPACE", "C:");
PRINT("GETDISKFREESPACE : ",lngResult);
END SUB

'Mode GETDIRECTORYSIZE or Mode 29 (syntax 13)

SUB systemgetdirectorysize()
'Declare variables
```

```
DIM lngResult as long;
lngResult = SYSTEM("GETDIRECTORYSIZE", "C:\\WINNT");
PRINT("GETDIRECTORYSIZE : ",lngResult);
END SUB
```

Example 2

This example changes various system settings.

Change system time forward by one hour:

```
SUB Summer ()
Const OneHour = 3600000;
DIM LongTime As Double;
DIM NewTime As Str;
DIM NewDate As Str;
LongTime = DateTimeValue () + OneHour;
NewDate = DateTimeString(LongTime, "D");
NewTime = DateTimeString(LongTime, "T");
SYSTEM(13, NewDate, NewTime);
END SUB
```

Log a user onto the system:

```
SUB Boss ()
SYSTEM("LOGIN", "superuser", "bigboss");
END SUB

SUB AutoOff ()
If (Activity == 0) Then
SYSTEM("LOGOUT");
End If
END SUB ()
```

Manage a multi-screen system:

```
' START.PVB Loaded on startup of system
SUB Main ()
Program("PRELOAD", "setreg.pvb", "");
SYSTEM("SETSYSREGION", 2); ' two screens
END SUB

' SETREG.PVB
SUB Main ()
END SUB

SUB LeftWin (WinName)
SYSTEM("SETREGION", 1); ' Select left screen
WINDOW("OPEN", WinName);
END SUB

SUB RightWin (WinName)
SYSTEM("SETREGION", 2); ' Select right screen
WINDOW("OPEN", WinName);
END SUB
```

TEMPORARY_DB Example

Applies To

This example exercises several modes of the TEMPORARY_DB command.

```
'Mode OFF or Mode 0 (syntax 1)

SUB temporarydboff()
DIM intResult as integer;
intResult = TEMPORARY_DB("OFF");
END SUB

'Mode ON or Mode 1 (syntax 1)

SUB temporarydbon()
'enables creation of temporary variables in animations and symbols.
DIM intResult as integer;
intResult = TEMPORARY_DB("ON");
END SUB

'Mode ADDBIT or Mode 2 (syntax 2)

SUB temporarydbaddbit()
DIM intResult as integer;
intResult = TEMPORARY_DB("ADDBIT", "@STATEAB03", "label, language 1", "label, language
2");
END SUB

'Mode ADDREG or Mode 3 (syntax 3)

SUB temporarydbaddreg()
DIM intResult as integer;
intResult = TEMPORARY_DB("ADDREG", "@REGISTER01", "label, language 1", "label, language
2", -20, 35, "##.##u", "øC");
END SUB

'Mode ADDTXT or Mode 4 (syntax 4)

SUB temporarydbaddtxt()
DIM intResult as integer;
intResult = TEMPORARY_DB("ADDTXT", "@TEXTE01", "label, language 1", "label, language 2",
32);
intResult = TEMPORARY_DB("ADDTXT", "@TEXTE02", "label, language 1", "label, language 2",
32);
END SUB
```

TEXTBOX Example

Applies To

Example of the use of the TEXTBOX instruction.

```
Sub Search()

  Dim iFind As Integer;

'-------- Search for first occurrence of VCX
   iFind = TextBox("INDEXOF", "SearchEngine", "", "TextBox1", "VCX");

'-------- If found then select text to highlight it.
  If (iFind >= 0) Then
        TextBox("SELECTTEXT", "SearchEngine", "", "TextBox1", iFind, 3);
  End if

End Sub()
```

TEXTVAR Examples

Applies To

Example 1

This example exercises several modes of the TEXTVAR command.

It uses a file called "ReadMe.txt" stored in the TP folder of your current project.

It requires these variables:

- TEXT01 of type TEXT
- TEXT02 of type TEXT

```
SUB Main()
textvarbuftotext();
textvartexttobuf();
textvarfiletotext();
textvartexttofile();
textvartextcompare();
textvartextcopy();
textvartextlen();
END SUB

'Mode BUFTOTEXT or Mode 1 (syntax 1)

SUB textvarbuftotext()
DIM IntResult as integer;
DIM lngbuffer1 as long;
lngbuffer1 = FILETOBUF("ReadMe.txt");
IntResult = TEXTVAR("BUFTOTEXT", "TEXT01", lngbuffer1, 0, 0, 15);
free_buffer(lngbuffer1);
END SUB

'Mode TEXTTOBUF or Mode 2 (syntax 1)

SUB textvartexttobuf()
DIM IntResult as integer;
DIM strResult as Str;
DIM lngbuffer1 as long;
lngbuffer1 = ALLOC_BUFFER(250);
IntResult = TEXTVAR("TEXTTOBUF", "TEXT01", lngbuffer1, 0, 0, 15);
strResult = CGET_BUFFER(lngbuffer1,0,15,0);
PRINT("The returned string is: ",strResult);
free_buffer(lngbuffer1);
END SUB

'Mode FILETOTEXT or Mode 3 (syntax 2)

SUB textvarfiletotext()
DIM IntResult as integer;
IntResult = TEXTVAR("FILETOTEXT", "TEXT02", "ReadMe.txt", 0, 0, 15);
END SUB

'Mode TEXTTOFILE or Mode 4 (syntax 3)

SUB textvartexttofile()
DIM IntResult as integer;
DIM lngbuffer1 as long;
lngbuffer1 = FILETOBUF("ReadMe.txt");
IntResult = TEXTVAR("TEXTTOFILE", "TEXT01","text1.txt",0, 0, 15);
END SUB

'Mode TEXTCOMPARE or Mode 5 (syntax 4)
```

```
SUB textvartextcompare()
'Compare the first 4 characters
PRINT(TEXTVAR("TEXTCOMPARE","TEXT01","TEXT02",4));
END SUB

'Mode TEXTCOPY or Mode 6 (syntax 5)

SUB textvartextcopy()
DIM intResult as integer;
intResult = TEXTVAR("TEXTCOPY","TEXT01","TEXT03");
END SUB

'Mode TEXTLEN or Mode 7 (syntax 6)

SUB textvartextlen()
DIM intResult as integer;
intResult = TEXTVAR("TEXTLEN","TEXT01");
PRINT("Length of the character string: ",intResult);
END SUB
```

Example 2

This example sets and displays text variables.

It assumes that a mimic contains these text variables:

- mytext
- mytext2

The program uses short delays, so there will be pauses before results are displayed.
A file test.txt in project folder TP contains between 4 and 50 characters (here: "abc,123"). The program allocates a memory buffer and loads it from the text file.

```
SUB Main()
DIM hbuf as long;
PRINT("Open file result: ", FOPEN("test.txt", "r+"));
PRINT("Put to file result: ", FPUTS("test.txt", "abc,123"));
hbuf = FILETOBUF("test.txt");
PRINT("Buffer contains: ", CGET_BUFFER(hbuf, 0, 50, 0));

 'clear the text variables
@mytext = "";
@mytext2 = "";
DELAY(0.1); 'to apply value to variable
PRINT("Text variables mytext: ", @mytext, "; mytext2: ", @mytext2);

 'copy buffer contents to first variable
PRINT("BufToText all: ", TEXTVAR( "BUFTOTEXT", "mytext", hbuf, 0, 0, 0), " chars.");
DELAY(0.1); 'to apply value to variable
PRINT("mytext: ", @mytext, "; mytext2: ", @mytext2);

 'copy from variable mytext (4 chars. from offset 3) to end of file
PRINT("TextToFile: ", TEXTVAR("TEXTTOFILE", "mytext", "test.txt", 3, 1, 4), " chars.");

 'copy 3 characters from buffer at offset 4 to the other variable
PRINT("BufToText: ", TEXTVAR("BUFTOTEXT", "mytext2", hbuf, 0, 4, 3), " chars.");
DELAY(0.1); 'to apply value to variable
PRINT("Text variables mytext: ", @mytext, "; mytext2: ", @mytext2);
'compare the first 4 characters of the variables
PRINT("Compare result (0 means equal): ", TEXTVAR("TEXTCOMPARE", "mytext", "mytext2",
4));
'copy 3 characters from file to start of variable
PRINT("FileToText: ", TEXTVAR("FILETOTEXT", "mytext2", "test.txt", 0, 8, 3), " chars.");
DELAY(0.1); 'to apply value to variable
PRINT("Text variable mytext2: ", @mytext2);
```

```
 'copy variable mytext2 to mytext
PRINT("TextCopy result: ", TEXTVAR("TEXTCOPY", "mytext", "mytext2"));
DELAY(0.1); 'to apply value to variable
PRINT("Text variables mytext: ", @mytext, "; mytext2: ", @mytext2);
PRINT("Compare result (0 means equal): ", TEXTVAR("TEXTCOMPARE", "mytext", "mytext2",
4));

 'close the file and release the buffer
FCLOSE("test.txt");
FREE_BUFFER(hbuf);
END SUB
```

Results

The program displays these results in the Program Management window (F9, Show Results):

```
Open file result: 1
Put to file result: 1
Buffer contains: abc,123,123
Text variables mytext: ; mytext2:
BufToText all: 11 chars.
mytext: abc,123,123; mytext2:
TextToFile: 4 chars.
BufToText: 3 chars.
Text variables mytext: abc,123,123; mytext2: 123
Compare result (0 means equal): 1
FileToText: 3 chars.
Text variable mytext2: 123
TextCopy result: 1
Text variables mytext: 123; mytext2: 123
Compare result (0 means equal): 0
```

TREEVIEW Example

Applies To

This example contains a set of functions to support a pair of Tree-view form controls using the TREEVIEW instruction.

```
SUB Main()
END SUB

'declare variables:
SUB tvwSel()
DIM lRet As Long;
DIM stvwText As Str;
DIM stvwText1 As Str;
DIM stvwData As Str;
DIM stvwData1 As Str;

'retrieve contents of selected item in 1st control:
lRet = TREEVIEW ( "GETSELECTEDINDEX", "ctrl","","tvw" );
TREEVIEW ( "SETSELECTEDINDEX", "ctrl","","tvw1", lRet);
stvwText = TREEVIEW( "GETTEXT", "ctrl","","tvw", lRet);
stvwText1 = TREEVIEW( "GETTEXT", "ctrl","","tvw1", lRet);
stvwData = TREEVIEW( "GETUSERDATA", "ctrl","","tvw", lRet);
stvwData1 = TREEVIEW( "GETUSERDATA", "ctrl","","tvw1", lRet);

'put its contents in a frame:
SET ( "tvwText" , stvwText);
SET("tvwText1", stvwText1);
SET("tvwData", stvwData);
SET("tvwData1" , stvwData1);
SET("tvwCount" , TREEVIEW( "COUNT", "ctrl","","tvw") );
SET("tvwCount1" , TREEVIEW( "COUNT", "ctrl","","tvw1"));
SENDLIST ("BLOC");

'concatenate and load contents into a form control of each kind:
COMBOBOX ( "LOAD", "ctrl","","cbx2", ADDSTRING("cbx",stvwData));
LISTBOX ( "LOAD", "ctrl","","lbx2", ADDSTRING("lbx",stvwData));
CHECKLIST ( "LOAD", "ctrl","","chk2", ADDSTRING("chk",stvwData));
OPTIONLIST ( "LOAD", "ctrl","","opt2", ADDSTRING("opt",stvwData));
TREEVIEW ( "LOAD", "ctrl","","tvw2", ADDSTRING("tvw",stvwData));
END SUB

'retrieve content of selected item in 2nd control:
SUB tvwSel1()
DIM lRet As Long;
DIM stvwText As Str;
DIM stvwText1 As Str;
DIM stvwData As Str;
DIM stvwData1 As Str;
lRet = TREEVIEW( "GETSELECTEDINDEX", "ctrl","","tvw1" );
TREEVIEW( "SETSELECTEDINDEX", "ctrl","","tvw", lRet);
stvwText = TREEVIEW( "GETTEXT", "ctrl","","tvw", lRet);
stvwText1 = TREEVIEW( "GETTEXT", "ctrl","","tvw1", lRet);
stvwData = TREEVIEW( "GETUSERDATA", "ctrl","","tvw", lRet);
stvwData1 = TREEVIEW( "GETUSERDATA", "ctrl","","tvw1", lRet);

'put its contents in a frame:
lRet = TREEVIEW( "COUNT", "ctrl","","tvw1");
SET ( "tvwText" , stvwText);
SET("tvwText1", stvwText1);
SET("tvwData", stvwData);
SET("tvwData1" , stvwData1);
SET("tvwCount" , TREEVIEW( "COUNT", "ctrl","","tvw") );
SET("tvwCount1" , TREEVIEW( "COUNT", "ctrl","","tvw1"));
SENDLIST ("BLOC");
```

```
END SUB

'select 2nd item:
SUB setSeltvw()
DIM lRet As Long;
TREEVIEW( "SETSELECTEDINDEX", "ctrl","","tvw", 2);
END SUB
```

TREND Examples

Applies To

The following example exercises a few modes of the TREND command.

It assumes that there is a window called BigTrend with a Trend Viewer identified as TRN1.

```
SUB Checkmode () ' Check current mode for trend
DIM res AS INTEGER;
res = TREND("GETTYPE", "BigTrend", "", "trn1", 1);
IF (res == 2) THEN
trend.type =REAL ();
ELSE
trend.type = HIST ();
END IF
END SUB

SUB REAL () ' Change trend to real time
TREND("SETTYPE", "BigTrend", "", "trn1", 0, 1);
RETURN ("RealTime");
END SUB

SUB HIST () ' Change trend to historic
TREND("SETTYPE", "BigTrend", "", "trn1", 0, 2);
RETURN ("Historic");
END SUB

SUB PERIOD2 () ' Double the trend period
DIM TrendPeriod AS DOUBLE;
TrendPeriod = TREND("GETPERIOD", "BigTrend", "", "trn1");
TREND("SETPERIOD", "BigTrend", "trn1", "", (TrendPeriod*2));
END SUB

SUB PERIOD1 () ' Half the trend period
DIM TrendPeriod AS DOUBLE;
TrendPeriod = TREND(7, "BigTrend", "", "trn1");
TREND("SETPERIOD", "BigTrend", "", "trn1", (TrendPeriod/2));
END SUB

SUB SCROLLPLUS100 () 'Scroll trend forward 100%
TREND ("SCROLLPERCENT", "BigTrend", "", "trn1", 100);
END SUB

SUB SCROLLMINUS100 () 'Scroll trend back 100%
TREND ("SCROLLPERCENT", "BigTrend", "", "trn1", -100);
END SUB

SUB SCROLLPLUS10 () 'Scroll trend forward 10%
TREND ("SCROLLPERCENT", "BigTrend", "", "trn1", 10);
END SUB

SUB SCROLLMINUS10 () 'Scroll trend back 100%
TREND ("SCROLLPERCENT", "BigTrend", "", "trn1", -10);
END SUB
```

VARIABLE Examples

Applies To

Example 1

This example exercises several modes of the VARIABLE command.

```
'Mode STATUS or Mode 1 (syntax 1)

SUB variablestatus()
PRINT("--STATUS @STATE01--");
PRINT("EXIST = ",VARIABLE("STATUS","@STATE01",1));
PRINT("VALID = ",VARIABLE("STATUS","@STATE01",2));
PRINT("MASK = ",VARIABLE("STATUS","@STATE01",3));
PRINT("ENABLE = ",VARIABLE("STATUS","@STATE01",4));
PRINT("COM = ",VARIABLE("STATUS","@STATE01",5));
PRINT("HDATE = ",VARIABLE("STATUS","@STATE01",6));
END SUB

'Mode MASK or Mode 2 (syntax 2)

SUB variablemask()
PRINT("--MASK @STATE01--");
PRINT("USERPROG1 = ",VARIABLE ("MASK", "@STATE01", 1));
PRINT("USERPROG2 = ",VARIABLE ("MASK", "@STATE01", 2));
PRINT("USERPROG3 = ",VARIABLE ("MASK", "@STATE01", 4);
PRINT("USERPROG4 = ",VARIABLE ("MASK", "@STATE01", 8));
PRINT("OPERATOR = ",VARIABLE ("MASK", "@STATE01", 16));
END SUB

'Mode UNMASK or Mode 3 (syntax 2)

SUB variableunmask()
PRINT("--UNMASK @STATE01--");
PRINT("USERPROG1 = ",VARIABLE ("UNMASK", "@STATE01", 1));
PRINT("USERPROG2 = ",VARIABLE ("UNMASK", "@STATE01", 2));
PRINT("USERPROG3 = ",VARIABLE ("UNMASK", "@STATE01", 4));
PRINT("USERPROG4 = ",VARIABLE ("UNMASK", "@STATE01", 8));
PRINT("OPERATOR = ",VARIABLE ("UNMASK", "@STATE01", 16));
END SUB

'Mode ENABLE or Mode 4 (syntax 3)

SUB variableenable()
PRINT("--ENABLE @STATE01--");
PRINT("ENABLE = ",VARIABLE ("ENABLE","STATE01"));
END SUB

'Mode DISABLE or Mode 5 (syntax 3)

SUB variabledisable()
PRINT("--DISABLE @STATE01--");
PRINT("DISABLE = ",VARIABLE ("DISABLE","STATE01"));
END SUB

'Mode LONGLABEL or Mode 6 (syntax 4)

SUB variablelonglabel()
PRINT("--LONGLABEL @STATE01--");
PRINT("LONGLABEL = ",VARIABLE ("LONGLABEL","STATE01"));
END SUB

'Mode ASSOCLABEL or Mode 7 (syntax 4)

SUB variableassoclabel()
```

```
PRINT("--ASSOCLABEL @STATE01--");
PRINT("ASSOCLABEL = ",VARIABLE ("ASSOCLABEL","STATE01"));
END SUB

'Mode SIMU or Mode 8 (syntax 5)

SUB variablesimuon()
PRINT("--SIMU @STATE02--");
PRINT("SIMU ON= ",VARIABLE ("SIMU","STATE02",1));
END SUB

SUB variablesimuoff()
PRINT("--SIMU @STATE02--");
PRINT("SIMU OFF= ",VARIABLE ("SIMU","STATE02",0));
END SUB

'Mode DOMAIN or Mode 9 (syntax 4)

SUB variabledomain()
PRINT("--DOMAIN @STATE01--");
PRINT("DOMAIN = ",VARIABLE ("DOMAIN","STATE01"));
END SUB

'Mode Nature or Mode 10 (syntax 4)

SUB variableNature()
PRINT("--Nature @STATE01--");
PRINT("Nature = ",VARIABLE ("Nature","STATE01"));
END SUB

'Mode UNIT or Mode 11 (syntax 4)

SUB variableunit()
PRINT("--UNIT @REGISTER01--");
PRINT("UNIT = ",VARIABLE ("UNIT","REGISTER01"));
END SUB

'Mode NUMBER or Mode 12 (syntax 6)

SUB variablenumber()
PRINT("--NUMBER @REGISTER01--");
PRINT("NUMBER = ",VARIABLE ("NUMBER","REGISTER01"));
END SUB

'Mode THRESHOLD_GETTYPE or Mode 13 (syntax 3)

SUB variablethresholdgettype()
PRINT("--THRESHOLD_GETTYPE @REGISTER01--");
PRINT("THRESHOLD_GETTYPE = ",VARIABLE ("THRESHOLD_GETTYPE","REGISTER01"));
END SUB

'Mode THRESHOLD_GETVALUE or Mode 14 (syntax 7)

SUB variablethresholdgetvalue()
PRINT("--THRESHOLD_GETVALUE @REGISTER01--");
PRINT("THRESHOLD_GETVALUE = ",VARIABLE ("THRESHOLD_GETVALUE","REGISTER01", 1));
END SUB

'Mode THRESHOLD_SETVALUE or Mode 15 (syntax 8)

SUB variablethresholdsetvalue()
DIM intValue as integer;
intValue= 8;
PRINT("--THRESHOLD SETVALUE @REGISTER01--");
PRINT("THRESHOLD_SETVALUE = ",VARIABLE ("THRESHOLD_SETVALUE","REGISTER01",1,intValue,1));
```

```
END SUB

'Mode BATT or Mode 22 (syntax 9)

SUB variablebatt()
PRINT("--BATT @REGISTER01--");
PRINT("BATT = ",VARIABLE ("BATT","REGISTER01",2));
END SUB

'Mode TATT or Mode 23 (syntax 10)

SUB variabletatt()
PRINT("--TATT @REGISTER01--");
PRINT("TATT = ",VARIABLE ("TATT","REGISTER01",3));
END SUB

'Mode WRITE or Mode 24 (syntax 11)

SUB variablewrite()
PRINT("--WRITE--");
'PRINT("WRITE = ",VARIABLE ("WRITE",0)); 'WRITE OFF
PRINT("WRITE = ",VARIABLE ("WRITE",1)); 'WRITE ON
END SUB

'Mode LOCKTONODE or Mode 25 (syntax 12)

SUB variablelocktonode()
PRINT("--LOCKTONODE--");

'Lock messages to station 2
PRINT("LOCKTONODE = ",VARIABLE ("LOCKTONODE",2));
END SUB

'Mode UNLOCKTONODE or Mode 26 (syntax 12)

SUB variableunlocktonode()
PRINT("--UNLOCKTONODE--");

'Unlock messages to station no. 2
PRINT("UNLOCKTONODE = ",VARIABLE ("UNLOCKTONODE",2));
END SUB

'Mode FLOWPARAMTONODE or Mode 27 (syntax 13)

'Control all variables for remote client station no.2
' with a stopping threshold of 100 and activation threshold of 150
SUB variableFlowParamToNode()
VARIABLE("FLOWPARAMTONODE",2,100,150,15);
END SUB

'Control of register and text variables for client station no.5,
' with a stopping threshold of 50 and activation threshold of 60
SUB variableFlowParamToNode2()
VARIABLE("FLOWPARAMTONODE",5,50,60,12);
END SUB

'Mode IMPORTBUFFER or Mode 28 (syntax 14)

SUB variableimportbuffer()
DIM lngbuffer1 as long;
lngbuffer1 = FILETOBUF("var.txt");
PRINT("--IMPORTBUFFER--");
PRINT("IMPORTBUFFER = ",VARIABLE ("IMPORTBUFFER",lngbuffer1));
FREE BUFFER(lngbuffer1);
END SUB
```

```
'Mode IMPORTFILE or Mode 29 (syntax 15)

SUB variableimportfile()
PRINT("--IMPORTFILE--");
PRINT("IMPORTFILE = ",VARIABLE ("IMPORTFILE","var.txt"));
END SUB

'Mode STOPWATCHLIST or Mode 31 (syntax 17)

SUB variablestartwatchlist()
DIM lngbuffer1 as long;
DIM lngbuffer2 as long;
DIM intResult as integer;
lngbuffer1 = FILETOBUF("var2.txt");
PRINT("--STARTWATCHLIST--");
lngbuffer2 = VARIABLE("STARTWATCHLIST","",lngbuffer1,"STATE02",1));
intResult = VARIABLE("STOPWATCHLIST",lngbuffer2);
FREE_BUFFER(lngbuffer1);
END SUB

'Mode GET_LONG_IN_DB or Mode 32 (syntax 18)

SUB variablegetlongindb()
PRINT("--GET_LONG_IN_DB--");
PRINT("GET_LONG_IN_DB = ",VARIABLE ("GET_LONG_IN_DB","REGISTER01"));
END SUB

'Mode GET_DOUBLE_IN_DB or Mode 33 (syntax 19)

SUB variablegetdoubleindb()
PRINT("--GET_DOUBLE_IN_DB--");
PRINT("GET_DOUBLE_IN_DB = ",VARIABLE ("GET_DOUBLE_IN_DB","REGISTER01"));
END SUB

'Mode GET_PHYSICAL_MIN or Mode 34 (syntax 19)

SUB variablegetphysicalmin()
PRINT("--GET_PHYSICAL_MIN--");
PRINT("GET_PHYSICAL_MIN = ",VARIABLE ("GET_PHYSICAL_MIN","REGISTER01"));
END SUB

'Mode GET_PHYSICAL_MAX or Mode 35 (syntax 19)

SUB variablegetphysicalmax()
PRINT("--GET_PHYSICAL_MAX--");
PRINT("GET_PHYSICAL_MAX = ",VARIABLE ("GET_PHYSICAL_MAX","REGISTER01"));
END SUB

'Mode GET_CONTROL_MIN or Mode 36 (syntax 19)

SUB variablegetcontrolmin()
PRINT("--GET_CONTROL_MIN--");
PRINT("GET_CONTROL_MIN = ",VARIABLE ("GET_CONTROL_MIN","REGISTER01"));
END SUB

'Mode GET_CONTROL_MAX or Mode 37 (syntax 19)

SUB variablegetcontrolmax()
PRINT("--GET_CONTROL_MAX--");
PRINT("GET_CONTROL_MAX = ",VARIABLE ("GET_CONTROL_MAX","REGISTER01"));
END SUB

'Mode SETBATT or Mode 38 (syntax 20)
```

```
SUB variablesetbatt()
PRINT("--SETBATT @REGISTER01--");
PRINT("SETBATT = ",VARIABLE ("SETBATT","REGISTER01",2));
END SUB

'Mode SETTATT or Mode 39 (syntax 21)

SUB variablesettatt()
PRINT("--SETTATT @REGISTER01--");
PRINT("SETTATT = ",VARIABLE ("SETTATT","REGISTER01",3));
END SUB

'Mode SAVE or Mode 40 (syntax 22)

SUB variablesave()
PRINT("--SAVE *--");
PRINT("SAVE = ",VARIABLE ("SAVE"));
END SUB

'Mode GET_ALARM_PRIORITY or Mode 42 (syntax 3)

SUB variablegetalarmpriority()
PRINT("--GET_ALARM_PRIORITY @ALARM01--");
PRINT("GET ALARM PRIORITY = ",VARIABLE ("GET ALARM PRIORITY","ALARM01"));
END SUB

'Mode GET_COMMAND_LEVEL or Mode 43 (syntax 3)

SUB variablegetcommandlevel()
PRINT("--GET_COMMAND_LEVEL @ALARM01--");
PRINT("GET_COMMAND_LEVEL = ",VARIABLE ("GET_COMMAND_LEVEL","ALARM01"));
END SUB

'Mode GET_CONTROL_LEVEL or Mode 44 (syntax 3)

SUB variablegetcontrollevel()
PRINT("--GET_CONTROL_LEVEL @REGISTER01--");
PRINT("GET_CONTROL_LEVEL = ",VARIABLE ("GET_CONTROL_LEVEL","REGISTER01"));
END SUB

'Mode GET_TEXT_LEVEL or Mode 45 (syntax 3)

SUB variablegettextlevel()
PRINT("--GET_TEXT_LEVEL @TEXT01--");
PRINT("GET_TEXT_LEVEL = ",VARIABLE ("GET_TEXT_LEVEL","TEXT01"));
END SUB

'Mode GET_TYPE or Mode 46 (syntax 3)

SUB variablegettype()
PRINT("--GET_TYPE @TEXT01--");
PRINT("GET_TYPE = ",VARIABLE ("GET_TYPE","TEXT01"));
END SUB

'Mode GET_DEADBAND_TYPE or Mode 53 (syntax 3)

SUB variablegetdeadbandtype()
PRINT("--GET_DEADBAND_TYPE @REGISTER01--");
PRINT("GET_DEADBAND_TYPE = ",VARIABLE ("GET_DEADBAND_TYPE","REGISTER01"));
END SUB

'Mode GET_DEADBAND_VALUE or Mode 54 (syntax 23)

SUB variablegetdeadbandvalue()
PRINT("--GET_DEADBAND_VALUE @REGISTER01--");
```

```
PRINT("GET_DEADBAND_VALUE = ",VARIABLE ("GET_DEADBAND_VALUE","REGISTER01"));
END SUB

'Mode SET_DEADBAND or Mode 55 (syntax 24)

SUB variablesetdeadband()
PRINT("--SET_DEADBAND @REGISTER01--");
PRINT("SET_DEADBAND = ",VARIABLE ("SET_DEADBAND","REGISTER01", 3, 2));
END SUB
```

Example 2

This example shows several uses of the VARIABLE instruction.

```
' Subroutine to return variable status
SUB status()
DIM res AS INTEGER;
res = VARIABLE("STATUS", setvar.name, "VALID");
IF (res == 0) THEN
setvar.valid = 0;
ELSE
setvar.valid = 1;
END IF
res = VARIABLE("STATUS", setvar.name, "ENABLE");
IF (res == 0) THEN
setvar.enable = 0;
ELSE
setvar.enable = 1;
END IF
END SUB

' Toggle inhibit/enable
SUB inhibit()
DIM res AS INTEGER;
IF (setvar.enable == 0) THEN
res = VARIABLE("DISABLE", setvar.name);
ELSE
res = VARIABLE("ENABLE", setvar.name);
END IF
END SUB

SUB threshold()
PRINT(VARIABLE("THRESHOLD_GETVALUE", "txremp", 0));
VARIABLE("THRESHOLD_SETVALUE", "txremp", 0, 52, 1);
PRINT(VARIABLE("THRESHOLD_GETTYPE", "txremp"));
END SUB
```

Example 3

This example shows the use of the IMPORTFILE mode.

```
'*********************************************************************
' ImportVarObject
' ---------------
' Import the objects in a file passed as an argument
' _____________________________________________________________________

SUB ImportVarObject()
 DIM a_strImpFileName AS STR; ' Name of the import file
 DIM l_RetCode AS INTEGER; ' Return code
 DIM l_strPath AS STR; ' Project path
 DIM l_strFullImpFileName AS STR; ' Import file name and path
 DIM l_strFullReportFileName AS STR; ' Report file name and path
 DIM l_strFullErrorFileName AS STR; ' Error file name and path

 a_strImpFileName = GETARG("ARG1");
 l_strPath = GETPROJECTDIR();
```

```
 l_strFullImpFileName = ADDSTRING(l_strPath, "\\TP\\");
 l_strFullImpFileName = ADDSTRING(l_strFullImpFileName, a_strImpFileName);
 l_strFullReportFileName = ADDSTRING(l_strPath, "\\TP\\");
 l_strFullReportFileName = ADDSTRING(l_strFullReportFileName,"FileReport.log");
 l_strFullErrorFileName = ADDSTRING(l_strPath, "\\TP\\");
 l_strFullErrorFileName = ADDSTRING(l_strFullErrorFileName, "FileReport.err");

 l_RetCode = VARIABLE("IMPORTFILE", l_strFullImpFileName, 0, "SYS.IMPORTRESULT",
l_strFullReportFileName, l_strFullErrorFileName);
 Print("Variable import ", l_strFullImpFileName, " Ret code ", l_RetCode);
END SUB
```

Example 4

This example imports the object in the file passed as a parameter from the command.

```
' ImportVarObject
' ---------------
'Import variable
SUB ImportVarObject()
' [in] name of file to import
DIM a_strImpFileName AS STR;
' result code
DIM l_RetCode AS INTEGER;
' project path
DIM l_strPath AS STR;
 ' path & name of file to import
DIM l_strFullImpFileName AS STR;
 ' path & name of file containing the import report
DIM l_strFullReportFileName AS STR;
 ' path & name of file of formatting errors if any
DIM l_strFullErrorFileName AS STR;
 a_strImpFileName = GETARG("ARG1");
l_strPath = GETPROJECTDIR();
l_strFullImpFileName = ADDSTRING( l_strPath, "\\TP\\");
l_strFullImpFileName = ADDSTRING( l_strFullImpFileName, a_strImpFileName);
l_strFullReportFileName = ADDSTRING( l_strPath, "\\TP\\");
l_strFullReportFileName = ADDSTRING( l_strFullReportFileName,"FileReport.log");
l_strFullErrorFileName = ADDSTRING( l_strPath, "\\TP\\");
l_strFullErrorFileName = ADDSTRING( l_strFullErrorFileName, "FileReport.err");
 l_RetCode = VARIABLE( "IMPORTFILE", l_strFullImpFileName, 0, "SYS.IMPORTRESULT",
l_strFullReportFileName, l_strFullErrorFileName);
 PRINT( "Variable import ", l_strFullImpFileName, " RetCode = ", l_RetCode);
END SUB
```

WEBVUE Examples

Applies To

The following example lists the users on all Web client stations, from the Supervisor station:

```
SUB MAIN()
CheckList();
END SUB

SUB CheckList()
  DIM res AS INTEGER;
  DIM ClientList AS STR;
  DIM hbuffer As LONG;
  hbuffer = ALLOC_BUFFER(50);  'allocate a buffer of 50KB
  res = WEBVUE("LIST", hbuffer);  'get a list of user-station pairs
  ClientList = CGET_BUFFER(hbuffer,0,255);
  PRINT("Number of web clients: ", res);
  PRINT("List of web clients: ", ClientList);
  FREE_BUFFER(hbuffer);
END SUB
```

From a Web client station, obtain the user name for that station:

```
SUB GetWebUser()
  DIM i As Integer;
  DIM lUser As Long;
  DIM sUser As Str;
  lUser = ALLOC_BUFFER (128);
  i = GETARG ("WEB");
  WEBVUE ("USERNAME", i, lUser);
  sUser = CGET_BUFFER(lUser,0,255);
  PRINT ("User: ", sUser);
  FREE_BUFFER (lUser);
END SUB
```

This Sub is run when a context mimic is opened in a WebVue client. It provides the name of the context variable:

```
Sub Load()
  Dim lContext As LONG;
  Dim iSession As INTEGER;
  Dim sContext As STR;
  Dim Ret as INTEGER;
  PRINT ("SetVarWithContext WebVue and Supervisor");
  iSession = GETARG ("WEB");
  WINDOW("CLOSE","TREND","");
  If (iSession != 0) Then
    WINDOW("CLOSE","Trend","");
    lContext = ALLOC_BUFFER ( 1000 );
    WEBVUE ("CONTEXT", iSession, lContext);
    sContext = CGET_BUFFER (lContext, 0, SEQ_BUFFER("LEN",lContext));
    PRINT (sContext);
    WINDOW("OPEN","Trend","");
  End If
End Sub
```

WINDOW Example

Applies To

This example exercises several modes of the WINDOW command.

```
Const cstrealtime = 0;
Const cstref1 = 1;
Const cstref2 = 2;
Const csttest = 3;
SUB main()
windowcurrentname();
windowcurrentbranch();
END SUB

'Mode CLOSE or Mode 0 (syntax 1)

SUB windowclose()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("CLOSE","window01","branch01");
END SUB

SUB windowcloserealtime()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("CLOSE","window01","branch01", cstrealtime);
END SUB

SUB windowclosetest()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("CLOSE","window01","branch01", csttest);
END SUB

SUB windowcloseref1()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("CLOSE","window01","branch01", cstref1);
END SUB

SUB windowcloseref2()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("CLOSE","window01","branch01", cstref2);
END SUB

'Mode OPEN or Mode 1

SUB windowopen()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("OPEN","window01","branch01");
END SUB

SUB windowopenparent()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("OPEN","window01","branch01","MENU","");
END SUB

SUB windowopenparentref()
'Declare return code
DIM intReturn as integer;
intReturn = WINDOW("OPEN","window01","branch01","MENU","",150,150);
END SUB
```

```
'Mode IS OPEN or Mode 2 (syntax 1)

SUB windowisopen()
PRINT("IS_OPEN : ",WINDOW("IS_OPEN","window01","branch01"));
END SUB

'Mode SHOW or Mode 3 (syntax 1)

SUB windowshow()
PRINT("SHOW : ",WINDOW("SHOW","window01","branch01"));
END SUB

'Mode HIDE or Mode 4 (syntax 1)

SUB windowhide()
PRINT("HIDE : ",WINDOW("HIDE","window01","branch01"));
END SUB

'Mode CHANGE or Mode 5 (syntax 10)
SUB windowchange()
PRINT("CHANGE : ",WINDOW("CHANGE","window01","branch01",50,50,200,200));
END SUB

'Mode CLOSEUNDER or Mode 6 (syntax 1)

SUB windowcloseunder()
PRINT("CLOSEUNDER : ",WINDOW("CLOSEUNDER","window01","branch01"));
END SUB

'Mode PRELOAD or Mode 7 (syntax 1)

SUB windowpreload()
PRINT("PRELOAD : ",WINDOW("PRELOAD","window01","branch01"));
END SUB

'Mode CLOSEALL or Mode 8 (syntax 6)

SUB windowcloseall()
PRINT("CLOSEALL : ",WINDOW("CLOSEALL"));
END SUB

'Mode MAIN or Mode 9 (syntax 3)
SUB windowmain()
PRINT("MAIN : ",WINDOW("MAIN",50,50,200,200));
END SUB

'Mode CURRENTNAME or Mode 10 (syntax 7)

SUB windowcurrentname()
PRINT("CURRENTNAME : ",WINDOW("CURRENTNAME"));
END SUB

'Mode CURRENTbranch or Mode 11 (syntax 8)

SUB windowcurrentbranch()
PRINT("CURRENTbranch : ",WINDOW("CURRENTbranch"));
END SUB

'Mode CAPTION or Mode 12 (syntax 8)

SUB windowcaption()
DIM strTitle as Str;
strTitle = "#D                                    ";
'strTitle = "#D";
```

```
PRINT("CAPTION : ",WINDOW("CAPTION","window01","branch01",strTitle));
END SUB

'Mode REFSET or Mode 13 (syntax 9)

SUB windowrefset()
PRINT("REFSET : ",WINDOW("REFSET","window01","branch01",cstrealtime,csttest));
END SUB

'Mode CURRENTREF or Mode 14 (syntax 7)

SUB windowcurrentref()
PRINT("CURRENTREF : ",WINDOW("CURRENTREF"));
'Mode_REALTIME 0
'Mode_REF1 1
'Mode_REF2 2
'Mode_TEST 3
END SUB

'Mode POPUPCLOSE or Mode 15 (syntax 6)

SUB windowpopupclose()
PRINT("POPUPCLOSE : ",WINDOW("POPUPCLOSE"));
END SUB

'Mode ACCESSLEVEL or Mode 16 (syntax 1)

SUB windowaccesslevel()
PRINT("ACCESSLEVEL : ",WINDOW("ACCESSLEVEL","window01","branch01"));
END SUB

'Mode OPENNEW or Mode 17 (syntax 1)

SUB windowopennew()
PRINT("OPENNEW : ",WINDOW("OPENNEW","window01","branch01"));
END SUB

'Mode PRINT or Mode 18 (syntax 1)

SUB windowPRINT()
'The window must first be open
PRINT("PRINT : ",WINDOW("PRINT","window01","branch01"));
END SUB

'Mode ZOOM or Mode 19 (syntax 11)

SUB windowzoom()
'The window must first be open
PRINT("ZOOM : ",WINDOW("ZOOM","window01","branch01",50,50,70));
END SUB

'Mode GETREGION or Mode 20 (syntax 1)

SUB windowgetregion()
PRINT("GETREGION : ",WINDOW("GETREGION","window01","branch01"));
END SUB

'Mode GETSUBWINDOW or Mode 21 (syntax 12)

SUB windowgetsubwindow()
'PRINT("GETSUBWINDOW : ",WINDOW("GETSUBWINDOW","#M1"));
PRINT("GETSUBWINDOW : ",WINDOW("GETSUBWINDOW","#U"));
END SUB

'Mode GETSUBbranch or Mode 22 (syntax 12)
```

```
SUB windowgetsubbranch()
'PRINT("GETSUBbranch : ",WINDOW("GETSUBbranch","#M1"));
PRINT("GETSUBbranch : ",WINDOW("GETSUBbranch","#U"));
END SUB

'Mode SETPREVIOUS or Mode 23 (syntax 6)

SUB windowsetprevious()
PRINT("SETPREVIOUS : ",WINDOW("SETPREVIOUS"));
END SUB

'Mode LAYER or Mode 24 (syntax 13)
SUB windowlayer()
PRINT("LAYER : ",WINDOW("LAYER","window01","branch01",cstrealtime,2,2));
END SUB

'Mode HARDCOPY or Mode 25 (syntax 14)
SUB windowhardcopy()
'The window must first be open
PRINT("HARDCOPY : ",WINDOW("HARDCOPY","window01","branch01",cstrealtime,1));
END SUB

'Mode SAVE or Mode 26 (syntax 15)
SUB windowsave()
PRINT("SAVE : ",WINDOW("SAVE","window01","branch01","ASCII32"));
'save the mimic in ASCII
END SUB
```

XMLPATH Examples

Applies To

Example 1 - Extracting data from an xml file

Contents of the file books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
  <bookstore>
    <book category="Cooking">
      <details lang="en" title="Everyday Italian" author="Giada De Laurentiis"
year="2005" price="30.99" />
    </book>
    <book category="Children">
      <details lang="en" title="Harry Potter" author="J K. Rowling" year="2005"
price="5.99" />
    </book>
    <book category="Web">
      <details lang="en" title="XQuery Kick Start" author="James McGovern" year="2003"
price="49.99" />
    </book>
  </bookstore>
```

Sample program to extract data

```
Sub Main()
  Dim cXPath As Str, cReturn As Str;
  Dim iReturn As Integer, i As Integer;
  Dim sReturn As Single;

  iReturn = XMLPATH ( "LOADFILE", "CTX1", "books.xml" );
  Print ("Return from LOADFILE = ", iReturn);

  For (i=1; i<4; i++)
    cXPath = FORMAT ("bookstore/book[%d].category", i);
    cReturn = XMLPATH ( "GETSTR", "CTX1", cXPath);
    Print(cXPath, " = ", cReturn);
    cXPath = FORMAT ("bookstore/book[%d]/details.author", i);
    cReturn = XMLPATH ( "GETSTR", "CTX1", cXPath);
    Print(cXPath, " = ", cReturn);
    cXPath = FORMAT ("bookstore/book[%d]/details.title", i);
    cReturn = XMLPATH ( "GETSTR", "CTX1", cXPath);
    Print(cXPath, " = ", cReturn);

    cXPath = FORMAT ("bookstore/book[%d]/details.year", i);
    iReturn = XMLPATH ( "GETINT", "CTX1", cXPath);
    Print(cXPath, " = ", iReturn);

    cXPath = FORMAT ("bookstore/book[%d]/details.price", i);
    sReturn = XMLPATH ( "GETSINGLE", "CTX1", cXPath);
    Print(cXPath, " = ", sReturn);
  Next

  iReturn = XMLPATH ( "UNLOAD", "CTX1");
  Print ("Return from UNLOAD = ", iReturn);
End Sub
```

Example 2 - Data returned by mode LIST

```
<list myid="767565456788" > //myid is my own session id if the verb is call from a WebVue
context
<session> //A first active session
```

```
  id="535625424245" //First session id
  creationtime= xy //Session opening date and time. Value of type double that can be
converted using the DATETIME verb.
  username="web" //Session username
  address="192.168.1.123" //Remote web client IP address (or router address)
  computername="" // Remote web client computer name if it can be retrieved
  clienttype=" webvue " //Session type. Value is webvue for WebVue sessions and wst for
Web Services Toolkit sessions.   <wst> // Additional information to identify partners'
products sessions
  key=""
  <wst />
<session />

<session> //A second active session
  id="535625422312"
  creationtime=xxxxxxx
  username="user_dr"
  address="192.168.1.124"
  computername=""
  clienttype="
  webvue " //="webvue" or "wst"
  <wst> //This session is used by the Dream Report Web Services driver
  key=" DreamReport"
  <wst />
<session />
<list />
```

Script to analyze the returned data

```
Sub WebList()
  Dim lHandle As Long;
  Dim i As Integer;
  Dim sPath As Str;

  WEBVUE ( "LIST" );
  lHandle = XMLPATH ( "GET", "webvue/list", "list.myid");
  If (lHandle != 0 ) Then
    PRINT ("myid ",CGET_BUFFER ( lHandle, 0, 255 ));
  End If

  i=0;

  While (1)
    sPath = FORMAT ("list.session[%d].id", i+1);
    lHandle = XMLPATH ( "GET", "webvue/list", sPath );

    If (lHandle == 0 ) Then
      Break;
    End If

    PRINT ("id ",CGET_BUFFER ( lHandle, 0, 255 ));
    sPath = FORMAT ("list.session[%d].creationtime", i+1);
    PRINT( "creationtime ", XMLPATH ( "GETDOUBLE", "webvue/list", sPath ));
    sPath = FORMAT ("list.session[%d].username", i+1);
    lHandle = XMLPATH ( "GET", "webvue/list", sPath );
    PRINT ("username ",CGET_BUFFER ( lHandle, 0, 255 ));
    sPath = FORMAT ("list.session[%d].computername", i+1);
    lHandle = XMLPATH ( "GET", "webvue/list", sPath );
    PRINT ("computername ",CGET_BUFFER ( lHandle, 0, 255 ));
    sPath = FORMAT ("list.session[%d].address", i+1);
    lHandle = XMLPATH ( "GET", "webvue/list", sPath );
    PRINT ("address ",CGET_BUFFER ( lHandle, 0, 255 ));
    sPath = FORMAT ("list.session[%d].clienttype", i+1);
    lHandle = XMLPATH ( "GET", "webvue/list", sPath );
    PRINT ("clienttype ",CGET_BUFFER ( lHandle, 0, 255 ));
    i=i+1;
```

```
  Wend
End
```

Example 3 - Using XMLPATH to unpack a namespace

```
Sub main()
End Sub

Sub subscription()
  Dim WinName as Str;
  Dim WinBranch as Str;
  Dim NameSpace as Str;
  Dim iRet as Integer;

  WinName = WINDOW("CURRENTNAME");
  WinBranch = WINDOW("CURRENTBRANCH");
  NameSpace = AddString(WinBranch,"/", WinName, "/alarm1");
  @lineselect.Syno = WinName;
  @lineselect.Branch = WinBranch;
  @lineselect.AlarmSelected = "";
  @lineselect.NameSpace = NameSpace;
  iRet = AlarmDisplay"LINESELECT",WinName,WinBranch,"alarm1","P","","SelectionLine","");
  Print("Subscription with result: ",iRet);
End Sub

Sub SelectionLine()
  Dim WinName as Str;
  Dim WinBranch as Str;
  Dim NameSpace as Str;
  Dim AlarmSelected as Str;
  Dim iBuf as Long;

  WinBranch = WINDOW("CURRENTBRANCH");
  NameSpace = AddString(WinBranch,"/",WinName,"/alarm1");
  @lineselect.NameSpace = NameSpace;
  iBuf = XMLPATH("GET",NameSpace,"lineselect/variable");
  AlarmSelected = CGET_BUFFER(iBuf, 0, 255);
  @lineselect.AlarmSelected = AlarmSelected ;
  XMLPATH("UNLOAD",NameSpace);
End Sub
```

Reserved Words

See Also

Words marked (*) are undocumented. Do not use.

A

ACOS
ADDCHAINE
ADDSTRING
ALARM
ALARMDISPLAY
ALIAS
ALLOC_BUFFER
ANIMATION
APPLICATION
ARRET (*)
AS
ASC
ASCIIFIELD
ASIN
ASSOCLABEL
ASSOCIATEDACTIONS
ATAN

B

BACNET
BASIC
BEEP
BIN
BRANCH (*)
BREAK
BUFTOEXCEL
BUFTOFILE
BYVAL

C

CALL (*)
CAPTION
CGET_BUFFER
CHAR (*)
CHART
CHECKLIST

E

ELSE
ELSEC (*)
EMAIL
END
ENDSUB
ERROR
EVENT
EXCELTOBUF
EXECUTE (*)
EXIT
EXP
EXPORT
EXPRESSION

F

FCLOSE
FEOF
FGETC
FGETS
FILETOBUF
FOPEN
FOR
FORMAT
FORMULA
FPUTC
FPUTS
FREAD
FREE_BUFFER
FSEEK
FTP
FSTAT
FTP
FUNCTION
FWRITE

G

L

LAN
LANGUAGE
LCASE
LEFT
LEN
LGET_BUFFER
LISTBOX
LNS
LOG
LOGDISPLAY
LOGICAL
LOGICAL64
LONG
LONWORKS
LPRINT
LTRIM
LVAL

M

M104
M6185
MAJANA (*)
MAPDISPLAY
MDMP3
MULTIMEDIA
MID

N

NEXT

O

OCT
OPC
OPTIONLIST

P

S

SELECTOR
SEND (*)
SENDLIST
SEQ_BUFFER
SET
SGET_BUFFER
SHELL (*)
SIM (*)
SIN
SINGLE
SLEEP (*)
SMS
SPACE
SNMP
SQRT
START (*)
STATION_FILTER
STOP
STR
STRING
SUB
SVAL
SVALA
SVBATCH
SVBRANCH
SVKEY
SVLOG
SVSQL
SVTREND
SYSTEM

T

TAN
TEMPORARY_DB
TEXTVAR
THEN

Reference topics

Access Rights Weighting

Applies To

Administration rights

For administration rights the binary weights represent:

Weight	Meaning
1	Modify date and time.
2	Modify password.
4	Create and modify user accounts.
8	Delete user accounts and/or their associations.
16	Associate stations to users profiles.
32	Create profiles.
64	Delete profiles.
128	Automatic logoff enabled.
256	Password lifetime enabled.

For example: In a project, if the Default Profile has all of these options checked except the last two (i.e. binary 1111111), the returned integer is 127.

System access rights

For system access rights the binary weights represent:

Weight	Meaning
1	Access rights
2	Command and acknowledge
4	Window access
8	Exit (Quit)
16	Help
32	Install
64	Preferences
128	Administration
256	Desktop
512	Recipe
1024	Timetable

Recipe access rights

For recipe access rights the binary weights represent:

Weight	Meaning
1	Manager
2	Save
4	Creation
8	List modification
16	Real-time
32	Send
64	Delete

Window printing rights

Returns 1 if a user has window printing rights, else 0.

For more information see the User Rights book in the configuration Help.

Associated Action File Formats

See Also Applies To

The process of configuring Associated Actions for alarms is described in the topic 'Associating an Action with an Alarm' in the main Help.

The tables in this book specify:

- The format in which Associated Actions are stored in the application project.
- The formats for changing them.

Files

The files for storing and importing Associated Actions are held in the project folder as follows:

File name	Sub-folder	Contents
ACTION.DAT	C	Stored action records: one line per action.
*XXXX.*DAT	TP	Changes: one line per addition or deletion.

where *XXXX* is the name used in the ASSOCIATEDACTIONS instruction.

This table shows the storage format for an Associated Action for an alarm. Show table

No.	Description	Type	Size (bytes)	Value(s)
1	Record type (optional)	Character	6	ACTION
2	Type of action (0 = action associated with alarms)	Numeric	2	0
3	Button for selection (LB = left, RB = right)	Character	2	LB or RB
4	Variable name	Character	40	
5	Program/window flag (P= program, W = window)	Character	1	P/W
6	Name of program or window	Character	12	
7	Branch	Character	30	
8	Function	Character	30	
9	Arguments (in quotes)	Character	2,047	
10*	List of producer stations	Character	14	
11*	List of consumer stations	Character	14	
12*	Origin of the object (0 if local, >0 if remote)	Numeric	2	
13*	Version number of the object	Numeric	2	
14*	Comment on the object	Character	80	

* New fields.

The record type is optional (i.e. the user can write a record with or without it).

Adding an Associated Action

The format for adding an Associated Action entry for an alarm is as follows. It starts with the keyword "AAA,0,".

The record type is optional (i.e. the user can write a record with or without it).

Show table

No.	Description	Type	Size (bytes)	Value(s)
1	Record type	Character	6	AAA
2	Type of action (0 = action associated with alarms)	Numeric	2	0
3	Button for selection (LB = left button, RB = right button)	Character	2	LB or RB
4	Variable name	Character	40	
5	Program/window flag (P= program, W = window)	Character	1	P or W
6	Name of program or window	Character	12	
7	Branch	Character	30	
8	Function	Character	30	
9	Arguments (in quotes)	Character	2,047	
10	List of producer stations	Character	14	
11	List of consumer stations	Character	14	
12	Origin of the object (0 if local, >0 if industry standard)	Numeric	2	
13	Version number of the object	Numeric	2	
14	Comment on the object	Character	80	

The example below shows the sequence of lines.

```
AAA,0,PAR,"action 1 label Fra","action 1 label Eng","action 2 label Fra","action 2 label
Eng"
AAA,0,LB,"testAA",P,"x","","","","S1","C1",0,0,"action 1 test"
AAA,0,RB,"testAA",W,"mm","","","","S1","C1",0,0,"action 2 test"
AAA,0,LB,"ALARMAA",P,"X","","","","S1","C1",0,0,"test1"
AAA,0,RB,"ALARMAA",W,"mm","","","","S2","C2",0,0,"test2"
```

Deleting an Associated Action

The format for deleting an existing Associated Action entry for an alarm is as follows. It starts with the keyword "DAAA". Show table

No.	Description	Type	Size (bytes)	Value(s)
1	Record type	Character	6	DAAA
2	Type of action (0 = action associated with alarms)	Numeric	2	0
3	Button for selection (LB = left, RB = right)	Character	2	LB or RB
4	Variable name	Character	40	
5	Program/window flag	Character	1	P or W

	(P= program, W = window)		
6	Name of program or window	Character	12
7	Branch	Character	30
8	Function	Character	30
9	Arguments (in quotes)	Character	2,047
10	List of producer stations	Character	14
11	List of consumer stations	Character	14
12	Origin of the object	Numeric	2
	(0 if local, >0 if industry standard)		
13	Version number of the object	Numeric	2
14	Comment on the object	Character	80

The sequence of lines depends on the desired effect:

- The first line is required if you do not wish to delete all of the entries. (Without it, all of the actions will be lost.)
- From the second line onwards, at least five fields per line are required.

```
DAAA,0,PAR,"action 1 label Fra","action 1 label Eng","action 2 label Fra","action 2 label
Eng"
DAAA,0,LB,"testAA",P,"x","","","","","",0,0,""
DAAA,0,RB,"testAA",W,"mm","","","","","",0,1,""
```

Changing the Sound Played by WEBVUE Mode MULTIMEDIA

See Also

The sound file played is called bip.wav and is embedded in the Java applet AiWebVue.jar located in the following folder.

```
<Installation folder>\bin\WebServerExtensions\WebVue\Classes
```

The jar file is in zip format and can be edited using any good zip editor. The basic procedure is as follows. Before starting take a backup of AiWebVue.jar as a precaution.

1. Create a new sound file and name it bip.wav.
2. Change the filename of AiWebVue.jar to AiWebVue.zip.
3. Edit the zip file with any zip editor and replace the existing bip.wav with the new one you have created.
4. Rename the file back to the original name AiWebVue.jar.

If you have already used WebVue, you will have to flush the java cache in order to force the download of the edited jar file next time WebVue is started. Instructions on how to do this may be found in the main help in the topic Updating the (WebVue) applet.

Communication Object Parameters

See Also Applies To

CIMWAY mode CFG is used to dynamically make configuration changes to equipment communication objects. That is, those communication objects configured from the menu command Configure.Commuication.Equipment. The CIMWAY instruction has the following syntax when used in CFG mode. For information on other modes see the CIMWAY topic.

CIMWAY(*"CFG"*, *ComObj*, *Modif*, *Param*, [, *VarName*]);

Argument	Meaning
ComObj	The communication object: a network, a node or a frame.
Modif	The property of the communication object to be modified. Either PORT_NUMBER, EQT_ADDRESS or MEMORY_ADDRESS.
Param	A parameter field, the syntax of which depends on the value of *Modif* and on the particular communication driver being modified. The general syntax of *Param* is P1#P2#P3...#Pn. Type STR.
VarName	The name of an optional register variable which can be used to check the status of the request. Type STR. 0 = Request is being processed 1 = Request succesfully completed 2 = Request failed

Modif = PORT_NUMBER

Modify the port number used to connect to the communication network. *ComObj* is the name of a network. For example Net1. In general only used for drivers using a serial port. *Param* is the port number in the range is from 1 to 64 (COM1 to COM64)

Modif = EQT_ADDRESS

Modify the equipment address. *ComObj* is the name of a node (device) attached to the network. For example Net1.Node1. *Param* is a string with the general syntax of P1#P2#P3...#Pn. The exact syntax depends on the communication driver being modified. See the following table.

Driver	Param
XBUS-IP-MASTER*	slave_address#ip1#ip2#ip3#ip4#port_number
IP-ISO-S7	cpu_slot#ip1#ip2#ip3#ip4#cpu_rack#ts_word#ts_db#0#reconnect_period#0
IP-SAIA	sbus_address#ip1#ip2#ip3#ip4#port#cpu#ts_address#flag
IP-SRTP*	1#ip1#ip2#ip3#ip4#reconnect_period#error_address#UTC_timestamp#password
S7-IP-Master*	1#ip1#ip2#ip3#ip4#PortFetch#error_address#DB_number#share_connection#reconnect_period

* Without redundancy. Extra fields may be required if there is redundancy. See the driver documentation.

For detailed explanation of each of Param's elements see the documentation for the corresponding driver.
For IP based drivers ip1#ip2#ip3#ip4 is the equipment IP address. For example 192#168#0#99.

Modif = MEMORY_ADDRESS

Modify a frame address. The parameter is normally an integer. The value depends on the equipment type.

Conditions of Variables

See Also Applies To

This topic lists the available conditions for each type of variable.

Bit or Alarm Variables

The expression for bit and alarm variables may contain a number of conditions under which the event may be triggered. The syntax used is as follows:

$State_1 > State_2, State_3 > State_4, ...State_{N-1} > State_N$

where '$State_{N-1} > State_N$' defines a transition from one state to another.

For bits, the states available are:

State	Meaning
0	false
1	true
S	significant (valid)
NS	non-significant (invalid)
ALL	any State

For example: ALL>NS, ALL>1.

For alarms, the available states are:

State	Meaning
S	significant (valid)
NS	non-significant (invalid)
ALL	any state
ACKON	on and acknowledged
ACKOFF	off
NOACKON	on and not acknowledged
NOACKOFF	off and not acknowledged

For example: ACKON>ALL, ACKOFF>ALL.

Text Variables

The expression for a text variable may contain only one condition.

Condition	Meaning
ALL>S	Transition from any state to a significant (Valid) value.
NS > S	Transition from a non-significant (Invalid) value to a significant value.
S > S	All changes of significant value.
S>ALL	Transition from a significant (Valid) value to any other state.
S > NS	Transition from a significant value to non-significant (Invalid).
= *string*	Equal to the given string.
> *string*	The ASCII value of the text variable is greater than that of *string*.
< *string*	The ASCII value of the text variable is less than that of *string*.

Register Variables

The expression for a register variable may contain only one condition.

Condition	Meaning
ALL>S	Transition from any state to a significant (Valid) value.
NS > S	Transition from a non-significant (Invalid) value to a significant value.
S > S	All changes of significant value.
S>ALL	Transition from a significant (Valid) value to any other state.
S > NS	Transition from a significant value to non-significant (Invalid).
= *value*	Equal to "value".
+ *value*	The variable increases by an amount equal or greater than *value*.
- *value*	The variable decreases by an amount equal or greater than *value*.
> *value*	The variable is greater than *value*.
< *value*	The variable is less than *value*.

Defining a Population

See Also Applies To

A population is defined by a number of text lines stored in a memory buffer. The text is placed in the buffer using the instruction SEQ_BUFFER.

Each population definition consists of a line defining its name followed by a number of other lines defining its attributes.

The number of attributes of each type is limited to 50 for a population, e.g. 50 domains, 50 natures etc.

If more attributes are added, they cause an error in the Log Viewer when the application is loaded: '\popu.dat line 104...'.

A work-around of defining more populations would impose system overheads.

Syntax for population name

POPULATION, *Version*, *Name*, *0*, *0*, *Comment*

Argument	Meaning
Version	Always 23.
Name	The name by which the population will be known.
Comment	A free format comment.

Syntax for population attributes

POPDEF, *Name*, *Attribute*, *Mode*, *Value*

Argument	Meaning
Name	The name of the population to which the attribute is to be applied.
Attribute	The attribute type: TATT1 A domain. TATT2 A nature FLAG A hexadecimal mask. 0080 Equipment variable. 0100 Internal variable. 0200 External variable. 0400 Broadcast variable. 1000 DDE variable. TYPE A hexadecimal mask. 0001 Bit variable 0002 Register variable 0004 Text variable 0008 Alarm variable
Mode	IN Include in the population. OUT Exclude from the population.
Value	The value of the attribute. For example if the attribute parameter is FLAG, it could be the value 0080.

File syntax for VARIABLE mode IMPORTFILE and IMPORTBUFFER

See Also Applies To

The instruction VARIABLE, modes IMPORTFILE or IMPORTBUFFER, are used to add, modify or delete configuration items from associated with variables. The changes are imported from a file or buffer.

This method is no longer recommended for configuration import. Instead it is preferable to use the Smart Generator and the Generic XML Import.

Adding or modifying items

The following items can be added or modified. The syntax of the entry in the file or buffer must comply to the syntax used in the item's project configuration file.

The configuration files are found in the project's C folder.

Configuration item	Project configuration file for syntax
Variables	VAREXP.DAT
Expression on variable	EXPRV.DAT
Expression template	EXPRM.DAT
Proprietary trend	HISTO.DAT
HDS trend	HDSTREND.DAT

Deleting items

The following items can be deleted. The syntax of the entry in the file or buffer is as follows.

Configuration item	Syntax
Variable	DNVAR, VariableName
Proprietary trend	DTR, VariableName, UnitName
HDS trend	DTR_HDS, VariableName, LogicalGroupName
Expression on variable	DEXPRESSIONONVAR,ExpressionName
Expression template	DEXPRM, ExpressionName

Event Masks

Applies To

Lower mask (Modes EVENTMASK and EVENTMASKEX)

Value	Meaning
0	None
1	Bit change
2	Acknowledge alarm
4	Send bit
8	Send register (operator action)
16	Send text (operator action)
32	Send recipe (operator action)
64	Bit change to 0
128	Login (operator action)
256	Logout (operator action)
512	Bit change to 1
1024	Bit change to invalid
2048	Alarm on, acknowledged.
4096	Alarm on, not acknowledged
8192	Alarm off, acknowledged
16384	Alarm off, not acknowledged
32768	Alarm not set (NS)
65536	Acknowledge alarm (operator action)
131072	Run a program (operator action)
262144	Alarm off
524288	Alarm on
1048576	Alarm not available
2097152	Alarm inhibited
4194304	Alarm masked by program
8388608	Alarm masked by variable
16777216	Alarm masked by User
33554432	Alarm masked by expression
67108864	User mask alarm (operator action)
134217728	User unmask alarm (operator action)
268435456	Alarm maintenance on (operator action)
536870912	Alarm maintenance off (operator action)
1073741824	Login attempt failed (operator action)

Upper mask (Mode EVENTMASKEX only)

Value	Meaning
1	Send to 0 (operator action)
2	Send to 1 (operator action)
4	State 0
8	State 1

It is simple to combine two or more event masks using the LOGICAL instruction.

Event Maintenance by Program

Applies To

The format in which events are imported from a file is the same as the format in which they are stored if they are entered manually in the Application Explorer.

- Configured events (whether imported or manually entered) are saved in an ASCII file, EVENT.DAT, in the project's C folder.
- Events can be imported from any ASCII file with the appropriate syntax. By default the file must be located in the project's TP folder.

Format of Events

The storage and/or import format for an Event is as follows

No	Description	Type	Size(bytes)	Value
1	Record type	Character		EVTPROG
2	Event name	Character	40	
3	Description	Character	80	
4	Reserved	Numeric		
5	Reserved	Numeric		
6	Name of server list	Character		
7	Reserved	Character		
8	Name of triggering variable	Character	40	
9	When triggered on bit	Numeric		0 = to 0. 1 = to 1. 2 = by an expression
10	Enable flag	Numeric		1 for enabled, else 0
11	Name of enabling bit	Character		
12	Trigger expression	Character	240	
13	Program name	Character		
14	Branch name	Character		
15	Function name	Character		
16	List of arguments	Character		

Deleting an Event

The format for deleting an existing Event entry is the same as adding an event except that the record type is DEVTPROG.

Native Filter Expressions

Applies To

The Supervisor's native filter expressions can be used with the following instructions.

- ALARMDISPLAY
- LOGDISPLAY
- SVALA
- SVLOG
- SVTREND
- SVBRANCH

A filter starts with an equals sign and is followed by one or more expressions combined with the logical operators. Each expression comprises an attribute, an operator and a value all enclosed in brackets.

For example =(#t BEG ALM) would include all variable names (#t) beginning (BEG) with the text ALM.

The following symbols are supported:

Symbol	Attribute
#t	Variable name.
#T	Variable label (description).
#B	Boolean attribute (as a word).
#A1 to #A16	Extended attributes. A1 and A2 are reserved for Domain and Nature.

Symbol	Operation
==	Equal.
!=	Not equal.
>	Greater than.
<	Less than
>=	Greater than or equal to.
<=	Less than or equal to.
^ or BEG	Beginning with "string"
[or INC	Including "string"
$ or END	Ending with "string"
&&	Logical AND used between expressions.
||	Logical OR used between expressions.

Attributes names are not permitted to include spaces. For example (#A5==T 5) is not permitted.

Examples of Filter Expressions

Filter	Meaning
=(#t BEG FACTORY)&&((#A1 == DOM1)||(#A1 == DOM2))	The name of the variable starts with FACTORY and the Domain is either DOM1 or DOM2.
=(#A1 == DOM1)||(#A2 == NAT1)	The Domain is DOM1 or the Nature is NAT1.
=(#A5 >=10)&&(#A5 < 35)	The attribute A5 is greater than or equal to 10 and less than 35:

How to Use Instructions That Have Asynchronous Behavior

See Also

What is asynchronous behavior?

Asynchronous behavior is when the execution of an instruction starts a process that runs independently of the SCADA Basic program that called it. The SCADA Basic program execution continues without waiting for the result of the independent process. Most of the instructions that delegate part or all of the processing to another module behave in this way to avoid unacceptable delay in program execution or it being blocked (waiting for the other processing to be completed). For example, instructions related to:

- Access to historical data where waiting for the data would result in an unacceptable delay,
- Handling of communication objects (start, stop...) where waiting for the end to end execution can take a long time depending on your configuration (time-out), driver and network performances,
- Script instructions requiring processing on another Supervisor station on the network.

The syntax of all these instructions includes an argument that is the name of a variables. The real-time value of the variable indicates the progress status of the independent process. For some instructions the variable is mandatory and a change in variable value is used to initiate further action when the independent process is complete. For other instructions the variable is optional and may be used to initiate further action, or as an indication to the user, or not at all.

Instructions and modes which have asynchronous behavior

Instruction	Modes	Status variable
EXPORT	GENERATE, GENERATE_DATES AND GENERATE_PERIOD	Optional
EXPORT_LOG	GETRECORD and GETSTATISTIC	Mandatory
EXPORT_TREND	GETRAW, GETSAMPLED, GETSTATISTIC and GETAGGREGATED	Mandatory
FILETRANSFER	DOWNLOAD, DOWNLOAD_DIRECTORY and DELETE	Optional
HISTORY	GETTREND	Optional
MAPDISPLAY*	LOADMARKERS	Optional
SELECTOR	HISTORICAL	Optional
SVBATCH*	SELECT	Optional
SVLOG**	EXTRACT	Optional
SVTREND**	GETTREND	Optional

* Not associated with historical data.

** The status variable is optional in the syntax although should be considered mandatory in all but the most trivial of uses.

How to use the status variable to initiate processing of the results

The program must monitor the change in value of the status variable in order to take informed decisions depending on the execution progress and success or failure to complete of the asynchronous process.

The accepted method to use the status variable is to programmatically add an event that is triggered by the change in value of the status variable. The event then runs a program which processes the result of the asynchronous instruction. The event is added before execution of the asynchronous instruction and deleted after the result of the asynchronous instruction has been processed. Its structure might look something like this.

```
Sub Initialize ()
'-------- Add event
  EVENT("ADDPROG", "StatusVarName", "TriggerCondition", "MyProgram", "",
"ProcessResult");

'-------- Asynchronous instruction
  INSTRUCTION1("MODE", "P1", "P2",...., "StatusVarname");
End Sub

Sub ProcessResult ()
'-------- Process result of asynchronous instruction
  INSTRUCTION1;
  INSTRUCTION2;
```

```
  .....
  .....

'-------- Delete event
  EVENT("DELPROG", "StatusVarName", "TriggerCondition", "MyProgram", "",
"ProcessResult");
End Sub
```

Using the DELAY instruction is not a solution with regard to the asynchronous nature of such instructions. It must not be used as a replacement to correctly handling status variables because:

- It blocks the execution of the caller script for the duration of the delay,
- There is no guarantee that the asynchronous operation is completed within the delay.

Example

```
'-------- Simplified Export_Log example
Sub Export_Log_EX1()
  Const BIT_TO_0 = 8192, BIT_TO_1 = 16384;
  Dim dTimenow as Double, dEvents As Double;

'-------- Initialise variables
  dTimeNow = DateTimeValue();
  dEvents = BIT_TO_0 + BIT_TO_1;

'-------- Set up event
Event("ADDPROG", "SVB.ExportLogStatus", "S>S", "E.svb", "", "OnChangeOfStatus");

'-------- Start Export_Log
  Export_Log("GETRECORD", "Loglist02", dTimeNow - 600000, dTimeNow, dEvents, 0, 29, "",
"@SVB.ExportLogStatus", ";", ",", 0, "#D/#M/#Y|#h:#m:#s|#E|#T", "Date|Time|Event|Title");
End Sub

'-------- On change of status variable process if complete else clean up
Sub OnChangeOfStatus()
  Dim sStatus As Single;
  Dim lBufferHandle As Long;
  Dim iLines As Integer;

  sStatus = SVB.ExportLogStatus;
  If(sStatus==0 || sStatus==4 || sStatus ==5) Then ' Export complete or reached limits
    Print("Export_Log complete - status is ", sStatus);
    lBufferHandle = Alloc_Buffer(102400);
    iLines = Export_Log("READBUFFER", lBufferHandle);
    If (iLines > 0) Then
      BufToExcel(lBufferHandle, ";", ",", "export.xlsx", "ex1", "APPEND", iLines);
      Print("Exported ",iLines, " lines to Excel");
    End If
    Export_Log("DISPOSE");
    Free_Buffer(lBufferHandle);
    Event("DELPROG", "SVB.ExportLogStatus", "S>S", "E.svb", "", "OnChangeOfStatus");
  Else
    If(sStatus==1) Then ' Export running - do nothing
      Print("Export_Log running");
    Else ' Export failed - cancel and clean up
      Print("Export_Log failed - status is ", sStatus);
      Export_Log("CANCEL");
      Export_Log("DISPOSE");
      Event("DELPROG", "SVB.ExportLogStatus", "S>S", "E.svb", "", "OnChangeOfStatus");
    End If
  End If
End Sub
```

Key Codes

See Also Applies To

Key	Key Code	Standard Action
F1	1 Shift type D - not available	Online help
F2	2	Logon / Logoff
F3	3	User accounts
F4	4	About window
F5	5	-
F6	6	Show menu
F7	7	Event viewer
F8	8	Hide menu
F9	9	Programs
F10	10 Shift type D - not available	Quit application
F11	11	Data exports
F12	12	VCR
F13	13	-
F14	14	-
F15	15	-
F16	16	-
Back space	17	-
Tab	18 Do not use.	-
Enter	19 Do not use.	-
Shift	20	-
Ctrl	21 Do not use.	-
Alt	22 Do not use.	-
Pause	23 Do not use.	-
Esc	24	-
Space	25 Do not use.	-
Page up	26	Previous window
Page down	27	Next window
End	28	Last zone
Home	29	First zone
Left arrow	30	-
Up arrow	31	Previous zone
Right arrow	32	-
Down arrow	33	Next zone
Print	34 Do not use.	
Insert	35	-
Delete	36	-
Numlock	37 Do not use.	
	121	Variable selector
	123	LonWorks configuration

An example of setting key codes is shown in the topic for the KEY instruction.

Masking by Program

See Also Applies To

Masking is normally a temporary operation connected with the current state of the process being monitored. A variable that has any mask other than 0 continues to be scanned but its value will not be propagated outside the real time database. Hence any activities connected to that variable will stop. (This includes alarming and recording to disk). The value is still accessible however, by a SCADA BASIC program. The mask of a variable may be set and cleared from several sources, each represented by a binary weight in the mask:

- By a SCADA BASIC program. Up to four levels of mask may be set in this way:

  1 = Level 1
  2 = Level 2
  4 = Level 3
  8 = Level 4

- By the operator. For example, an alarm bit may be masked using an option available from the Alarm Viewer. The binary weight of this option is 16.

- By dependence. If the value of a variable depends on another, for example a threshold bit, the source variable going invalid will cause the dependent variable mask to be set causing it also to be invalid. The binary weight of this option is 32. Another example of a variable that may appear as masked is an alarm that is inhibited by another alarm or bit.

When a variable is unmasked, its value immediately becomes available.

Message Encoding

Applies To

The following message encoding may be used when sending an e-mail using either the EMAIL or EVENT instructions.

Code	Encoding
874	Thai (Windows)
932	Japanese (Shift-JIS)
936	Chinese Simplified (GB2312)
949	Korean
950	Chinese Traditional (Big 5)
1250	Central European (Windows)
1251	Cyrillic (Windows)
1252	Western European (Windows)
1253	Greek (Windows)
1254	Turkish (Windows)
1255	Hebrew (Windows)
1256	Arabic (Windows)
1257	Baltic (Windows)
1258	Vietnamese (Windows)
20127	US-ASCII
28591	Western European (ISO)
28592	Central European (ISO)
28594	Baltic (ISO)
28595	Cyrillic (ISO)
28596	Arabic (ISO)
28597	Greek (ISO)
28599	Turkish (ISO)
28603	Estonian (ISO)
50220	Japanese (JIS)
50225	Korean (ISO)

Parameter Buffer Format for Alarm Data

See Also Applies To

The parameter buffer must contain a string constructed as follows:

MaxLines[,*Direction*[,*MinPriority*[,*MaxPriority*[,*ONNOACK*[,*ONACK*[,*OFFNOACK*[,*OFF*[,*NS*[,*Mask*[,*Filter*[,*Format*]]]]]]]]]]

Parameter	Meaning
MaxLines	The maximum number of lines to be extracted. A value of 0 indicates no limit.
Direction	Sort result: 0: Latest event at top. 1: Latest event at bottom.
MinPriority	Minimum alarm priority.
MaxPriority	Maximum alarm priority.
ONNOACK	1: include On, Not Ack items; 0: omit.
ONACK	1: include On, Ack items; 0: omit.
OFFNOACK	1: include Off Not Ack items; 0: omit.
OFF	1: include Off items; 0: omit.
NS	1: include invalid items; 0: omit.
Mask	Mask constants for log data type: 1: Bit transition 2: Alarm acknowledge 4: Send a command 8: Send a control value 16: Send text 32: Send a recipe 128: Log on 256: Log off 64: Bit transition to 0 512:Bit transition to 1 1024: Bit transition to NS 1600: All bit transitions 2048: Alarm On Ack 4096: Alarm On Nack 8192: Alarm Off 16384: Alarm Off Nack 32768: Alarm NS. 65536:Operator ack 63488: All alarm transitions 131072: Run program 262144: Alarm on 524288: Alarm off
Filter	A string defining an alarm filter. See the topic on FNative Filter Expressions. If null then there is no filter.
Format	A string defining the format of the extracted data. This uses the same syntax as that for defining the format of each line in an Alarm Viewer. If null, the default will be used: #D/#M/#Y #h:#m:#s #E #T #C.

Parameter Buffer Format for Log Data

See Also Applies To

The parameter buffer must contain a string constructed as follows:

LogList,StartTime,EndTime[,*MaxLines*[,*Direction*[,*Mask*[,*MinPriority*[,*MaxPriority*[,*Filter*[,*Format*]]]]]]]

Parameter	**Meaning**
LogList	The name of a log list.
StartTime	Start time for extraction.
EndTime	End time for extraction.
MaxLines	The maximum number of lines to be extracted. A value of 0 indicates no limit.
Direction	Sort result: 0: Latest event at top. 1: Latest event at bottom.
Mask	Mask constants for log data type: 1: Bit transition 2: Alarm acknowledge 4: Send a command 8: Send a control value 16: Send text 32: Send a recipe 128: Log on 256: Log off 64: Bit transition to 0 512:Bit transition to 1 1024: Bit transition to NS 1600: All bit transitions 2048: Alarm On Ack 4096: Alarm On Nack 8192: Alarm Off 16384: Alarm Off Nack 32768: Alarm NS. 65536:Operator ack 63488: All alarm transitions 131072: Run program 262144: Alarm on 524288: Alarm off
MinPriority	Minimum alarm priority.
MaxPriority	Maximum alarm priority.
Filter	A string defining an alarm filter. See the topic on FNative Filter Expressions. If null then there is no filter.
Format	A string defining the format of the extracted data. This uses the same syntax as that for defining the format of each line in an Alarm Viewer. If null, the default will be used: #D/#M/#Y #h:#m:#s #E #T #C.

Regular Expressions

See Also Applies To

The regular expressions used by SELECTOR are derived from the notation used in automata theory to describe formal languages and finite state machines.

Regular expressions consist of characters: operands such as "a", "O", and ", " and meta-characters (operators) such as "+", "|", and "[". A regular expression, which is similar to the familiar arithmetic expressions, can be either a basic expression or a complex expression that is formed by applying operators to several basic expressions.

The regular meta-characters for expressions, and their uses, are as follows:

Operator	Used for
\	Used in escape sequences to specify characters that would otherwise have no representation (similar to those used in the C language).
\b	Backspace.
\t	Horizontal tab.
\n	Newline or linefeed.
\f	New page or form feed.
\r	Carriage return.
ddd	Octal value.
c	*c* represents any character string which is taken literally.
^	The carat matches the beginning of a string. For example, "^abc" matches only those strings which start with "abc". When used as the first character of a character class it denotes a negated class.
$	The dollar matches the end of a string. For example, "$z" will only match strings with "z" as the last character.
.	The full stop matches any single character. Be careful as ".*" will match everything.
[	The open bracket denotes the start of a character class.
]	The closed bracket denotes the end of a character class.
|	The pipe character is the alternation operator, a|b" matched either "a" or "b".
()	Parentheses are used to group expressions in much the same way as for arithmetic expressions.
*	The closure operator matches 0 or more instance(s) of the specified string.
+	The plus sign indicates positive closure and matches 1 or more of the specified expression. For example, "+z" matches a string of 1 or more "z"s.
?	The question mark matches 0 or 1 instance of the specified expression. For example "?9" matches either the null string or "9".

Character classes

Character classes are shorthand for matching one of several characters.

For example [AaBb] is the same as (A|a|B|b)and matches "A", "a", "B", and "b".

There are also character ranges, such as [A-Z] which matches any uppercase character.

Negated character classes specify characters that should not be matched. For example, [^A-Z] matches everything except upper case alpha characters.

Examples of Regular Expressions

All variable names that have a first branch of "System":

```
^System
```

All variable names that end in a number:

```
[0-9]$
```

All variable names:

```
.*
```

Recipe Buffer Format

Applies To

The buffer used by the Recipe instruction modes READ and CREATE takes the following format.

```
Recipe header
Recipe variable definition 1
Recipe variable definition 2
. . .
Recipe variable definition N
```

Recipe header format

H,*Number*,*Name*,*Family*,*Level*,*AllowVar*,*TrackVar*,*Branch*,,*Type*,,*SendFlag*,*SendMode*,0,*OPCMode*

Argument	Meaning
H	Always H. Flag indicating the start of the recipe header.
Number	The recipe number.
Name	The recipe name.
Family	The recipe family. Optional
Level	The recipe level. Optional
AllowVar	The name of the recipe sending allowance bit variable. Optional
TrackVar	The name of the recipe tracking register or text variable. Optional
Branch	The recipe branch. Optional
Type	A flag indicating the recipe type. 0 = Proprietary 1 = Database
SendFlag	A flag indicating if the recipe should be sent if one or more of the variables is invalid. 0 = Don't send 1 = Send
SendMode	The send mode for the recipe. 0 = Multiple 1 = Block
OPCMode	The OPC mode. 0 = Optimised serialisation 1 = Full serialisation 2 = No optimization 3 = Full optimization

Recipe variable format

V,*VarName*,1,*Value*,*Minimum*,*Maximum*,0,*ValueVarName*

Argument	Meaning
V	Always V. Indicating the start of a variable definition.
VarName	The name of the variable.
Value	The value to which the variables is to be set or null if the value is contained in another variable.
Minimum	The minimum value to which the variable can be set.
Maximum	The maximum value to which the variable can be set.
ValueVarName	The name of a variable containing the value to which the variable is to be set. Optional

SENDLIST Send Mode

Applies To

The sending mode (*OPCMode*) parameter of the SENDLIST instruction determines exactly how the variables are sent depending on the communication method being used.

For equipment variables

- BLOC - The variables are sorted according to the communication frame that they belong to. All variables between the highest and lowest address (according to the list created by SET) are sent including any not in the list.
- MULTIPLE - The variables are sorted according to the communication frame that they belong to and into contiguous blocks. Only the variables from the list created by SET are sent, but a larger number of writes may be required (one write for each block).

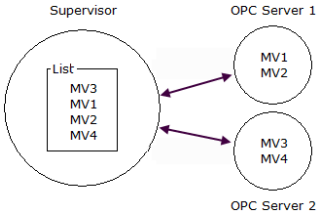
If the list being sent by SENDLIST contains only OPC variables either mode can be used.

For OPC variables

There are four options when sending variables to an OPC server, selected by the *OPCMode* parameter. The options are best illustrated by example.

Note that the order in which the variables are declared in the list directly affects the sequence in which t are written to the OPC servers.

The Supervisor is connected to two OPC servers. OPC Server 1 contains variables MV1 and MV2. OPC Server 2 contains MV3 and MV4. The Supervisor subscribes to MV1 and MV2 using the group Group1 and to MV3 and MV4 using Group2. A list is configured that contains all 4 variables in the order MV3, MV1, MV2, MV4.



- Mode 0 Optimized serialization
  - Write MV3 to OPC Server 2 and wait for the result.
  - If the result is OK, write MV1 and MV2 to OPC Server 1 and wait for the result.
  - If the result is OK, write MV4 and wait for the result.
  - The SENDLIST is completed. Three OPC writes are used.

- Mode 1 Full serialization
  - Write MV3 to OPC Server 2 and wait for the result.
  - If the result is OK, write MV1 to OPC Server 1 and wait for the result.
  - If the result is OK, write MV2 to OPC Server 1 and wait for the result.
  - If the result is OK, write MV4 and wait for the result.
  - The SENDLIST is completed. Four OPC writes are used.
- Mode 2 No optimization
  - Write MV3 to OPC Server 2.
  - Write MV1 to OPC Server 1.
  - Write MV2 to OPC Server 1.
  - Write MV4 to OPC Server 2.
  - Wait for all four results.
  - The SENDLIST is completed. Four OPC writes are used.
- Mode 3 Full optimization
  - Write MV3 and MV4 to OPC Server 2.
  - Write MV1 and MV2 to OPC Server 1.
  - Wait for all two results.
  - The SENDLIST is completed. Two OPC writes are used.

If the list being sent by SENDLIST contains only OPC variables, either mode can be used.

SVBATCH Data Structures

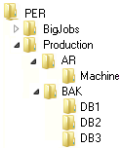
See Also Applies To

SVBATCH folder structure

The files that SVBATCH produces are stored in the project folder PER.

For each database there is a folder that contains one or more files, each of which represents a table (a list of records). Other folders contain backup and archive copies of the database.

Show picture



The Database File

This is held in the PER folder of the project: BATCHDB.DAT

It is automatically loaded on startup.

Format of the Parameter File

There is one INFOS.DAT file for each database located in the Database folder. The file contains one line with the following structure:

BASEINFOS, *DatabaseName*, *DatabaseType*, *NbAttributes*, *NbBackups*, , *AttributeMask*

Attribute	Meaning
DatabaseName	The name of the database and folder in which it is located.
DatabaseType	The type of database. For future use.
NbAttributes	The number of attributes configured to be stored.
NbBackup	The number of backup copies (in the folder BAK).
AttributeMask	A decimal number representing a binary mask that selects which of the attributes are to be used as indices when using the SELECT mode. (#A1 to #A32)

THIS MASK MUST BE EDITED MANUALLY.

Each attribute that is to be an index is represented by a binary 1. For example, to use attributes 1, 2, 4 and 5 the mask would be 110011 or 51 in decimal.

When mode SELECT is used, the text attributes used in the search expression must be indexed.

Example

```
BASEINFOS,DBREPORT,,19,3,,131071
```

where 131071 in decimal corresponds to the binary mask 1111111111111111.

It indicates that a search expression can be used on the first 17 text attributes. (#A1 to #A17)

Batch Report File

This is in the folder PER*DatabaseName* of the project: BatchId.RPT

Attribute	Meaning
BatchId	Identifies a batch.

The table (.rpt) files

Each table that you create in a database will file i

n the PER database folder. The file name is of the format *TableName*.rpt. The files are ASCII and may be opened with a text editor such as NotePad. Each file has the following structure.

```
BASE, "DatabaseName"
BATCH, "TableNAme",
DATE, "StartDate", "EndDate", "LongVal", "StringVal1", "StringVal2",.., "StringVal32"
```

There is a line starting with DATE for each record that is created.

Examples of a .rpt file

```
BASE,Production
BATCH,Machine 1
DATE,01/05/2002 06:00:00:000,01/05/2002 14:00:00:000,,"B100","Nuts","M4","22345"
DATE,01/05/2002 14:00:00:000,01/05/2002 22:00:00:000,,"B101","Nuts","M4","11222"
DATE,02/05/2002 08:00:00:000,02/05/2002 14:00:00:000,,"B102","Bolts","M8","897"
DATE,02/05/2002 14:00:00:000,02/05/2002 22:00:00:000,,"B103","Bolts","M4","5672"
DATE,03/05/2002 14:00:00:000,03/05/2002 22:00:00:000,,"B104","Bolts","M6","1234"
DATE,03/05/2002 14:00:00:000,03/05/2002 22:00:00:000,,"B105","Bolts","M6","5667"
DATE,04/05/2002 14:00:00:000,04/05/2002 22:00:00:000,,"B106","Bolts","M4","9876"
DATE,04/05/2002 14:00:00:000,04/05/2002 22:00:00:000,,"B107","Bolts","M6","5670"
```

```
BASE,DB3
BATCH,An2
DATE,08/04/1998 09:44:40:000,08/04/1998 09:48:41:000,1,"PRODUCT1","
PRODUCT2",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
DATE,08/04/1998 09:50:53:000,08/04/1998 09:57:05:000,2,"PRODUCT1",
,"PRODUCT3",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

Network Mode of SVBATCH

See Also Applies To

Description

It is possible to define lists of stations and to carry out the following operations on this list:

- SELECT to define a set of log entries.
- UPDATE to update a recording.
- The search criteria for UPDATE mode identify the batch and the start date. If a recording satisfies these criteria on different stations, it will be updated.
- DELETE works in there same way as UPDATE mode.
- CANCEL enables you to cancel a request for a selection (by SELECT or ARCHIVELIST). With this request, it is not necessary to carry out a GETNEXTBUFFER operation.
- ARCHIVE.
- BATCHLIST.
- ARCHIVELIST.
- GETNEXTBUFFER enables you to continue recording a selection when the SCADA BASIC buffer is too small to contain all the data.

The CREATE mode operates locally on the station.

Structure

All the requests are event-driven. The result of the request is obtained on transition to 1 of the configured event bit (in NETWORKBROADCAST).

The status buffer enables retrieval of the mode of the request in progress and to process it.

Once the process is completed, the NEXTCOMMAND mode enables you to reactivate the execution of requests.

The processing of all stored requests is held up until the NEXTCOMMAND mode is run.

Using Programs with a Branch

See Also

Programs may be started with a branch. Thereafter all variable names within the program that are not preceded by the @ character will be relative to that branch.

There are several ways in which a branch may be allocated to a program:

1. By using the TREE instruction within the program to change the name of the branch under program control.
2. From the Run-Program animation. Note that there is a limitation with the Main function (see note below).
3. When running a program or function from another program using the Program instruction.

If you are using the Run.Program animation to run the Main function of a program, the branch is not automatically set.

Instead you can use the instruction GETARG(12) to get the branch and TREE to set the branch.

Variable - Import Report Format

See Also Applies To

This method is no longer recommended for configuration import. Instead it is preferable to use the Smart Generator and the Generic XML Import.

The mode IMPORTFILE of the VARIABLE instruction generates an optional report file. Each line in the report file takes the following format.

No	Description
1	The station name.
2	Number of objects imported.
3	Number of variables added.
4	Number of variables modified.
5	Number of variables deleted.
6	Number of variables unchanged.
7	Number of objects for which the modification is not defined.
8	Number of failures.

Example

```
SERVER_1,116,0,20,0,0,96,0
CLIENT_1,116,0,20,0,0,96,0
```