Cat No. W09E-EN-12

Software

CX-Supervisor

Script Language Reference

OMRON

Notice

OMRON products are manufactured for use by a trained operator and only for the purposes described in this manual.

The following conventions are used to classify and explain the precautions in this manual. Always heed the information provided with them.

Note: Indicates information of particular interest for efficient and convenient operation of the product.

Caution:
Indicates information that, if not heeded, could possibly result in minor or relatively serious injury, damage to the product, or faulty operation.

Warning:
Indicates information that, if not heeded, could possibly result in serious injury or loss of life.

Trademarks and copyrights

MECHATROLINK is a registered trademark of Yaskawa Corporation.

Trajexia is a registered trademark of OMRON.

EtherCAT is a registered trademark of the EtherCAT Technology Group.

All other product names, company names, logos or other designations mentioned herein are trademarks of their respective owners.

Copyright

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

Notice 1

Trademarks and copyrights 1
Copyright 1

SECTION 1
Introduction 11

SECTION 2
Expressions 13

SECTION 3
Scripts 17

3-1 Object 17
3-2 Page 17
3-3 Project 17

SECTION 4
Script Language Reference 19

4-1 Points 22
4-1-1 Basic Point Assignment 22
4-1-2 Further Point Assignment 22
4-2 Logic and Arithmetic 23
4-2-1 Arithmetic Operators 23
4-2-2 Bitwise Operators 23
4-2-3 Logical Operators 24
4-2-4 Relational Operators 25
4-3 Control Statements 26
4-3-1 Simple Conditional Statements 26
4-3-2 Nested Conditional Statements 27
4-3-3 Case Select 29
4-3-4 FOR... NEXT Loop 30
4-3-5 DO WHILE/UNTIL Loop 31
4-4 Subroutines 32
4-4-1 Call 32
4-5 Punctuation 32
4-5-1 Quotation Marks / Command String Delimiters 32
4-5-2 Indentation 32
4-5-3 Multiple Commands 33
4-5-4 Parenthesis 33
4-5-5 Remarks 33
4-6 Indirection within Script Commands and Expressions 34
4-7 Point Arrays within Script Commands and Expressions 35
4-8 Using Aliases 35

SECTION 5
Functions and Methods.....39

5-1 Object Commands 44
5-1-1 Current Object 44
5-1-2 Other Objects 45
5-1-3 Blink 46
5-1-4 Colour 47
5-1-5 Disable 48
5-1-6 Height 49
5-1-7 Horizontal Fill 49
5-1-8 Move 50
5-1-9 Rotate 51
5-1-10 Vertical Fill 52
5-1-11 Visible 52
5-1-12 Width 53
5-2 Page Commands 53
5-2-1 Close Page 54
5-3 General Commands 54
5-3-1 Exponential 54
5-3-2 PlayOLE 54
5-3-3 DisplayPicture 55
5-3-4 PlaySound 56
5-3-5 Rand 56
5-3-6 RunApplication 56
5-3-7 RunHelp 57
5-3-8 SetLanguage 57
5-3-9 SetNYLED 58
5-3-10 GetPerformanceInfo 58
5-3-11 ShutDown 59
5-4 Communications Commands 59
5-4-1 CloseComponent 59
5-4-2 EnableOLE 60
5-4-3 EnablePLC 60
5-4-4 LaunchTroubleshooter 61
5-4-5 OpenComponent 61
5-5 Point Commands 62
5-5-1 CancelForce 62
5-5-2 CopyArray 62
5-5-3 DisableGroup 63
5-5-4 DisablePoint 63
5-5-5 EditPoint 63
5-5-6 EnableGroup 64
5-5-7 EnablePoint 64
5-5-8 Force 65
5-5-9 ForceReset 65
5-5-10 ForceSet 65
5-5-11 GetBit 66
5-5-12 InitialiseArray 66

5-5-13 InputPoint 66
5-5-14 OutputPoint 67
5-5-15 PointExists 67
5-5-16 SetBit 67
5-6 PLC Commands 68
5-6-1 OpenPLC 68
5-6-2 ClosePLC 68
5-6-3 GetPLCMode 69
5-6-4 SetPLCMode 69
5-6-5 SetPLCIPAddress 70
5-6-6 SetPLCPhoneNumber 71
5-6-7 UploadPLCProgram 71
5-6-8 DownloadPLCProgram 72
5-6-9 IsPLCOpen 72
5-6-10 PLCCommsFailed 73
5-6-11 PLCMonitor 73
5-7 Temperature Controller Commands 73
5-7-1 TCAutoTune 73
5-7-2 TCBackupMode 74
5-7-3 TCGetStatusParameter 74
5-7-4 TCRemoteLocal 75
5-7-5 TCRequestStatus 76
5-7-6 TCRspLsp 76
5-7-7 TCRunStop 77
5-7-8 TCSaveData 77
5-7-9 TCSettingLevel1 77
5-7-10 TCReset 77
5-8 Alarm Commands 78
5-8-1 AcknowledgeAlarm 78
5-8-2 AcknowledgeAllAlarms 78
5-8-3 AcknowledgeLatestAlarm 78
5-8-4 ClearAlarmHistory 79
5-8-5 CloseAlarmHistory 79
5-8-6 CloseAlarmStatus 79
5-8-7 DisplayAlarmHistory 79
5-8-8 DisplayAlarmStatus 80
5-8-9 EnableAlarms 80
5-8-10 IsAlarmAcknowledged 80
5-8-11 IsAlarmActive 81
5-9 File Commands 81
5-9-1 CloseFile 81
5-9-2 CopyFile 82
5-9-3 DeleteFile 82
5-9-4 EditFile 82
5-9-5 MoveFile 83
5-9-6 OpenFile 84
5-9-7 PrintFile 84
5-9-8 Read 84

5-9-9 ReadMessage 85
5-9-10 SelectFile 85
5-9-11 Write 86
5-9-12 WriteMessage 87
5-10 Recipe Commands 87
5-10-1 DisplayRecipes 87
5-10-2 DownloadRecipe 88
5-10-3 UploadRecipe 88
5-11 Report Commands 89
5-11-1 GenerateReport 89
5-11-2 PrintReport 89
5-11-3 ViewReport 90
5-11-4 EmailReport 90
5-12 Text Commands 91
5-12-1 BCD 91
5-12-2 Bin 91
5-12-3 Chr 91
5-12-4 FormatText 92
5-12-5 GetTextLength 93
5-12-6 Hex 93
5-12-7 Left 93
5-12-8 Message 93
5-12-9 Mid 94
5-12-10 PrintMessage 94
5-12-11 Right 94
5-12-12 TextToValue 95
5-12-13 ValueToText 95
5-12-14 EmailText 96
5-13 Event/Error Commands 96
5-13-1 ClearErrorLog 96
5-13-2 CloseErrorLog 97
5-13-3 DisplayErrorLog 97
5-13-4 EnableErrorLogging 97
5-13-5 LogError 97
5-13-6 LogEvent 98
5-14 Printer Commands 98
5-14-1 ClearSpoolQueue 98
5-14-2 EnablePrinting 98
5-14-3 PrintActivePage 99
5-14-4 PrintPage 99
5-14-5 PrintScreen 100
5-14-6 PrintSpoolQueue 100
5-15 Security Commands 100
5-15-1 Login 100
5-15-2 Logout 101
5-15-3 SetupUsers 101
5-15-4 ChangeUserPassword 101
5-16 Data Logging Commands 102

5-16-1 AuditPoint 102
5-16-2 ClearLogFile 102
5-16-3 CloseLogFile 102
5-16-4 CloseLogView 103
5-16-5 ExportAndViewLog 103
5-16-6 ExportLog 104
5-16-7 OpenLogFile 105
5-16-8 OpenLogView 106
5-16-9 StartAuditTrail 107
5-16-10 StopAuditTrail 107
5-16-11 StartLogging 107
5-16-12 StopLogging 108
5-17 Database Commands 108
5-17-1 DBAddNew 108
5-17-2 DBClose 109
5-17-3 DBDelete 110
5-17-4 DBExecute 110
5-17-5 DBGetLastError 111
5-17-6 DBMove 112
5-17-7 DBOpen 113
5-17-8 DBProperty 114
5-17-9 DBRead 115
5-17-10 DBSchema 116
5-17-11 DBState 117
5-17-12 DBSupports 117
5-17-13 DBUpdate 118
5-17-14 DBWrite 118
5-18 Serial Port Functions 119
5-18-1 InputCOMPort 119
5-18-2 OutputCOMPort 120
5-18-3 CloseCOMPort 120
5-18-4 OpenCOMPort 121
5-18-5 SetupCOMPort 121
5-19 ActiveX Functions 122
5-19-1 Getting a property value 122
5-19-2 Writing a property value 122
5-19-3 Executing a method 123
5-19-4 Responding to events 123
5-19-5 ExecuteVBScriptFile 124
5-19-6 GenerateEvent 124

SECTION 6
Script Example 127
6-1 Balloon Script 127

SECTION 7
Colour Palette 131

Appendix A
OPC Communications Control 135

A.1 Component Properties 135
A.2 Script Interface 135
A.3 Functions 135
A.3.1 Value 135
A.3.2 Read 136
A.3.3 Write 136

Appendix B
CX-Server Communications Control 137

B.1 Functions 137
B.2 Value 138
B.3 Values 138
B.4 SetDefaultPLC 139
B.5 OpenPLC 139
B.6 ClosePLC 139
B.7 Read 139
B.8 Write 139
B.9 ReadArea 140
B.10 WriteArea 141
B.11 RunMode 141
B.12 TypeName 141
B.13 IsPointValid 141
B.14 PLC Memory Functions 141
B.15 ListPLCs 142
B.16 ListPoints 142
B.17 IsBadQuality 143
B.18 ClockRead 143
B.19 ClockWrite 143
B.20 RawFINS 143
B.21 Active 144
B.22 TCGetStatus 144
B.23 TCRemoteLocal 144
B.24 SetDeviceAddress 144
B.25 SetDeviceConfig 145
B.26 GetDeviceConfig 145
B.27 UploadProgram 146
B.28 DownloadProgram 146
B.29 Protect 146
B.30 LastErrorString 147

Appendix C
OMRON FH Vision Controls 149

C.1 OMRON FH Image Window 149
C.1.1 Properties 149

C.1.2 Methods 151
C.1.3 Events 153
C.2 OMRON FH Panel Window 154
C.2.1 Properties 154
C.2.2 Methods 154
C.2.3 Events 155
C.3 OMRON FH Text Window 155
C.3.1 Properties 155
C.3.2 Methods 157
C.3.3 Events 158

Appendix D
Obsolete Script Functions 161
D.1 Sleep 161
D.2 DDE Commands 161
D.2.1 DDEExecute 162
D.2.2 DDEInitiate 162
D.2.3 DDEOpenLinks 163
D.2.4 DDEPoke 163
D.2.5 DDERequest 164
D.2.6 DDETerminate 165
D.2.7 DDETerminateAll 165
D.2.8 EnableDDE 165
D.3 Graph Commands 166
D.3.1 ClearGraph 166
D.3.2 StartGraph 166
D.3.3 StopGraph 166
D.3.4 EditGraph 167
D.3.5 SaveGraph 168
D.3.6 Snapshot 168
D.4 Printer Commands 168
D.4.1 GetSpoolCount 168
D.4.2 SetPrinterConfig 169
D.5 JScript 169
D.5.1 ExecuteJScript 170
D.5.2 ExecuteJScriptFile 170
D.6 Atan 170
D.7 Sqrt 170
D.8 GetPointValue / SetPointValue 170
D.9 Colour Palette (ARGB Values Used in Script Code) 171

Appendix E
CX-Supervisor Script Language 173
E.1 Points 174
E.1.1 Basic Point Assignment 174
E.1.2 Further Point Assignment 174
E.2 Logic and Arithmetic 175

E.2.1 Arithmetic Operators 175
E.2.2 Bitwise Operators 176
E.2.3 Logical Operators 176
E.2.4 Relational Operators 177
E.3 Control Statements 178
E.3.1 Simple Conditional Statements 178
E.3.2 Nested Conditional Statements 179
E.3.3 Case Select 181
E.3.4 FOR... NEXT Loop 182
E.3.5 DO WHILE/UNTIL Loop 183
E.4 Subroutines 183
E.4.1 Call 183
E.4.2 Return 184
E.5 Punctuation 184
E.5.1 Command String Delimiters 184
E.5.2 Indentation 184
E.5.3 Multiple Commands 185
E.5.4 Parenthesis 185
E.5.5 Quotation Marks 185
E.5.6 Remarks 185
E.6 Indirection within Script Commands and Expressions 186
E.7 Point Arrays within Script Commands and Expressions 187
E.8 Using Aliases 187

Appendix F
Glossary of Terms 191

Revision history 199

SECTION 1
Introduction

This reference manual describes the script language syntax as a supplement to the CX-Supervisor User Manual. It provides detailed definition of the syntax of CX-Supervisor scripts that drive project, page, object actions and CX-Supervisor expressions as used by objects and scripts.

Typographic conventions used in the examples in this reference manual are as follows:

- Script commands and reserved words are shown in the preferred case, which may be either lower-, upper- or mixed-case.
- Points are shown in lower-case. Objects are shown in upper-case.

The following terms are used in this reference manual:

- Application. A set of files, containing an executable file, that carry out certain tasks. This reference manual refers to the Microsoft Excel and Microsoft Word for Windows applications.
- Constant. A point or object within a script that takes only one specific value.
- Executable. A file that contains programs or commands, and has an '*.EXE' extension.
- Nesting. To incorporate one or more IF THEN ELSE/ELSEIF END IF statements inside a structure of the same kind.
- Operands. Constants or point variables.
- Operators. Relational, arithmetic, and logical statements, for instance '+', '<=' or 'AND'.
- Or ('|'). The '|' symbol is used to represent 'or', where there are two or more forms of the same syntax.
- Point Types. Either Boolean, Integer, Real or Text.
- Point Variable. A point or object within a script that may take different values.
- Strings. Data in the form of text delimited by quotation marks (" "), which can be assigned to a point.
- The '{' and '}' braces. Must be inserted around the argument command or an error is reported. An error is reported if there are spaces between braces.
- 'TRUE' and 'FALSE'. Refer exclusively to Boolean states, where Boolean state 0 is 'FALSE' and Boolean state 1 is 'TRUE'.

SECTION 2
Expressions

This chapter describes the use of expressions within scripts.

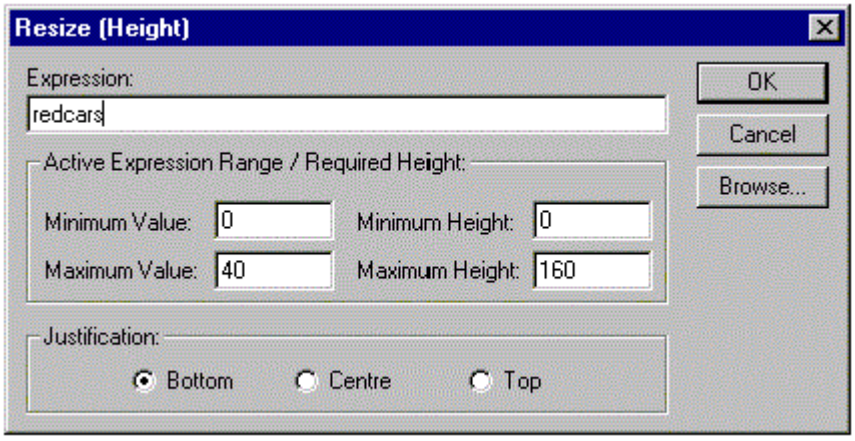
Expressions consist of operators and operands:

- Operators are relational, arithmetic, logical and include many functions.
- Operands are constants or point variables.

Expressions can be used in a script as part of a statement (refer to chapter 3 Scripts, chapter 4 CX-Supervisor Script Language, and Chapter 6 Functions and Methods). However expressions can be applied to the following actions directly using the associated Expression: or Digital Expression: field:

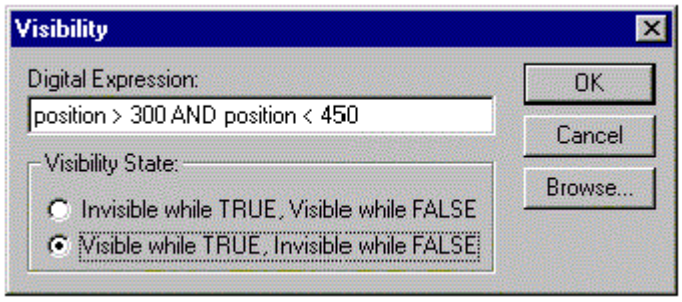
- Blink.
- Close page.
- Colour Change (Analogue).
- Colour Change (Digital).
- Display Status Text.
- Display Text Point.
- Display Value.
- Edit point value (Analogue).
- Edit point value (Digital).
- Edit point value (Text).
- Enable/Disable.
- Horizontal move.
- Horizontal percentage fill.
- Resize height.
- Resize width.
- Rotate.
- Show page.
- Vertical move.
- Vertical percentage fill.
- Visible.

The following example of a simple expression contains a point ('redcars') attached to a particular object with an appropriate object action, Resize (Height). At runtime, once the value of the point has been met within the attributes declared within the Active Expression Range/Required Height: fields, the current object is resized accordingly. This example is an Integer or Real example, whereby the value of the point either falls inside or outside the specified range. In this example, the point 'redcars' must fall between 0 and 40 for the expression to be met.

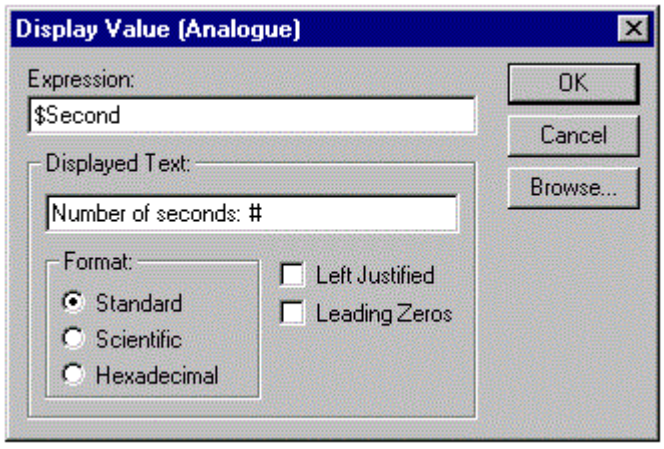


The following example of a more complex expression contains a test on point 'position'. If 'position' is more than 300 in value, and 'position' is less than 450 in value, i.e. the value of 'position' is between 300 and 450, then the expression has been met, and an action is initiated (in this instance the current object is made visible if the expression is met). This example is a Boolean example, whereby either the expression is met ('TRUE') or not met ('FALSE'). A Boolean value is always returned from a Digital Expression: field, as opposed to an Expression: field, which returns an Integer or Real value.

Operators used within this example are fully described in chapter 4, Logic and Arithmetic.



The following example of an expression contains a value point 'prompt' which is included at the value position denoted by a '#' symbol.



Refer to the CX-Supervisor User Manual for detailed dialog descriptions.

Note: Boolean Expressions execute when the expression is TRUE so it can be said that every Boolean expression has an inferred "== TRUE". Sometimes Boolean expressions can be difficult to read e.g. "bMyFlagPoint" or "BitMask & 0x80. It can help maintenance if this "== TRUE" is explicitly specified e.g. "bMyFlagPoint == TRUE" or "BitMask & 0x80 == TRUE".

Note: When using Boolean operators (e.g. ==, !=, &&, ||, |) never mix tests for Boolean and non Boolean operands. For example never use "bMyFlagPoint == 1" or "bMyFlagPoint == 0". Instead always test using the correct Boolean constant i.e. "TRUE" or "FALSE" for CX-Supervisor scripts, or "True" and "False" when using VBScript.

Note: On Condition scripts are only executed when the expression is TRUE. Sometimes this leads to peculiar results, for example using $Second as it will be executed when $Second changes to 59, and to 1 but not when it changes to 0. To execute a condition script any time a point changes, force the expression to always evaluate to TRUE for example "$Second || TRUE". This works because the $Second forces the expression to be tested when the point changes, but the || TRUE means the test will return TRUE regardless of the value of the point.

Note: Use array points in On Condition expressions with caution. The expression "MyArray[3] == 1" does not mean "execute every time the third element changes to 1". It means execute when any element of MyArray changes and the third element happens to be 1

Note: Using an array point without any index is the same as specifying element 0 i.e. MyArray actually means MyArray[0] == 1

SECTION 3
Scripts

A CX-Supervisor script is a simple programming language used to manipulate points. Scripts can be created at different levels, at object level, page level or project level. Although the script code can be applied to all levels of script, there are subtle differences, described in the following paragraphs.

3-1 Object

If a script is executed as a runtime action of an object, then the script can affect the object of the action, or any other, depending on the actual content of the script.

3-2 Page

Page scripts are concerned with manipulating points and graphical objects that are used or included within that page. In other words page scripts are used to drive a number of actions on the occurrence of a particular event. These actions may manipulate several graphical objects on one page.

3-3 Project

Scripts can be applied to a project to manipulate points. These scripts are associated with events that occur throughout the whole operating session

SECTION 4
Script Language Reference

This chapter provides the Script Language reference supported by CX-Supervisor, which is based on Microsoft Visual Basic scripting language called VBScript. These features are provided by the Windows Scripting Host, included by default with Windows and Internet Explorer.

It also provides detailed information on the syntax used to drive project, page and object actions. In conjunction with the script functions and methods described in Chapter 6, the script language provides a very powerful, fast and full featured programming language.

Note: CX-Supervisor currently supports 'VBScript', the recommended script language, plus its own 'CX-Supervisor Script' script language.Ultimately, the 'CX-Supervisor Script' script language may be depricated and therefore it is recommended that VBScript is used in CX-Supervisor applications. The 'CX-Supervisor Script' script language reference is now located in the Appendix section.

For a full User Guide, Language reference and details of the latest versions and support contact Microsoft at http://msdn.microsoft.com

Category	**Keyword / Feature**
Array handling	Array Dim, Private, Public, ReDim IsArray Erase LBound, UBound
Assignments	Set
Comments	Comments using ' or Rem
Constants/Literals	Empty Nothing Null True, False
Control flow	Do…Loop For…Next For Each…Next If…Then…Else Select Case While…Wend With
Conversions	Abs Asc, AscB, AscW Chr, ChrB, ChrW CBool, CByte CCur, Cdate CDbl, CInt CLng, CSng, CStr DataSerial, DateValue Hex, Oct Fix, Int Sgn TimeSerial, TimeValue

Category	Keyword / Feature
Date / Times	Date, Time DateAdd, DateDiff, DatePart DateSerial, DateValue Day, Month, MonthName Weekday, weekdayName, Year Hour, Minute, Second Now TimeSerial, TimeValue
Declarations	Class Const Dim, Private, Public, ReDim Function, Sub Property Get, Property Let, Property Set
Error Handing	On Error Err
Expressions	Eval Excute RegExp Replace Test
Formatting Strings	FormatCurrency FormatDateTime FormatNumber FormatPercent
Input / Output	InputBox LoadPicture MsgBox
Literals	Empty False Nothing Null True
Math	Atn, Cos, Sin, Tan Exp, Log, Sqr Randomize, Rnd
Miscellaneous	Eval Function Execute Statement RGB Function
Objects	CreateObject Err Object GetObject RegExp

Category	Keyword / Feature
Operators	Addition (+), Subtraction (-) Exporentiation (^) Modulus arithmetic (Mod) Multiplication (*), Division (/) Integer Division (\) Negation (-) String concatentation (&) Equality (=), Inequality (<>) Less Than (<), LessThan or Equal(<+) Greater Than (>) Greater Than or Equal To (>=) Is And, Or, Xor Eqv, Imp
Options	Option Explicit
Procedures	Call Function, Sub Property Get, Property Let, Property Set
Rounding	Abs Int, Fix, Round Sgn
Script Engine ID	ScriptEngine ScriptEngineBuildVersion ScriptEngineMajorVersion ScriptEngineMinorVersion
Strings	Asc, AscB, AscW Chr, ChrB, ChrW Filter, InStr, InStrB InStrRev Join Len, LenB LCase, UCase Left, LeftB Mid, MidB Right, RightB Replace Space Split StrComp String StrReverse LTrim, RTrim, Trim
Variants	IsArray IsDate IsEmpty IsNull IsNumeric IsObject TypeName VarType

4-1 Points

4-1-1 Basic Point Assignment

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value.
expression	The value to be assigned to pointname. The expression may be of type Boolean, Integer, Real or Text.

Typical Examples

```
count = 100
```

The Integer or Real point 'count' is assigned the value 100.

```
result = TRUE
```

The Boolean point 'result' is assigned the state ''TRUE".

```
name = "Valve position"
```

The Text point 'name' is assigned the associated text, contained within quotation marks.

Note: When assigning Real (floating point) values to an Integer point the assignment uses the 'Symetrical Rounding Down' (towards 0) standard. This means a value of 4.1 would be assign a value 4. A value of -4.1 would asign a value of -4.

References

Refer to chapter 4, Punctuation for details of the use of quotation marks.

4-1-2 Further Point Assignment

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value.
expression	The value to be assigned to pointname. The expression may be of type Boolean, Integer or Real and can include other points, logical or arithmetical expressions. Mathematical precedence is applied as follows: • Parenthesis (highest). • Unary minus and NOT logical operator. • Multiplication, division and modulus. • Addition and subtraction. • Greater than, less than, greater than or equal to, and less than or equal to relational operators. • Equal to and not equal to relational operators. • Bitwise AND, XOR, OR. • AND logical operator, OR logical operator (lowest).

Typical Examples

```
lift = height + rate/5.0
```

The Integer or Real point 'lift' is assigned the value calculated by the value of point 'rate' divided by 5, plus the value of point 'height'. Precedence can be changed by the introduction of parenthesis.

```
lift = lift - 0.2
```

The Integer or Real point 'lift' is assigned the value calculated by the current value of point 'lift' minus 0.2.

```
distance = distance * time
```

The Integer or Real point 'distance' is assigned the value calculated by the current value of point 'distance' multiplied by point 'time'.

References

Refer to chapter 4, Logic and Arithmetic for details of the use of arithmetic and logic functions. Refer to chapter 4, Punctuation for details of the use of parenthesis.

4-2 Logic and Arithmetic

4-2-1 Arithmetic Operators

Syntax

```
pointname = expression
```

Remarks

Argument	**Description**
pointname	The point name to be assigned a value based on an arithmetical expression.
expression	The value to be assigned to pointname. The expression may include the following operators with points and constants: • Addition '+'. • Subtraction '-'. • Multiplication '*'. • Division '/'. • Modulus 'Mod'. • Integer division '\'. • Exponential '^'.

Typical Examples

```
result = 60 + 20/5
```

The Integer or Real point 'result' is assigned the value calculated by the value of 20 divided by 5, plus 60.

```
lift = height + rate/5.0
```

The Integer or Real point 'lift' is assigned the value calculated by the value of point 'rate' divided by 5, plus the value of point 'height'. Precedence can be changed by the introduction of parenthesis.

References

Refer to chapter 4, Punctuation for details of the use of parenthesis.

4-2-2 Bitwise Operators

Syntax

```
pointname = expression
```

or

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
pointname	The pointname to be assigned a value based on the bitwise operation.
expression	The value to be assigned to pointname, or to be evaluated as a Boolean expression. The expression can include the following operators with points and constants: • Bitwise AND. • Bitwise OR. • Bitwise XOR.

Typical Examples

```
MSB = value AND 128
```

The Boolean point 'MSB' is set 'TRUE' if the binary representation of 'value' has the bit set which is worth 128, as demonstrated below.

	128	64	32	16	8	4	2	u
value =	1	0	1	0	1	0	1	0
128 =	1	0	0	0	0	0	0	0

If 'value' = 170 then 'MSB' will be set to TRUE.

```
status = status Or 32
```

The value of ‘status’ will be set to 42, as demonstrated below.

	128	64	32	16	8	4	2	u
status =	0	0	0	0	1	0	1	0
32 =	0	0	1	0	0	0	0	0

If 'status' is initially 10 then 'status' will be set to 42.

4-2-3 Logical Operators

Syntax

```
pointname = expression
```

or

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
Pointname	The point name to be assigned a value based on a logical expression.
Expression	The Boolean value to be assigned to pointname or the Boolean value forming a conditional statement. The expression includes the following operators with points and constants: • And 'AND'. • Or 'OR'. • Not 'NOT'. • Xor 'XOR'.

Typical Examples

```
flag = temp AND speed
```

The Boolean point 'flag' is assigned a value based on the logic of point 'temp' AND point 'speed'. If 'temp' and 'speed' are both not zero, 'flag' is set to 1, or ''TRUE''. A value of zero in either 'temp' or 'speed' supplies 'FALSE' or 0 to 'flag'.

```
IF flag AND temp AND speed THEN
    flag = FALSE
END IF
```

The Boolean point 'flag' is assigned 'FALSE', on the condition that 'flag' AND point 'temp' AND point 'speed' are all not zero. If the condition fails, then 'flag' is not assigned 'FALSE'.

References

Refer to chapter 4, Control Statements for details of the use of the IF THEN ELSE/ELSEIF END IF statements.

4-2-4 Relational Operators

Syntax

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
Expression	The value forming a conditional statement. The expression may include the following operators with points and constants: • Greater than '>'. • Less than '<'. • Greater than or equal to '>='. • Less than or equal to '<='. • Not equal to '<>'. • Equal to '='.

Typical Example

```
IF fuel < 0 THEN
    fuel = 0
END IF
```

The point 'fuel' is assigned the value 0 on the condition that currently, 'fuel' is less than 0. If 'fuel' is not less than 0, then it is not assigned the new value.

References

Refer to chapter 4, Control Statements for details of the use of the IF THEN ELSE/ELSEIF END IF statements.

4-3 Control Statements

4-3-1 Simple Conditional Statements

Syntax

```
IF condition THEN
    statementblock1
END IF
```

or

```
IF condition THEN
    statementblock1
ELSE
    statementblock2
END IF
```

Remarks

Argument	Description
Condition	The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string.
Statementblock1	One or more statements which are performed if the condition is met.
Statementblock2	One or more statements which are performed if the condition is not met.

Typical Examples

```
IF fuel < 0 THEN
    fuel = 0
END IF
```

Provided Integer point 'fuel' is less than 0, then it is assigned the value 0.

```
IF burner THEN
    fuel = fuel - rate
END IF
```

Provided Boolean point 'burner' is ''TRUE'', then Integer point 'fuel' is assigned a new value. It is also possible to apply 'IF burner = TRUE THEN' as the first line, with identical results.

```
IF distance > 630 AND distance < 660 AND lift >= -3
THEN
    winner = TRUE
    burner = FALSE
END IF
```

Provided that Integer point 'distance' is greater in value than 630 AND 'distance' is less in value than 660 (i.e. 'distance' is a value between 630 and 660) AND point 'lift' is greater than or equal to -3, then Boolean points 'winner' and 'burner' are assigned new values.

```
IF burner AND fuel > 0 AND rate > 0 THEN
    fuel = fuel - rate
ELSE
    lift = 0
    altitude = 0
END IF
```

Provided that Boolean point 'burner' is ''TRUE'' AND points 'fuel' and 'rate' are greater in value than 0, then 'fuel' is assigned a new value. Otherwise points 'lift' and 'altitude' are assigned a new value.

References

Refer to chapter 4, Punctuation, Indentation for details on the layout of code.

4-3-2 Nested Conditional Statements

Syntax

```
IF conditionA THEN
    statementblock1
    IF conditionB THEN
        statementblock3
    END IF
ELSE
    statementblock2
END IF
```

or

```
IF conditionA THEN
    statementblock1
    IF conditionB THEN
        statementblock3
    ELSE
        statementblock4
    END IF
ELSE
    statementblock2
END IF
```

or

```
IF conditionA THEN
    statementblock1
ELSEIF conditionB THEN
```

```
        statementblock3
    END IF
```

or

```
    IF conditionA THEN
        statementblock1
    ELSE
        statementblock2
        IF conditionB THEN
            statementblock3
        ELSE
            statementblock4
        END IF
    END IF
```

Remarks

Argument	Description
conditionA	The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string.
conditionB	This condition is nested in the first condition, either on a successful or unsuccessful evaluation of conditionA. The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string. There is no limit to the number of nested conditional statements.
statementblock1	One or more statements which are performed if conditionA is met.
statementblock2	One or more statements which are performed if conditionA is not met.
statementblock3	One or more statements which are performed if conditionB is met.
statementblock4	One or more statements which are performed if conditionB is not met.

Typical Examples

```
    IF burner AND fuel > 0 AND rate > 0 THEN
        lift = lift + rate/5
    ELSE
        count = 1
        IF altitude > 140 THEN
            lift = lift - 0.2
        END IF
    END IF
```

Provided a successful evaluation has been made to points 'burner' AND 'fuel' AND 'rate', point 'lift' is updated with the current value of rate divided by 5 plus 'lift'. Otherwise, a further evaluation is required on point 'altitude'. If 'altitude' is currently greater than 140, then 'lift' is decremented by 0.2.

```
    IF burner AND fuel > 0 AND rate > 0 THEN
        lift = lift + rate/5
    ELSE
```

```
        IF altitude > 140 THEN
            lift = lift - 0.2
        END IF
    END IF
    IF burner AND fuel > 0 AND rate > 0 THEN
        lift = lift + rate/5
    ELSEIF altitude > 140 THEN
        lift = lift - 0.2
    END IF
```

These two examples are identical. The use of the ELSEIF statement combines the ELSE statement and the IF/END IF statements for brevity. It is acceptable to have more than one ELSEIF statement in an IF THEN ELSE/ ELSEIF END IF construct.

References

Refer to chapter 4, Punctuation for details of the use of indentation.

4-3-3 Case Select

Syntax

```
SELECT CASE expression
    CASE expression
        statementblock1
    CASE expression
        statementblock2
    CASE expression
        statementblock3
END SELECT
```

or

```
SELECT CASE expression
    CASE expression
        statementblock1
    CASE expression
        statementblock2
    CASE ELSE
        statementblock3
END SELECT
```

Remarks

Argument	**Description**
expression	The expression may be a point, or a calculation of constants and/or points that produces a result.
statementblock1	One or more statements that are only performed if the preceding CASE expression is met.
statementblock2	One or more statements that are only performed if the preceding CASE expression is met.
statementblock3	One or more statements that are only performed if the preceding CASE expression is met.

Typical Examples

```
SELECT CASE colourvalue
    CASE 1
        Colour “PageName”, “Text_1”, “Blue”
    CASE 2
        Colour “PageName”, “Text_1”, “Green”
```

```
        CASE 3
            Colour “PageName”, “Text_1”, “Cyan”
        CASE ELSE
            Colour “PageName”, “Text_1”, &HFF000000
    END SELECT
```

This example shows the assignment of a colour according to the value of a point. The value of Integer point 'colourvalue' is evaluated and compared with each case until a match is found. When a match is found, the sequence of actions associated with the CASE statement is performed. When 'colourvalue' is 1, the colour given to the current object is blue, when 'colourvalue' is 2, the colour given to the current object is green, when 'colourvalue' is 3, the colour given to the current object is cyan. If 'colourvalue' falls outside the integer range 1-3, then the colour given is solid black. Like ELSE and ELSEIF, the CASE ELSE statement is optional.

```
    SELECT CASE TRUE
        CASE temperature > 0 AND temperature <= 10
            Colour “PageName”, “Text_1”, “Blue”
        CASE temperature > 10 AND temperature <= 20
            Colour “PageName”, “Text_1”, “Green”
        CASE temperature > 20 AND temperature <= 30
            Colour “PageName”, “Text_1”, “Red”
        CASE ELSE
            Colour “PageName”, “Text_1”, &HFFFFFFFF
    ENDSELECT
```

In this example, instead of using a point as the condition as with the previous example, the value is the condition - in this case Boolean state ''TRUE'' - with the integer point 'temperature' being tested at each case. If it is ''TRUE'' that 'temperature' is between 0 and 10, then the current object is set to blue, or if it is ''TRUE'' that 'temperature' is between 11 and 20, then the current object is set to green, or if it is ''TRUE'' that 'temperature' is between 21 and 30, then the current object is set to red. If none of these CASE statements are met, then the current object is set to white. Like ELSE and ELSEIF, the CASE ELSE statement is optional.

References

Refer to chapter 6, Object Commands for details of applying attributes to an object and for the use of the Colour object command. Refer to chapter 8, Colour Palette for details of the Colour Palette colour designation.

4-3-4 FOR... NEXT Loop

Syntax

```
    FOR pointname = startpt TO endpt STEP steppt
        statementblock1
    NEXT
```

Remarks

Argument	Description
pointname	The pointname to be used as the loop counter.
startpt	The initial setting of pointname, and the first value to be used through the loop.
endpt	The last value to be used. The loop ends when pointname exceeds this value.

Argument	Description
steppt	Amount to increase pointname by every pass of the loop. Steppt can be negative to count backwards providing startpt is larger than endpt. The STEP keyword and variable may be omitted in which case pointname is incremented at each pass of the loop (identical to adding STEP 1).

Typical Examples

```
FOR loopcount = 0 TO 100
    VerticalFill “PageName”, “Rectangle_1”, loopcount
NEXT
```

In this example, 'Ellipse_1' is gradually filled 100 times.

```
FOR loopcount = 100 TO 0 STEP -5
    VerticalFill “PageName”, “Rectangle_1”, loopcount
NEXT
```

In this example, the fill for 'Ellipse_1' is gradually removed 20 times (100 times/-5).

Note: Loop statements should be used with caution, as they consume processor time while they are running and some other parts of the system may not be updated.

4-3-5 DO WHILE/UNTIL Loop

Syntax

```
DO WHILE expression
    statementblock
LOOP
```

or

```
DO
    statementblock
LOOP WHILE expression
```

or

```
DO UNTIL expression
    statementblock
LOOP
```

or

```
DO
    statementblock
LOOP UNTIL expression
```

Remarks

Argument	Description
expression	The expression may be a point, or a calculation of constants and/or points that produces a result.
statementblock	One or more statements to be executed multiple times depending on expression.

Typical Example

```
DO WHILE dooropen = TRUE
    Message ("You must shut the door before
    continuing")
LOOP
```

```
DO
    nextchar = Mid (Mystring, position, 1)
    position = position + 1
LOOP UNTIL nextchar = "A"
```

Note: Loop statements should be used with caution, as they consume processor time while they are running and some other parts of the system may not be updated.

4-4 Subroutines

4-4-1 Call

Syntax

```
CALL subroutine (arguments)
```

Remarks

Argument	Description
subroutine	The name of the subroutine defined at project level.
arguments	The list of arguments required by the subroutine separated by commas. Each argument may be a pointname, constant, arithmetical or logical expression or any valid combination.

Typical Example

```
CALL MySub ($Second, "Default", 2 + Int1)
```

4-5 Punctuation

4-5-1 Quotation Marks / Command String Delimiters

Typical Examples

```
name = "Valve position"
```

The Text point 'name' is assigned associated text, contained within quotation marks. Quotation marks must be used in this instance.

```
Message("This text to be displayed as a message.")
```

Passing static text as arguments to functions.

```
BlueCarsAck = IsAlarmAcknowledged("BLUEPAINT")
```

The point 'BlueCarsAck' is assigned a Boolean state based on the alarm 'BLUEPAINT'. Quotation marks must be used for an alarm name.

It is possible to embed double quote characters inside a string by using 2 double quotes characters.

Syntax

```
“Some ““string”” text”
```

Typical Example

```
Message(“Error: ““Invalid Function”” occurred”)
```

4-5-2 Indentation

Typical Examples

```
IF burner AND fuel > 0 AND rate > 0 THEN
lift = lift + rate/5
ELSE
```

```
IF altitude > 140 THEN
lift = lift - 0.2
END IF
END IF

IF burner AND fuel > 0 AND rate > 0 THEN
    lift = lift + rate/5
ELSE
    IF altitude > 140 THEN
        lift = lift - 0.2
    END IF
END IF
```

Both examples provide identical functionality, but the use of indentation, either spaces or tabs to show the construction of the statements aids readability.

The use of the ELSEIF statement in this example was omitted for clarity.

4-5-3 Multiple Commands

Typical Examples

```
count = 75
result = log(count)
count = 75 : result = log(count)
```

Both examples provide identical functionality, but the use of the colon between statements allows both to reside on the same line.

4-5-4 Parenthesis

Typical Examples

```
result = 20 + 30 * 40
```

The result is 1220.

```
result = (20 + 30) * 40
```

The values in parenthesis are calculated first. The result is 2000.

References

Refer to chapter 4, Logic and Arithmetic, Arithmetric Operations for further details.

4-5-5 Remarks

Syntax

```
REM | rem comment
```

or

```
'comment
```

Remarks

Argument	Type	Description
Comment	---	Descriptive text.

Typical Examples

```
REM The following statement adds two numbers
result = 45 + 754
result = 45 + 754  'add two numbers
```

4-6 Indirection within Script Commands and Expressions

It is possible to use text points directly or indirectly in place of literal string arguments within scripts and expressions. For instance, each of the following commands has the same effect:

- Using a string literal;

```
PlayOLE("ole_1", 0)
```

- Using a textpoint directly;

```
textpoint = "ole_1"
PlayOLE(textpoint, 0)
```

- Using a textpoint indirectly via the 'ind' function.

```
text = "ole_1"
textpoint = "text"
PlayOLE(ind(textpoint), 0)
```

It is possible to use text points indirectly in place of point name arguments within script commands. For instance, each of the following commands has the same effect:

- Using a point name directly;

```
verbnumber = 0
PlayOLE("ole_1", verbnumber)
```

- Using a textpoint indirectly via the 'ind' function.

```
verbnumber = 0
textpoint = "verbnumber"
PlayOLE("ole_1", ind(textpoint))
```

An example using Indirection

The value of point indirection can be seen in a situation where it is necessary to dynamically change the pointname that an object is linked to. In the following example a toggle button is configured to control the Boolean state of one of four points:

- The four Boolean points to be controlled are called 'motor1', 'motor2', 'motor3' and 'motor4'.
- The text point 'textpoint' is used to store the name of the Boolean point to be controlled.
- The text point 'text' is used to store the string value of the integer point 'index'
- The integer point 'index' (which has a range 1-4) is used to dynamically change the point being controlled.
- Access to any of the four Boolean points 'motor1', 'motor2', 'motor3', 'motor4' can be achieved by applying indirection to 'textpoint' using the 'ind' function and changing the contents of 'textpoint'.

For instance, in order to dynamically change the Boolean point a toggle button is linked to follow these steps.

1, 2, 3…

1. Link the toggle button to a textpoint using indirection e.g. ind(textpoint).
2. Link the following script code to run as required. e.g. on clicking a button.
   - Text = ValueToText(index)
   - TextPoint = "motor" + text
3. The ValueToText function converts the integer value of the point 'index' into a string held in the textpoint 'text'. Therefore the point 'text' contains either '1', '2', '3' or '4'. The expression 'motor' + text appends the contents of the point 'text' to the literal string 'motor'. Therefore 'textpoint' contains

either 'motor1', 'motor2', 'motor3' or 'motor4' dependent on the value of 'index'. Change the value of the 'index' to determine which Boolean point to control. e.g. via the Edit Point Value (Analogue) animation.

4-7 Point Arrays within Script Commands and Expressions

It is possible to access the elements of a point array directly from within scripts or expressions.

- Setting the value of an array point directly;

```
arraypoint(2) = 30
```

- Getting the value of an array point directly;

```
value = arraypoint(2)
```

Note: The examples shown are based on VBScript. Expressions are currently based on CX-Supervisor Script and will therefore have a different syntax. For example, ‘arraypoint(2) = 30’ should be ‘arraypoint[2] = 30’ when used in an expression.

An example using Point Arrays

The value of array points can be seen in a situation where it is necessary to dynamically change the pointname that an object is linked to. In the following example a toggle button is configured to control the Boolean state of one of four elements of an array point.

The Boolean array point 'motor' is configured to contain 4 elements.

The integer point 'index' (which has a range 0-3) is used to dynamically change the element of the point being controlled.

In order to dynamically change the element of a Boolean point that a toggle button is linked to follow these steps.

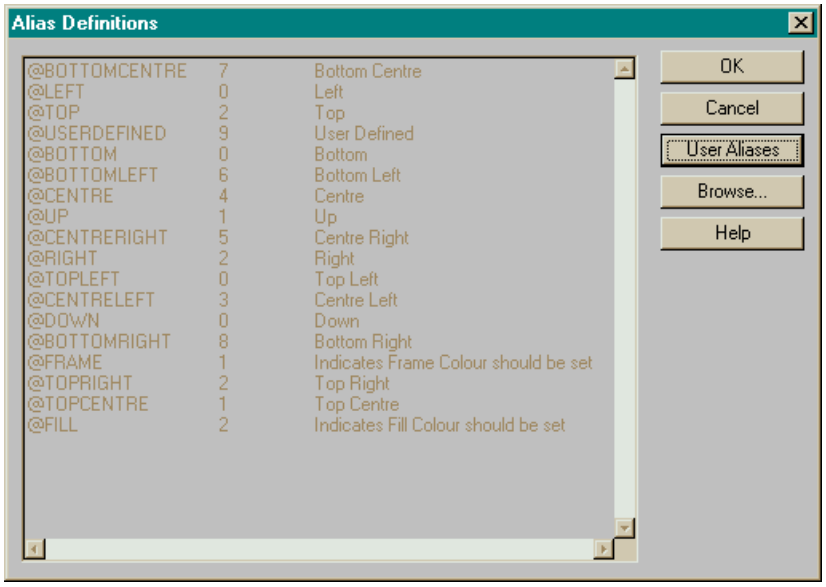
1, 2, 3…

1. Link the toggle button to an array point. e.g. 'motor(index)'.
2. Change the value of the 'index' to determine which element of the Boolean point to control. e.g. via the Edit Point Value (Analogue) animation.

4-8 Using Aliases

This facility is used to declare an alias - that is, to define a text string that can be used in place of another text string or a number within any script or expression. The Alias Definitions dialog is displayed by selecting the "Alias Definition..." option from the Project menu. It can also be displayed if "Aliases..." is selected from the script editor. The dialog displays either the User defined aliases or the preset System aliases and is toggled between these two displays by pressing the User/System Alias button.

The following illustration shows the Alias Definitions dialog displaying a number of User defined aliases. The System aliases are pre-defined and can not be edited or added to.



Syntax:

```
@AliasNameAlias definition 'optional comment
```

Remarks:

Argument	Type	Description
@AliasName	string	The string name of the alias
Alias definition	string	This is a string representing the actual text or expression of the expanded alias.
' comment	string	This is an optional comment.

The @ symbol at the beginning of each line initiates each alias command. For example, the text string @SomePoint could be used to represent any sequence of characters in a script - e.g. it could be defined as:

```
@SomePoint = InArray(1)
```

or even

```
@SomePoint = Inarray(1) + Inarray(2) / 2
```

This is an easy way of identifying the individual members of array points. It can also be used to associate names with numbers, for example,

```
@SecondsPerDay = 86400
```

Alias definitions are stored in a simple text file in the project directory, called <project name>.pre. The format of the file consists of any number of lines such as:

```
@Test1 = InArray(12) * 10
```

i.e. an @ symbol followed by the name of the alias, then an equals sign (or space), followed by the definition of the alias. Anything that follows the last apostrophe (') symbol on a line is interpreted as a comment. Any line which does not start with the @ symbol is also assumed to be a comment.

Typical Examples

```
Declare boiler temperatures
@BoilerTemp1 = InArray(0) ' for boiler room 1
@BoilerTemp2 = InArray(1) ' for boiler room 2
@SecondsPerMinute  = 60   ' sets duration
```

Aliases may also be used to create a complicated expression such as

```
@HYPOTENUSEsqrt(Opposite * Opposite + Adjacent *
Adjacent) 'Calculates length of Hypotenuse
```

This can be used in a script in the following way:

```
Opposite = 8.45
Adjacent = 9.756
length = @HYPOTENUSE
```

where Opposite, Adjacent and length are all REAL points.

Note: Changing an alias definition after it has been used in an expression or script will not automatically change the result in the script. The appropriate script or expression where that alias is used must be accessed and recompiled by pressing the OK button in order to apply the changes or performing the Rebuild All command from the Project menu.

SECTION 5
Functions and Methods

This chapter describes the Functions and Methods available to the scripting language. The following table shows all Functions/Methods available. Refer to the relevant sub-section for full details about a Function or Method.

Function Name	Function Type	Type	Remarks
AcknowledgeAlarm	alarm command	Scr	Acknowledges an alarm.
AcknowledgeAllAlarms	alarm command	Scr	Acknowledges all alarms.
AcknowledgeLatestAlarm	alarm command	Scr	Acknowledge the latest alarm.
Acos	unary function	All	Applies unary expression.
Asin	unary function	All	Applies unary expression.
Atn	unary function	All	Applies unary expression.
AuditPoint	Data Logging command	Scr	Logs a point value into the CFR database.
CancelForce	point command	Scr	Removes the forcing of values on a point.
ChangeUserPassword	Data Logging command	Scr	Changes a user's Windows password.
Chr	text command	All	Displays a character based on the ASCII character set.
ClearAlarmHistory	alarm command	All	Clears the alarm history.
ClearErrorLog	event/error commands	All	Clears the error log.
ClearLogFile	Data Logging command	Scr	Clears a data log file
ClearSpoolQueue	printer command	All	Discards any queued messages or alarms.
close	object command	Scr	Closes a specified page.
CloseAlarmHistory	alarm command	All	Closes the current alarm history.
CloseAlarmStatus	alarm command	Scr	Closes the current alarm status.
CloseComponent	comms command	All	Closes a component for a PLC (e.g. CX-Server components).
CloseErrorLog	error command	Scr	Closes the currently open Error Log.
CloseFile	file command	Scr	Closes the open file.
CloseLogFile	Data Logging command	Scr	Closes a data log file
CloseLogView	Data Logging command	Scr	Closes the log viewer
ClosePLC	PLC command	Scr	Close communications with a PLC.
colour	object command	OP	Specifies a colour to an object.
CopyArray	point command	All	Copies the content of an array.
CopyFile	file command	Scr	Copies a specified file.

Function Name	Function Type	Type	Remarks
cos	unary function	All	Applies unary expression.
dec	functiion	All	Reduces the value of a point by the specified amount.
DeleteFile	file command	Scr	Deletes the specified file.
disable	object command	OP	Disables an object.
DisableGroup	point command	All	Prevents a group of points to be read or written.
DisablePoint	point command	Scr	Disables communications to a point.
display	object command	Scr	Displays a specified page.
DisplayAlarmHistory	alarm command	Scr	Displays the current alarm history.
DisplayAlarmStatus	alarm command	Scr	Displays the alarm status of all current alarms.
DisplayErrorLog	event command	Scr	Displays the current Error Log.
DisplayPicture	general command	Scr	Reload an image for a picture object
DisplayRecipes	recipe command	Scr	View the current recipes in the project.
DownloadPLCProgram	PLC command	All	Downloads specified files to the PLC.
DownloadRecipe	recipe command	Scr	Downloads a specified recipe.
EditFile	file command	All	Edits a specified file.
EmailReport	report command	All	Produces a report based on a report template and emails it to the specified recipients.
EmailText	text command	All	Emails a message to the specified recipients.
EnableAlarms	alarm command	All	Enables alarm functions.
EnableErrorLogging	error command	Scr	All actions become subject to Error Logging.
EnableGroup	point command	All	Permits a group of points to be read or written.
EnableOLE	comms command	Scr	Allows use of OLE functions.
EnablePLC	comms command	Scr	Allows use of PLC functions.
EnablePoint	point command	Scr	Enables communications to a point.
EnablePrinting	printer command	All	Permits printing of Alarms or messages.
ExportAndViewLog	Data Logging command	Scr	Exports data log and views
ExportLog	Data Logging command	Scr	Exports data log
FileExists	file command	All	Specifies the existence of a file.

Function Name	Function Type	Type	Remarks
Force	point command	Scr	Locks the value of a point.
ForceReset	point command	Scr	Sets a point value to 0.
ForceSet	point command	Scr	Sets a point value to 1.
FormatText	text command	All	Inserts text with standard 'C' formatting characters.
GenerateReport	report command	All	Produces a report based on a report template.
GetBit	point command	All	Retrieves a bit from a point.
GetPerformanceInfo	general command	All	Retrieves internal performance and diagnostic values.
GetPLCMode	PLC command	All	Retrieves the mode of a PLC.
GetTextLength	text command	All	Specifies the number of characters in a text point.
height	object command	OP	Specifies the height of an object.
horizontal%fill	object command	OP	Specifies the horizontal fill of an object.
inc	functiion	All	Increases the value of a point by the specified amount.
ind	functiion	All	Accesses a point from indirect reference via its point name.
InputPoint	point command	Scr	Reads a value from a point.
IsAlarmAcknowledged	alarm command	Scr	Tests if a specified alarm has been acknowledged.
IsAlarmActive	alarm command	Scr	Tests if a specified alarm is currently active.
LaunchTroubleshooter	comms command	Scr	Launches SGW tool for troubleshooting controllers
Left	statement	Scr	Extracts characters from the left of a string
lsb	unary function	All	Returns the least significant bit set in an integer value.
log	unary function	All	Calculates the natural logarithm on a number.
log10	unary function	All	Calculates the base-10 logarithm on a number.
LogError	error command	Scr	Logs an error message with the error logger.
LogEvent	error command	Scr	Logs an event message with the error logger.
Login	security command	Scr	Logs a user into a run-time application.
Logout	security command	Scr	Logs a user out of a run-time application.
Message	text command	Scr	Outputs a string in a message box.

Function Name	Function Type	Type	Remarks
Mid	text command	Scr	Extracts a substring from a string.
move	object command	OP	Moves an object.
MoveFile	file command	Scr	Moves the specified file.
msb	unary function	All	Returns the most significant bit set in an integer value.
OpenComponent	comms command	All	Opens a component for a PLC (e.g. CX-Server components).
OpenFile	file command	Scr	Opens the specified file.
OpenLogFile	Data Logging command	Scr	Opens a data log file
OpenLogView	Data Logging command	Scr	Opens the Data Log Viewer
OpenPLC	PLC command	Scr	Opens communications with a PLC.
OutputPoint	point command	Scr	Displays the current value of a point.
PlayOLE	gen. command	Scr	Plays an OLE object.
PlaySound	gen. command	Scr	Plays a sound file.
PLCCommsFailed	PLC command	All	Specifies if the PLC communications have failed.
PLCMonitor	PLC command	Scr	Monitors a PLC.
PointExists	point command	All	Specifies the existence of a point.
PrintActivePage	gen. command	Scr	Prints the currently active page.
PrintFile	file command	Scr	Prints the specified file.
PrintMessage	text command	All	Prints messages to the configured 'Alarm/message printer'.
PrintPage	gen. command	Scr	Prints the specified page.
PrintReport	report command	All	Prints a report
PrintScreen	gen. command	Scr	Prints the current display screen.
PrintSpoolQueue	printer command	All	Prints all queued alarms or messages.
Rand	gen. command	Scr	Calculates a random number.
Read	file command	Scr	Reads data from an open file into a point.
ReadMessage	file command	All	Reads text from an external file.
Right	text command	Scr	Extracts characters from the right of a string.
rotate	object command	OP	Rotates an object.
RunApplication	gen. command	Scr	Runs the specified application.
RunHelp	gen. command	Scr	Runs the specified help file.
SelectFile	file command	All	Specifies a file name and path.
SetBit	point command	All	Sets a specific bit from a point.

Function Name	Function Type	Type	Remarks
SetNYLED	gen. command	Scr	Sets the hardware LEDs on NY IPC
SetPLCMode	PLC command	All	Sets the mode of a PLC.
SetPLCPhoneNumber	PLC command	All	Sets a phone number to a PLC.
SetupUsers	security command	Scr	Defines users and passwords for Login.
shl	functiion	All	Performs a bitwise shift left of an integer value
shr	function	All	Performs a bitwise shift right of an integer value
ShutDown	gen. command	Scr	Terminates CX-Supervisor.
sin	unary function	All	Applies unary expression.
sqr	unary function	All	Applies unary expression.
StartAuditTrail	Data Logging command	Scr	Starts Audit Trail logging.
StopAuditTrail	Data Logging command	Scr	Stops Audit Trail logging.
StartLogging	Data Logging command	Scr	Starts a data set logging.
StopLogging	Data Logging command	Scr	Stops a data set logging.
tan	unary function	All	Applies unary expression.
TCAutoTune	temp. controller command	All	Starts or stops a temperature controller auto-tune operation.
TCBackupMode	temp. controller command	All	Defines how a temperature controller stores internal variables.
TCGetStatusParameter	temp. controller command	All	Retrieves the temperature controller status parameter.
TCRemoteLocal	temp. controller command	All	Defines the operational mode of a temperature controller.
TCRequestStatus	temp. controller command	All	Retrieves the temperature controller status.
TCReset	temp. controller command	All	Resets the temperature controller.
TCRspLsp	temp. controller command	All	Defines the setpoint mode used by the temperature controller.
TCRunStop	temp. controller command	All	Defines either auto-output mode shift or manual output mode shift.
TCSaveData	temp. controller command	All	Saves data associated with the temperature controller.
TCSettingLevel1	temp. controller command	All	Performs a settinglevel function for the temperature controller.
TextToValue	text command	Scr	Converts a string to a numerical point value.

Function Name	Function Type	Type	Remarks
ToBool	function	VB	Converts a numeric value into a Boolean value.
ToInt	function	VB	Converts a Boolean value into a numeric value.
UploadPLCProgram	PLC command	All	Uploads programs in the PLC to specified files.
ValueToText	text command	Scr	Converts a numerical value into a text point.
vertical%fill	object command	OP	Specifies the vertical fill of an object.
ViewReport	report command	All	Displays a report
visible	object command	OP	Toggles the visibility of an object.
width	object command	OP	Specifies the width of an object.
Write	file command	Scr	Writes a value to an open file.
WriteMessage	file command	All	Writes text to an external file.

The 'Type' column refers to the types of script and expression the function can be applied to. 'All' refers to expressions and both script languages (VBScript and CX-Supervisor Script). 'Scr' refers to scripts only (both languages). 'OP' refers to Object and Page scripts only. ‘VB’ refers to ‘VBScript’ only.

5-1 Object Commands

Object commands can be used to control native CX-Supervisor graphical objects, like rectangles or lines.

Note: Using the ‘Colour’ command as an example, the CX-Supervisor Script and VB Script syntax is as follows:

CX-Supervisor Script: **rect.colour = Red**

VB Script: **Colour “page1”, “rect”, “Red”**

5-1-1 Current Object

Syntax

```
objectcommand
```

Remarks

Argument	Description
	"The expression can be made up of the following commands, which are also described in chapter 6, Object Commands: • Colour command. • Disable command. • Visible command. • Move command. • Rotate command. • Vertical fill command. • Horizontal fill command. • Height command. • Width command. The content of the commands are made up of arithmetical or logical expressions, x and y co-ordinates, or references, varying between commands. The colour command requires a colour identifier.

Typical Example

```
colour (red)
```

The current object is specified as red in colour.

References

Refer to:

- Chapter 6, Blink for use of the blink command.
- Chapter 6, Colour for use of the colour command.
- Chapter 6, Disable for use of the disable command.
- Chapter 6, Height for use of the height command.
- Chapter 6, Horizontal Fill for use of the horizontal fill command.
- Chapter 6, Move for use of the move command.
- Chapter 6, Rotate for use of the rotate command.
- Chapter 6,Vertical Fill for use of the vertical fill command.
- Chapter 6, Visible for use of the visible command.
- Chapter 6, Width for use of the width command.

5-1-2 Other Objects

Syntax

```
objectname.objectcommand
pagename.objectname.objectcommand
```

Remarks

Argument	Description
objectname	This is the name of the object. The object is provided with a generic name on creation, which can be amended later to something more meaningful. The script is automatically updated following any amendment to the object name.

Argument	Description
objectcommand	This can be made up of the following commands, which are described in chapter 6, Object Commends: • Blink command • Colour command. • Disable command. • Visible command. • Move command. • Rotate command. • Vertical fill command. • Horizontal fill command. • Height command. • Width command. The content of the commands are made up arithmetical or logical expressions, x and y co-ordinates, or references, varying between commands. The colour command requires a colour identifier.

Typical ‘CX-Supervisor Script‘ Examples

```
POLYGON_1.colour (red)
POLYGON_1.colour = red
```

The specified object, 'POLYGON_1' is set to be red in colour.

References

Refer to:

- CX-Supervisor User Manual for details of object names.
- Chapter 6, Blink for use of the blink command.
- Chapter 6, Colour for use of the colour command.
- Chapter 6, Disable for use of the disable command.
- Chapter 6, Height for use of the height command.
- Chapter 6, Horizontal Fill for use of the horizontal fill command.
- Chapter 6, Move for use of the move command.
- Chapter 6, Rotate for use of the rotate command.
- Chapter 6, Vertical Fill for use of the vertical fill command.
- Chapter 6, Visible for use of the visible command.
- Chapter 6, Width for use of the width command.

5-1-3 Blink

Syntax (CX-Supervisor Script)

```
objectname.blink (colourID, status)
```

Syntax (VB Script)

```
Blink “pagename”, “objectname”, colourID, status
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.

Argument	Description
colourID	The colourID may be an Integer point, or a calculation of constants and/or points that produce a 32-bit Integer value. This is the desired colour's RGB value and transparency (Alpha channel). (format: 0xAARRGGBB). NOTE: for backward compatibility with old projects the AA field must not use values of 00, 01 and 02, as these are reserved. An AA field value of 00 will be treated as a BGR value (format: 0x00BBGGRR). In this case the colour will always be opaque. Colours can also be expressed by their name. Refer to section 8 for details. Alternatively, an integer value of 0x1000000 can be added to a number 0-65 to select a palette entry.
status	This argument may be omitted. May be one of: TRUE - turn blinking On. FALSE - turn blinking Off. If omitted, TRUE is assumed.

Typical 'CX-Supervisor Script' Examples

```
blink (red, TRUE)
```

Start blinking red.

```
LINE_1.blink(0xFFFF00, status)
```

The object LINE_1 starts or stops blinking yellow depending on value of Boolean point 'status'.

5-1-4 Colour

Syntax (CX-Supervisor Script)

```
objectname.colour (colourID, context)
colour (colourID, context)

An equals sign may be used as an alternative to
brackets:

objectname.colour = colourID
colour = colourID
```

Either spelling 'colour' or 'color' is acceptable.

Note: An equals sign may also be used for most other object commands, even if it is not directly specified in this manual.

Syntax (VB Script)

```
Colour "pagename", "objectname", colourID, context
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.

Argument	Description
colourID	The colourID may be an Integer point, or a calculation of constants and/or points that produce a 32-bit Integer value. This is the desired colour's RGB value and transparency (Alpha channel). (format: 0xAARRGGBB). NOTE: for backward compatibility with old projects the AA field must not use values of 00, 01 and 02, as these are reserved. An AA field value of 00 will be treated as a BGR value (format: 0x00BBGGRR). In this case the colour will always be opaque. Colours can also be expressed by their name. Refer to section 8 for details. Alternatively, an integer value of 0x1000000 can be added to a number 0-65 to select a palette entry.
context	This argument is optional an may be omitted. It defines which part of the object has it's colour changed. May be one or more of: @FILL - change fill colour @FRAME - changes frame colour If omitted both are changed. Equivalent to @FILL | @FRAME

Typical 'CX-Supervisor Script' Examples

```
TEXT_3.colour (blue)
```

or

```
TEXT_3.colour = blue
```

The object 'TEXT_3' is set to blue.

```
BALL.colour (35 + 0x1000000)
```

The object 'BALL' is set to colour 35 from the colour palette.

```
BALL.colour (0xFF0000,@FILL)
```

The object 'BALL' is set to blue.

```
shade = tint1 + tint2
IF shade > 65 OR shade < 0 THEN
    shade = 0
END IF
ELLIPSE_1.colour (shade + 0x1000000)
```

The point 'shade' is set to a value based on 'tint1' and 'tint2', and is tested first to ensure that it is a value between 0 and 65. If 'shade' falls outside this range, then it cannot be applied as a colour to an object, and is therefore reset to 0 (or black). ELLIPSE_1' is set to the palette colour of the value of shade.

References

Refer to chapter 6, Colour Palette for details of colour names and colour numbers.

5-1-5 Disable

Syntax (CX-Supervisor Script)

```
objectname.disable (expression)
```

Syntax (VB Script)

```
Disable "pagename", "objectname", expression
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
expression	The expression can be made up of points resulting in 'TRUE' or 'FALSE'.

Typical 'CX-Supervisor Script' Examples

```
disable (TRUE)
```

The current pushbutton object to which this example applies is disabled.

```
PUSH_8.disable (count AND flag)
```

The selectable object 'PUSH_8' is disabled provided Integer point 'count' AND Boolean point 'flag' return ''TRUE''.

5-1-6 Height

Syntax (CX-Supervisor Script)

```
objectname.height (expression, context)
objectname.height = expression
```

Syntax (VB Script)

```
Height "pagename", "objectname", expression, context
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
expression	This is a value, point or an arithmetic expression returning a new height value in pixels.
context	This argument is optional and may be omitted. It defines which part of the object is the datum, and remains static. May be one of: @TOP - uses object top as datum. @CENTRE - uses object centre as datum @BOTTOM - uses object bottom as datum If omitted @CENTRE is assumed

Typical 'CX-Supervisor Script' Examples

```
height (100)
```

or

```
height = 100
```

The height of the current object is set to 100.

```
LINE_1.height (stretch/offset, @top)
```

The height of object 'LINE_1' is changed to the value calculated by points 'stretch' and 'offset', keeping the top where it is.

5-1-7 Horizontal Fill

Syntax (CX-Supervisor Script)

```
objectname.horizontal%fill (expression, context)
```

Syntax (VB Script)

```
Horizontal%Fill "pagename", "objectname", expression, context
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
expression	This is an arithmetic expression that must return a value between 0 and 100. On return of a valid result, the fill commences from left to right.
context	This argument is optional and may be omitted. It defines which side of the object is filled from. May be one of: @LEFT - fill from the left @RIGHT - fill from the right If omitted, @LEFT is assumed

Typical 'CX-Supervisor Script' Examples

```
horizontal%fill (50)
```

The current object to which this example applies is filled by 50%.

```
ELLIPSE_1.horizontal%fill (GAS_LEVEL, @RIGHT)
```

The object 'ELLIPSE_1' is filled from the right, provided the point 'GAS_LEVEL' returns a valid result, between 0 and 100.

5-1-8 Move

Syntax (CX-Supervisor Script)

```
objectname.move (x co-ordinate, y co-ordinate)
```

Syntax (VB Script)

```
Move "pagename", "objectname", x co-ordinate, y co-ordinate
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
x co-ordinate y co-ordinate	The x and y co-ordinates of the origin of the object at its resultant position in pixels are specified in the form (x, y). Points alone or as part of an arithmetic expression may be used as a basis for this expression.

Typical 'CX-Supervisor Script' Examples

```
move (100, 200)
```

The current object to which this example applies is moved to the specified position.

```
POLYGON_1.move (xpos, ypos/5)
```

The object 'POLYGON_1' is moved to the position specified by points 'xpos' and 'ypos' divided by 5.

5-1-9 Rotate

Syntax (CX-Supervisor Script)

```
objectname.rotate (angle, context, fixed, xcoord, ycoord)
```

Syntax (VB Script)

```
Rotate “pagename”, “objectname”, angle, context, fixed, xcoord, ycoord
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
angle	The angle of rotation can range between 0 to 360 in a clockwise direction. Points alone, or as part of an arithmetic expression may be used as an angle.
context	This argument is not required and may be omitted. May be one of: @TOPLEFT - rotate around top left of object @TOPCENTRE -rotate around top centre of object @TOPRIGHT - rotate around top right of object @CENTRELEFT - rotate around centre left of object @CENTRE - rotate around centre of object @CENTRERIGHT - rotate around centre right of object @BOTTOMLEFT - rotate around bottom left of object @BOTTEMCENTRE - rotate around bottom centre of object @ BOTTOMRIGHT - rotate around bottom right of object @USERDEFINED - user defined point specified in xcoord and ycoord.
fixed	This argument may be omitted. If this boolean value is true, the rotation origin is fixed to the screen, even if the object is moved. Otherwise, the rotation origin is relative to object position.
xcoord ycoord	Only required if @USERDEFINED is specified. These integer variables specify the rotation origin in pixels

Typical ‘CX-Supervisor Script‘ Examples

```
rotate (45)
```

The current object to which this example applies is rotated by 45 .

```
RECTANGLE_1.rotate(tilt, @USERDEFINED, 0, -100, 10)
```

The object 'RECTANGLE_1' is rotated by the value of 'tilt', about a point -100, 10 relative to the objects current position.

```
rotate (a * sin(b))
```

The current object is rotated based on the result of an arithmetic expression involving points named 'a and 'b'.

5-1-10 Vertical Fill

Syntax (CX-Supervisor Script)

```
objectname.vertical%fill (expression, context)
```

Syntax (VB Script)

```
VerticalFill “pagename”, “objectname”, expression, context
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
expression	This is an arithmetic expression that must return a value between 0 and 100. On return of a valid result, the fill commences from bottom to top.
context	This argument may be omitted. May be one of: @DOWN - Fill object downwards @UP - Fill object upwards If omitted, @UP is assumed

Typical ‘CX-Supervisor Script‘ Examples

```
vertical%fill (50)
```

The current object to which this example applies is filled by 50%.

```
ELLIPSE_1.vertical%fill (OIL_QUANTITY, @DOWN)
```

The object 'ELLIPSE_1' is filled provided the point 'OIL QUANTITY' returns a valid result, between 0 and 100.

5-1-11 Visible

Syntax (CX-Supervisor Script)

```
objectname.visible (expression)
```

Syntax (VB Script)

```
Visible “pagename”, “objectname”, expression
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
expression	The expression can be made up of points resulting in 'TRUE' or 'FALSE'.

Typical ‘CX-Supervisor Script‘ Examples

```
visible (TRUE)
```

The current object to which this example applies becomes visible.

```
POLYLINE_8.visible (count AND flag)
```

The object 'POLYLINE_8' is made visible provided Integer point 'count' AND Boolean point 'flag' return ''TRUE''.

5-1-12 Width

Syntax (CX-Supervisor Script)

```
objectname.width (expression, context)
```

Syntax (VB Script)

```
Width “pagename”, “objectname”, expression, context
```

Remarks

Argument	Description
pagename	This is the name of the page.
objectname	This is the name of the object. Where a script is directly attached to an object, objectname is not required.
expression	This is a value, point or an arithmetic expression returning a new width value in pixels.
context	This argument may be omitted. May be one of: @LEFT - use left of object as datum. @CENTRE - use centre of object as datum. @RIGHT - use right of object as datum. If omitted, @CENTRE is assumed.

Typical ‘CX-Supervisor Script‘ Examples

```
width (150)
```

The width of the current object is set to 150.

```
LINE_1.width (squeeze/offset, @RIGHT)
```

The width of object 'LINE_1' is changed to the value calculated by points 'squeeze' and 'offset', keeping the rightmost point fixed.

5-2 Page Commands

Display Page

Syntax

```
display ("pagename")
```

or

```
display ("pagename", X, Y)
```

Remarks

Argument	Description
pagename	This is the name of the page for display, based on its filename without the file extension, e.g. the pagename for CAR.PAG is simply 'CAR'.

Typical Examples

```
display ("CAR")
```

The page 'CAR.PAG' is displayed.

```
textpoint = "CAR"
display(textpoint)
```

The page 'CAR.PAG' is displayed.

```
display("CAR", 100, 200)
```

The page 'CAR.PAG' is displayed in a custom position, 100 pixels across from the left of the main window and 200 pixels down from the top.

5-2-1 Close Page

Syntax

```
close ("pagename")
```

Remarks

Argument	Description
pagename	This is the name of the page for closure, based on its filename without the file extension, e.g. the pagename for CAR.PAG is simply 'CAR'. The pagename for closure must be currently open.

Note: The 'close' operation will cause the page to be unloaded, including all objects, ActiveX controls and scripts. Care must be taken not to attempt to access them after the close instruction.

Note: Where the script containing the 'close' instruction is on the page to be closed, this should be the last instruction in the script as it will cause the script to be unloaded.

Typical Examples

```
close("CAR")
```

The page 'CAR.PAG' is closed.

```
textpoint = "CAR"
close(textpoint)
```

The page 'CAR.PAG' is closed.

5-3 General Commands

5-3-1 Exponential

Description

Mathematical function to calculate a value raised to a power.

Syntax

```
result = Exp (value, exponent)
```

Remarks

Argument	Type	Description
result	integer	Point name to receive returned result of value raised to the power of exponent.
value	integer	Number to raise.
exponent	integer	Power to raise value by.

Typical Example

```
MSBMask = Exp (2, 15)
```

In this example, 'MSBMask' is assigned the value 215, i.e. 32,768.

5-3-2 PlayOLE

Description

Initiate an OLE verb or 'method' on an OLE 2 object. The verb number is object dependent so refer to the object's documentation. This function is now largely obsolete as most objects are nowadays ActiveX objects.

Syntax

```
returnstate = PlayOLE("objectname",OLEVerbNumber)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
objectname	string	The identifier of the OLE object to be played.
OLEVerbNumber	integer	The verb number has a specific meaning to the OLE application. Typical values are: 0: specifies the action that occurs when an end-user double clicks the object in its container. The object determines this action (often 'edit' or 'play'). -1: instructs the object to show itself for editing or viewing. Usually an alias for some other object-defined verb. -2: instructs an object to open itself for editing in a window separate from that of its container. -3: causes an object to remove its user interface from the view. Applies only to objects that are activated in-place. Positive numbers designate object specific verbs.

Typical Example

```
PlayOLE("ole_1",0)
```

The object 'ole_1' is played using its primary verb.

5-3-3 DisplayPicture

Description

Reload a picture for a Picture object.

Syntax

```
returnstate = DisplayPicture("objectname", filename)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
objectname	string	The identifier of the bitmap object with a to be loaded and displayed
filename	string	The filename of the bitmap to be displayed. This can be a constant (inside quotes) or a text point.

Typical Example

```
DisplayPicture("Bitmap_1","C:\Application\Floorplan1.
bmp")
```

The object "Bitmap_1" will load and display the Floorplan1 bitmap.

```
DisplayPicture("Bitmap_2", txtFileName)
```

The object "Bitmap_2" will load and display the file name stored in txtFileName text point.

5-3-4 PlaySound

Description

Plays a Windows .WAV sound file using the standard Windows sound channel and Sound Card driver.

Syntax

```
returnstate = PlaySound("soundfile")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
soundfile	string	Path of sound file to be played.

Typical Example

PlaySound("c:\noise.wav")

The soundfile "c:\noise.wav" is played.

5-3-5 Rand

Description

Returns a random integer, between 0 and the specified limit.

Syntax

```
pointname = Rand(upperlimit)
```

Remarks

Argument	Type	Description
upperlimit	integer	The maximum negative or positive integer value that the Rand function can generate.
pointname	Integer point	Point that contains the integer returned from the Rand function.

Typical Example

```
randomnumber = Rand(upperlimit)
```

A random integer in the range 0 to upperlimit is returned and contained in the point 'randomnumber'. Maximum upperlimit is 32767.

Note: If 'upperlimit' is negative then the range is 0 to the negative number.

5-3-6 RunApplication

Description

Requests the operating system runs a new program. It will run in a separate process and RunApplication does not wait for the application to be launched. The specified filename must be executable i.e. have an extension of .EXE, .COM or .BAT.

Syntax

```
returnstate = RunApplication("executable")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
executable	string	Pathname of executable file.

Typical Example

```
RunApplication("c:\myprog.exe")
```

The executable file c:\myprog.exe is run.

Note: Functions that launch applications on the target system could be abused as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-3-7 RunHelp

Description

Invokes the Windows Help engine and loads a help file, showing a specific topic number.

Syntax

```
returnstate = RunHelp("helpfile",helpindex)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
helpfile	string	Pathname of helpfile to be run.
helpindex	integer	Index into a help topic as defined by the help file being run.

Typical Example

```
RunHelp("c:\myhelp.hlp",0)
```

The helpfile c:\myhelp.hlp is run, and topic 0 shown.

5-3-8 SetLanguage

Description

Change the language of text on display. This will reload the system language file from the program folder (i.e. with a .LNG extension), and the user defined text from the application folder (i.e. with a .USL extension). This function is the programmatic equivalent of the user right clicking and changing the "Language Settings…" option.

Syntax

```
SetLanguage("language name")
```

Remarks

Argument	Type	Description
language name	string	Name of language to set to. Must be identical to filename of related file with ".lng" file extension. Standard options are English, Czech, Danish, Deutsch, Español, Finnish, French, Italiano, Nederlands (België), Norwegian, Português, Slovenija and Swedish. In addition "Default" will load the designers default language.

Typical Example

```
SetLanguage("Español")
```

In this example, the Spanish language files will be loaded.

```
SetLanguage("Default")
```

In this example, the language will revert to the default specified by the application designer.

5-3-9 SetNYLED

Description

Sets the status LEDs on the NY IPC (applies to NYB and NYP only. NY5 status LEDs are dedicated to the embedded controller status)

Syntax

```
returnstate = SetNYLED ID, Action
```

Remarks

Argument	Type	Description
returnstate	Boolean	True if LED set sucessfully, otherwise false, for example run on NY5 device or regular PC.
ID	Integer	Which LED to perform action on. 0 = The Run Mode LED, 1 = The Error LED
Action	Integer	0 = Turns the LED off 1 = Turns the LED on 2 = Continously blinks the LED (250ms on / 250ms off) 3 = Continously blinks the LED (500ms on / 500ms off) 4 = Continously blinks the LED (1000ms on / 1000ms off) 5 = Single pulse of the LED (500ms on pulse)

Typical Example

```
SetNYLED 0, 1
```

In this example, the Run mode LED is turned on.

```
SetNYLED 1, 2
```

In this example, the Error LED is continuously blinked quickly.

5-3-10 GetPerformanceInfo

Description

Read the value of a performance and diagnostics Property as shown by the Performance Monitor and Diagnostics dialog.

Syntax

```
returnvalue = GetPerformanceInfo(PLC, Point, "Property
Name")
```

Remarks

Argument	Type	Description
returnvalue	integer	Value of the property returned.
PLC	string	If specified, is the name of the PLC to get the property of. If the property is not a PLC property then specify empty string "".
Point	string	If specified, is the name of the Point to get the property of. If the property is not a Point property then specify empty string "".
Property Name	string	Name of Property to read. Must be identical to the displayed property name. If both PLC and Point are empty strings then the 'Summary' property is returned.

Typical Example

```
iPerfIndex = GetPerformanceInfo("", "", "Performance
Index")
```

In this example, the Summary Performance Index will be read..

```
iCPUTime = GetPerformanceInfo("", "", "Processing Time
(ms)")
```

In this example, the CPU Time processing time will be read.

```
iCPS = GetPerformanceInfo("MyPLC", "", "Actual CPS")
```

In this example, the actual characters per second for 'MyPLC' will be returned.

```
iReadCallBacks = GetPerformanceInfo("", "MyPoint",
"Read Callbacks")
```

In this example, the read callbacks for 'MyPoint' point will be returned.

5-3-11 ShutDown

Description

Closes the CX-Supervisor application.

Syntax

```
returnstate = ShutDown()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
ShutDown()
```

CX-Supervisor runtime operation is terminated.

5-4 Communications Commands

5-4-1 CloseComponent

Syntax

```
Returnstate = CloseComponent(ComponentName, PLCName)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
ComponentName	text	A Text point or text constant containing the name of the component to close.
PLCName	text	Text point or text constant containing the name of the PLC that the component to close is attached to.

Typical Examples

```
CloseComponent("PLC Data Monitor", "MyPLC")
```

In this example, the PLC Data Monitor component monitoring the PLC 'MyPLC' is closed.

```
Component = "Performance Monitor"
PLC = "PLC06"
OK = CloseComponent(Component, PLC)
```

In this example, the Performance Monitor component monitoring the PLC 'PLC06' is closed. 'OK' is used to determine if the action was successful.

5-4-2 EnableOLE

Syntax

```
returnstate = EnableOLE(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Pointname	bool point	A Boolean point that holds the required enable/ disable state.

Typical Examples

```
EnableOLE(result)
```

OLE functions are enabled based on the value of point 'result'. If result is 'TRUE', then OLE is enabled. If result is 'FALSE', then OLE is disabled.

```
EnableOLE(TRUE)
```

OLE functions can also be enabled directly without using a point to hold the desired status.

5-4-3 EnablePLC

Syntax

```
returnstate = EnablePLC(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Pointname	bool point	A Boolean point that holds the required enable/ disable state.

Typical Examples

```
EnablePLC(result)
```

PLC functions are enabled based on the value of point 'result'. If result is 'TRUE', then PLC functions are enabled. If result is 'FALSE', then they are disabled.

```
EnablePLC(TRUE)
```

PLC functions can also be enabled directly without using a point to hold the desired status.

5-4-4 LaunchTroubleshooter

Description

Launches the SYSMAC Gateway Event Log tool to troubleshoot device errors (if installed).

Syntax

```
returnstate = LaunchTroubleshooter()
```

Remarks

Argument	Type	Description
returnstate	Boolean	True when sucessfull, otherwise False, for example when SYSMAC Gateway has not been installed.

Typical Examples

```
LaunchTroubleshooter()
```

The SYSMAC Gateway Event Log tool is launched.

5-4-5 OpenComponent

Syntax

```
Returnstate = OpenComponent(ComponentName, PLCName)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
ComponentName	text	A Text point or text constant containing the name of the component to open.
PLCName	text	Text point or text constant containing the name of the PLC that the component to open is attached to.

Typical Examples

```
OpenComponent("PLC Data Monitor", "MyPLC")
```

In this example, the PLC Data Monitor component monitoring the PLC 'MyPLC' is opened.

```
Component = "Performance Monitor"
PLC = "PLC06"
OK = OpenComponent(Component, PLC)
```

In this example, the Performance Monitor component monitoring the PLC 'PLC06' is opened. 'OK' is used to determine if the action was successful.

5-5 Point Commands

5-5-1 CancelForce

Syntax

```
returnstate = CancelForce(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point. If the point is an array point then all elements within the array have the CancelForce command applied.

Typical Example

```
CancelForce(point1)
```

The forcing of values on the point 'point1' is cancelled.

References

Refer to PLC operation manuals for a detailed description of Force Set, and Force Reset.

5-5-2 CopyArray

Syntax

```
CopyArray (SourceArray, DestArray)
```

Remarks

Argument	Type	Description
SourceArray		Name of point array to copy from.
DestArray		Name of point array to copy to.

Typical Example

```
InitArray (DestArray, 0)
```

First initialise 'DestArray'.

```
SourceArray(0) = 1
SourceArray(1) = 2
SourceArray(2) = 3
```

Then, initialise 'SourceArray' to {1, 2, 3}.

```
CopyArray (SourceArray, DestArray)
```

Finally, copy the content of the source array 'SourceArray' to the destination array 'DestArray'.

The two arrays do not have to be the same size as each other, for example if 'DestArray' contains 20 elements, only elements (0), (1) and (2) are set to 1, 2 and 3 respectively, the remaining elements are unchanged i.e. O's. If 'DestArray' is smaller than 'SourceArray' i.e. it contains two elements then only elements (0) and (1) are set to 1 and 2 respectively.

Note: 'CopyArray' accepts arrays of different type i.e. Boolean arrays can be copied into Real arrays, the only restriction is that Text arrays cannot be copied into numeric arrays and vice- versa.

5-5-3 DisableGroup

Syntax

```
returnstate = DisableGroup(groupname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
groupname	text	Name of the group containing the points to disable.

Typical Example

```
DisableGroup("<Default>")
```

All points belonging to the <Default> group is disabled thus preventing values from being read\written.

5-5-4 DisablePoint

Syntax

```
returnstate = DisablePoint(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Pointname	point	Name of point to be disabled.

Typical Example

```
DisablePoint(point1)
```

The point 'point1' is disabled thus preventing values to be read/written.

Note: This is useful for optimisation of communications.

5-5-5 EditPoint

Syntax

```
EditPoint(BoolPoint, Caption, OffText, OnText)
```

or

```
EditPoint(AnalogPoint, Caption, MinValue, MaxValue,
Keyboard)
```

or

```
EditPoint(TextPoint, EchoOff, Keyboard)
```

Remarks

Argument	Type	Description
BoolPoint	point	Name of Boolean point to be edited
Caption	Text	Text Caption for Edit dialog
OffText	Text	Text description for Boolean state 0
OnText	Text	Text description for Boolean state 1
AnalogPoint	point	Name of Integer or Real point to be edited

Argument	Type	Description
MinValue	Int/Real	Minimum value to be entered
MaxValue	Int/Real	Maximum value to be entered
Keyboard	Bool	Flag set to TRUE to display the onscreen keyboard
TextPoint	point	Name of Text point to be edited
EchoOff	Bool	Flag set to TRUE if input is not to be echoed for security

Typical Example

```
EditPoint(bFlag, "Select ON or OFF", "ON", "OFF")
```

A dialog is displayed to edit the Boolean point 'bFlag', to "ON" or "OFF" with a caption "Select ON or OFF".

```
EditPoint(nValue, "Enter a new value", 0.000000,
9999.000000, FALSE )
```

A dialog is displayed to edit the analogue point 'nValue', between 0 and 9999 with a caption "Enter a new value" without using the onscreen keyboard.

```
EditPoint(txtMessage, "Set Text to", FALSE ,FALSE )
```

A dialog is displayed to edit the Text point 'txtMessage', with a caption "Set Text to", echoing the input and not displaying the onscreen keyboard.

5-5-6 EnableGroup

Syntax

```
returnstate = EnableGroup(groupname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
groupname	text	Name of the group containing the points to enable.

Typical Example

```
EnableGroup("<Default>")
```

All points belonging to the '<Default>' group is enabled thus allowing values to be read\written.

5-5-7 EnablePoint

Syntax

```
returnstate = EnablePoint(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point to be enabled.

Typical Example

```
EnablePoint(point1)
```

The point 'point1' is enabled thus allowing values to be read/written.

5-5-8 Force

Syntax

```
returnstate = Force(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point to have force state applied. If the point is an array point then all elements within the array have the Force command applied.

Typical Example

Force(point1)

The point 'point1' is locked in its current state. i.e. if it is currently set to 1 it cannot be changed until the forced state is removed via the CancelForce command.

5-5-9 ForceReset

Syntax

```
returnstate = ForceReset(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point. If the point is an array point then all elements within the array have the ForceReset command applied.

Typical Example

```
ForceReset(point1)
```

The Boolean point 'point1' has its value set to 'FALSE'.

References

Refer to PLC operation manuals for a detailed description of ForceSet, and ForceReset.

5-5-10 ForceSet

Syntax

returnstate = ForceSet(pointname)

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point. If the point is an array point then all elements within the array have the ForceReset command applied.

Typical Example

```
ForceSet(point1)
```

The Boolean point 'point1' has its value set to 'TRUE'.

References

Refer to PLC operation manuals for a detailed description of Force Set, and Force Reset.

5-5-11 GetBit

Syntax

```
returnpoint = GetBit(pointname,bit)
```

Remarks

Argument	Type	Description
pointname	Integer / real	This is the name of the point to get the bit value from. Point value may be used.
bit	integer	This specifies which bit to get the value of.
returnpoint	bool	This contains the return value 'TRUE' or 'FALSE'.

Typical Example

```
pointname = 256;
returnpoint = GetBit(pointname,8)
```

The point 'returnpoint' contains 'TRUE'.

5-5-12 InitialiseArray

Syntax

InitArray (arrayname, value)

Remarks

Argument	Type	Description
arrayname		Name of point array.
value		Value to set all elements of the array to.

Typical Example

```
InitArray (MyArray, 0)
```

In this example, all elements of the array 'MyArray' are set to 0.

5-5-13 InputPoint

Syntax

```
returnstate = InputPoint(pointname, returnflag)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	The point name whose data is to be read.
returnflag	point	Optional Boolean point which is set to 'TRUE' when value is returned from the PLC.

Typical Examples

```
InputPoint(point)
returnflag = FALSE
InputPoint(point, returnflag)
```

A request is made that the current value of point 'point' should be read. In the second example, returnflag is set to 'TRUE' when the value is returned from the PLC.

Note: The value is not returned immediately - it is not possible to use the returned value in the same script as the InputPoint command. Instead, the value should be accessed from within an "On Condition" script which has an expression of 'returnflag = TRUE'.

5-5-14 OutputPoint

Syntax

```
returnstate = OutputPoint(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	point	The point to be updated.

Typical Examples

```
OutputPoint(result)
```

The point 'result' is updated with its current value.

Note: The value of a point connected to a PLC is not be set if the point is currently in a "forced" state.

5-5-15 PointExists

Syntax

```
returnpoint = PointExists(pointname)
```

Remarks

Argument	Type	Description
pointname	string	pointnamestringThis text contains the point name.
returnpoint	point	Boolean point that contains the return value.

Typical Example

```
PointName="Testpoint"
Exists=PointExists(PointName)
```

The Boolean point 'Exists' is set to 'TRUE' if a point called 'TestPoint' exists.

Note: "PointName" is a text point which can be set to any string value.

5-5-16 SetBit

Syntax

```
returnstate = SetBit(pointname,bit,value)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Argument	Type	Description
pointname	integer/ real	This is the name of the point to set the bit for. Point arrays may be used.
bit	point	This specifies the bit to set.
value	bool	This specifies the value to set the bit to.

Typical Example

```
testpoint = 0;
SetBit(testpoint,4,TRUE)
```

The point 'testpoint' contains the value 16.

5-6 PLC Commands

5-6-1 OpenPLC

Syntax

```
Returnstate = OpenPLC("plcname", processed)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to be opened. If the PLC is being accessed using a communications component, e.g. the Omron CX-Communications Control this parameter should be the control name and PLC name separated by a dot e.g. "OMRONCXCommunicationsControl.controlPLC".
processed	bool	Flag set to TRUE when set operation has actually been completed.

Typical Example

```
OpenPLC("controlPLC", doneopen)
```

The PLC called controlPLC is opened for communication.

Note: The PLC may not be opened immediately after the statement has been executed. The processed flag will be set at a later time when the operation has been completed. Therefore, if using statements which require the operation to be completed create an On Condition script containing the code to be executed after the PLC is opened with the 'processed' flag as the expression (this is generally more efficient).

5-6-2 ClosePLC

Syntax

```
returnstate = ClosePLC("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Argument	Type	Description
plcname	string	Name of PLC to be opened. If the PLC is being accessed using a communications component, e.g. the Omron CX-Communications Control this parameter should be the control name and PLC name separated by a dot e.g. "OMRONCXCommunicationsControl.controlPLC".

Typical Example

```
ClosePLC("controlPLC")
```

The PLC called controlPLC is closed. No further communications with the PLC will take place until it is reopened.

5-6-3 GetPLCMode

Syntax

```
mode = GetPLCMode("plcname")
```

Remarks

Argument	Type	Description
mode	string	A Text point containing the current PLC mode. Possible modes are 'STOP', 'DEBUG', 'RUN', 'MONITOR' and 'UNKNOWN'.
plcname	string	Name of PLC

Typical Example

```
currentmode = GetPLCMode("controlPLC")
```

In this example, the current mode of the PLC 'controlPLC' is stored in the point 'currentmode'.

5-6-4 SetPLCMode

Syntax

```
returnstate = SetPLCMode("plcname", mode, processed)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC
mode	string	A value for the new PLC mode. Valid modes are 'STOP', 'DEBUG', 'RUN' and 'MONITOR'.
processed	bool	processed is set to 'TRUE' when the operation is actually completed.

Typical Examples

```
SetPLCMode("controlPLC", "STOP", done)
```

In this example, the mode of the PLC called 'controlPLC' is changed to "STOP".

Note: The mode may not be changed immediately after the statement has been executed. The processed flag 'done' is set at a later time when the operation has been completed. Therefore, if using statements that require the operation to be completed create an On Condition script containing the code to be executed after the mode is set, with the processed flag as the expression (e.g. 'done').

5-6-5 SetPLCIPAddress

Syntax

```
returnstate = SetPLCIPAddress("plcname", "IPAddress")
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the IP address has been set correctly, otherwise 0.
plcname	string	Name of PLC to change the IP address of.
IPAddress	string	New IP address for the PLC.

SetPLCIPAddress is used to change the IP address of a SYSMAC Gateway or CX-Server device, from that configured at design time in Developer to a new live value. The values of both parameters may be constant text or points changed dynamically at runtime. This can be necessary for example with devices configured for DHCP, or if a single device connection will be programmatically reconfigured to many sites at runtime. The new address is not saved and lasts only while the runtime is running so you can consider storing the values in non-volatile points or your own configuration file. If you are setting the address during initialisation, it is advisable to disable the default PLC setting “Open Device”. bReturn should always be checked to confirm if the function was successful or not.

Typical Examples (VBScript)

Example1

```
SetPLCIPAddress "controlPLC", "192.168.250.101"
```

The IP address for 'controlPLC' is changed to "192.168.250.101". All other PLC properties remain unchanged. bReturn should be checked to confirm if the function was successful or not.

Example2

```
txtMyDevice = “PLC_2”
txtNewAddr = “10.0.0.1”

if (IsPLCOpen(txtMyDevice) = True) then
     ClosePLC(txtMyDevice)
end if

bReturn = SetPLCIPAddress (txtMyDevice, txtNewAddr)
if (bReturn = True) then
     OpenPLC(txtMyDevice)
else
```

```
        LogError(“Failed to Set “ + txtMyDevice +” IP
    address to “ + txtNewAddr, 2)
    end if
```

If necessary the device is closed. The PLC named in text point txtMyDevice (for example “PLC_2”) has the IP address changed to value of text point txtNewAddr (for example "10.0.0.1"). All other PLC properties remain unchanged. If OK then connection to the device is established, otherwise an error is logged to the Event log.

Note: The PLC must be closed to perform this function, otherwise SetPLCIPAddress will return an error.

Note: The IP address specified must be a valid format IPV4 address, consisting of 4 numerical values separated by three dot ‘.’ characters, otherwise SetPLCIPAddress will return an error.

5-6-6 SetPLCPhoneNumber

Syntax

```
returnstate = SetPLCPhoneNumber("plcname", numbertext)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to change the number of.
numbertext	string	New phone number for the PLC.

Typical Example

```
SetPLCPhoneNumber("controlPLC", "01234 987654")
```

The phone number for the PLC is changed to the required value.

5-6-7 UploadPLCProgram

Syntax

```
returnstate = UploadPLCProgram(plcname, filename,
processed)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to upload the program from.
filename	string	Name of the file on disk to upload the program to. If a drive and path are not specified, the file is created in the current directory, which may not be the same as the application directory. If a filename is specified as "" the user is prompted at runtime for a filename.
processed	bool	processed is set to 'TRUE' when the operation is actually completed.

Typical Example

```
UploadPLCProgram("controlPLC", "Prog01.bin", done)
```

The program in the PLC 'controlPLC' is uploaded to the file 'Prog01.bin' in the current directory. Before continuing, the script waits up to five seconds for the action to succeed.

Note: The operation may not be complete immediately after the statement has been executed. The processed flag 'done' is set at a later time when the operation has been completed. Therefore, if using statements that require the upload to be completed create an On Condition script containing the code to be executed after the upload, with the processed flag as the expression (e.g. 'done').

Note: This command can only be used when the PLC is in 'STOP' mode. Refer to chapter 6, GetPLCMode or chapter 6, SetPLCMode for further information.

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-6-8 DownloadPLCProgram

Syntax

```
returnstate = DownloadPLCProgram(plcname, filename, processed)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC to download the program to.
filename	string	Name of the file on disk to download to the PLC. If a drive and path are not specified, the current directory is assumed, which may not be the same as the application directory. If a filename is specified as "" the user is prompted at runtime for a filename.
processed	bool	processed is set to 'TRUE' when the operation is actually completed.

Typical Example

```
DownloadPLCProgram("controlPLC", "Prog01.bin", done)
```

The program stored in the file 'Prog01.bin' in the current directory is downloaded to the PLC 'controlPLC'. Before continuing, the script waits up to five seconds for the action to succeed.

Note: The operation may not be complete immediately after the statement has been executed. The processed flag 'done' is set at a later time when the operation has been completed. Therefore, if using statements that require the upload to be completed create an On Condition script containing the code to be executed after the upload, with the processed flag as the expression (e.g. 'done').

Note: This command can only be used when the PLC is in 'STOP' mode. Refer to chapter 6, GetPLCMode or chapter 6, SetPLCMode for further information.

5-6-9 IsPLCOpen

Syntax

```
returnstate = IsPLCOpen("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to be checked.

Typical Example

```
IsOpen = IsPLCOpen("controlPLC")
```

The Boolean point IsOpen is set to true if the PLC called 'controlPLC' is currently open. Otherwise it is set to false.

5-6-10 PLCCommsFailed

Syntax

```
returnstate = PLCCommsFailed("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to be checked.

Typical Example

```
IsFailing = PLCCommsFailed ("controlPLC")
```

The point IsFailing is set to true if the PLC called controlPLC is currently not communicating. Otherwise it is set to false.

Note:This function returns to TRUE from the time when a communications timeout error with the named PLC occurs, until successful communication with the PLC takes place.

5-6-11 PLCMonitor

Syntax

```
returnstate = PLCMonitor("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to be monitored.

Typical Example

```
PLCMonitor("controlPLC")
```

The monitor dialog for the PLC called controlPLC is invoked. This dialog can be used to check PLC status, change mode, etc.

5-7 Temperature Controller Commands

5-7-1 TCAutoTune

Syntax

```
returnstate = TCAutoTune(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines the operation to be carried out when a TCAutoTune command is issued. 0: Indicates that the auto-tuning operation is to be stopped. 1: This mode is supported on the E5*K and is used to set the limit cycle of the manipulated variable change width to 40%. 2: This is used to start the auto-tuning operation.

Typical Example

```
temp1 = TCAutoTune("e5ak",temp2)
```

5-7-2 TCBackupMode

Syntax

```
returnstate = TCBackupMode(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines the method used by a temperature controller for storing internal variables. 0: In this mode variables are stored in RAM and EPROM. 1: In this mode variables are stored in RAM only.

Typical Example

```
temp1 = TCBackupMode("ea5k",temp2)
```

5-7-3 TCGetStatusParameter

Syntax

```
returnstate                                                =
TCGetStatusParameter(TController,paramID,value)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.

Argument	Type	Description
paramID	point	This is a point depicting the required parameter range 0 to 22: 0: ControlMode. 1: Output. 2: InputShiftDelay (Bool) E5*F, E5*X, E5*J. 3: DisplayUnit. 4: PIDConstantDisplay (Bool) E5*F, E5*X, E5*J. 5: OutputType. 6: CoolingType. 7: Output2. 8: Alarm1. 9: Alarm2. 10: InputType (Integer) E5*F, E5*X, E5*J. 11: OperationMode. 12: BackupMode. 13: AutoTuneMode. 14: OverFlow (Bool) E5*F, E5*X, E5*J. 15: UnderFlow (Bool) E5*F, E5*X, E5*J. 16: SensorMalfunction (Bool) E5*F, E5*X, E5*J. 17: ADConvertorFailure (Bool) E5*F, E5*X, E5*J. 18: RAMAbnormality (Bool) E5*F, E5*X, E5*J. 19: RAMMismatch (Bool) E5*F, E5*X, E5*J. 20: StatusWordsOnly (Bool) E5*K only (TRUE indicates valid words below). 21: Status0 (word) E5*K only. 22: Status1 (word) E5*K only.
value	point, real or int	The returned status parameter value. Refer to paramID above for details.

Typical Example

```
temp1 = TcGetStatusParameter("e5ak",temp2,temp3)
```

5-7-4 TCRemoteLocal

Syntax

```
returnstate = TCRemoteLocal(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.

Argument	Type	Description
mode	point	This is a point depicting the mode of operation and defines the operational mode of a temperature controller. 0: This specifies the temperature controller is in remote mode. 1: This specifies that the temperature controller is in local mode.

Typical Example

```
temp1 = TCRemoteLocal("e5ak",temp2)
```

Note: This command was previously called TCOperationalMode.

5-7-5 TCRequestStatus

Syntax

```
returnstate = TCRequestStatus(Tcontroller, returnflag)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.
returnflag	point	This is a point depicting that the status has been returned and is available for the command TCGetStatusParameter.

Typical Example

```
temp1 = TCRequestStatus("e5ak", temp2)
```

Note: The status information is NOT returned immediately - it is not possible to access the status information in the same script as the TCRequestStatus command. Instead, the status information should be accessed from within an "On Condition" script which has an expression of "returnflag = TRUE".

5-7-6 TCRspLsp

Syntax

```
returnstate = TCRspLsp(Tcontroller,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines the setpoint mode used by the temperature controller. 0: This specifies local setpoint mode. 1: This specifies remote setpoint mode.

Typical Example

```
temp1 = TCRspLsp("e5ak",temp2)
```

Note: This command was previously called TCSetpoint.

5-7-7 TCRunStop

Syntax

```
returnstate = TCRunStop(TController,mode)
```

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines either auto-output mode shift or manual output mode shift. 0: This specifies manual output mode shift. 1: This specifies auto-output mode shift.

Typical Example

```
temp1 = TCRunStop("e5ak",temp2)
```

Note: This command was previously called TCModeShift.

5-7-8 TCSaveData

Syntax

```
returnstate = TCSaveData(TController)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.

Typical Example

```
temp1 = TCSaveData("e5ak",temp2)
```

5-7-9 TCSettingLevel1

Syntax

```
returnstate = TCSettingLevel1(TController)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.

Typical Example

```
temp1 = TCSettingLevel1("e5ak")
```

5-7-10 TCReset

Syntax

```
returnstate = TCReset(TController)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
TController	string	This is a string representing the name of the temperature controller.

Typical Example

```
temp1 = TCReset("e5ak")
```

5-8 Alarm Commands

5-8-1 AcknowledgeAlarm

Syntax

```
returnstate = AcknowledgeAlarm("alarmname")
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
alarmname	string	This is the identifier of the alarm.

Typical Example

```
AcknowledgeAlarm("temphigh")
```

The alarm 'temphigh' is acknowledged.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-2 AcknowledgeAllAlarms

Syntax

```
returnstate = AcknowledgeAllAlarms()
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
AcknowledgeAllAlarms()
```

All alarms are acknowledged.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-3 AcknowledgeLatestAlarm

Syntax

```
returnstate = AcknowledgeLatestAlarm()
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
AcknowledgeLatestAlarm()
```

The most current alarm of the highest priority is acknowledged.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-4 ClearAlarmHistory

Syntax

```
returnstate = ClearAlarmHistory()
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
ClearAlarmHistory()
```

The alarm history window is cleared and the log is cleared.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-5 CloseAlarmHistory

Syntax

```
returnstate = CloseAlarmHistory()
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
CloseAlarmHistory()
```

The alarm history window is closed.

References

Refer to the CX-Supervisor User Manual for details of alarms

5-8-6 CloseAlarmStatus

Syntax

returnstate = CloseAlarmStatus()

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
CloseAlarmStatus()
```

The current alarm status window is closed.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-7 DisplayAlarmHistory

Syntax

```
returnstate = DisplayAlarmHistory()
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
DisplayAlarmHistory()
```

The alarm history window is displayed.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-8 DisplayAlarmStatus

Syntax

```
returnstate = DisplayAlarmStatus()
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.

Typical Example

```
DisplayAlarmStatus()
```

The current alarm status is displayed.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-9 EnableAlarms

Syntax

```
EnableAlarms (flag, "message")
```

Remarks

Argument	Type	Description
flag		If set 'TRUE' then alarm logging is enabled. If set 'FALSE' logging is disabled.
message		Text message which is recorded in the alarm log to indicate change of status.

Typical Example

```
EnableAlarms (TRUE, "Alarm logging enabled")
```

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-10 IsAlarmAcknowledged

Syntax

```
pointname = IsAlarmAcknowledged("alarmname")
```

Remarks

Argument	Type	Description
pointname	bool point	The Boolean point name to be assigned a value based on the test of an acknowledged alarm.
alarmname	string	The identifier of the alarm.

Typical Example

```
acknowledged = IsAlarmAcknowledged("temptoohigh")
```

The point 'acknowledged' is assigned Boolean state "TRUE" if the 'temptoohigh' alarm is currently acknowledged. The point is assigned Boolean state 'FALSE' if the alarm is not currently acknowledged.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-8-11 IsAlarmActive

Syntax

```
pointname = IsAlarmActive("alarmname")
```

Remarks

Argument	Type	Description
pointname	bool point	The Boolean point name to be assigned a value based on the test of an active alarm.
alarmname	string	The identifier of the alarm.

Typical Example

```
active = IsAlarmActive("temptoohigh")
```

The point 'active' is assigned Boolean state "TRUE" if the 'temptoohigh' alarm is currently active. The point is assigned Boolean state 'FALSE' if the alarm is not currently active.

References

Refer to the CX-Supervisor User Manual for details of alarms.

5-9 File Commands

5-9-1 CloseFile

Syntax

```
returnstate = CloseFile(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	bool	A Boolean point that holds the required status of whether blank spaces should be stripped from the file when it is closed.

Typical Examples

```
CloseFile(status)
```

The currently open file is closed. Blank spaces at the end of each line are stripped from the file if the Boolean point 'status' is set to 'TRUE'.

```
CloseFile(FALSE)
```

Note: If blank spaces are stripped from the file, then it greatly reduces in size but it takes slightly longer to close. Blank spaces should not be stripped from the file if it is being used on a network drive by more than one system at a time.

In this example, the currently open file is closed and any blank spaces are not stripped from the file.

5-9-2 CopyFile

Syntax

```
returnstate = CopyFile("sourcename", "destname")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
sourcename	string	Pathname of file to be copied. May include a "*" wildcard character.
destname	string	Pathname of destination of copy. If path name does not exist it is created.

Typical Example

```
CopyFile("c:\autoexec.bat", "c:\autoexec.old")
```

The file "c:\autoexec.bat" is copied to the file "c:\autoexec.old".

```
CopyFile("c:\logging\*.dlv", "a:\backup")
```

The data log files (ending in dlv) in "C:\logging" are copied to the "\backup" directory on drive A:

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-9-3 DeleteFile

Syntax

returnstate = DeleteFile("filename")

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be deleted.

Typical Example

```
DeleteFile("c:\pagename.pag")
```

The file "c:\pagename.pag" is deleted.

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-9-4 EditFile

Syntax

```
returnstate = EditFile("filename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be edited.

Typical Example

```
EditFile("C:\report3.txt")
FileExists
```

Syntax

```
returnpoint = FileExists (filename)
```

Remarks

Argument	Type	Description
filename	string	This text string contains the file name.
returnpoint	point	Boolean point that contains the return value.

Typical Example

```
FileName = "TEST.TXT"
```

Exists = FileExists(FileName)

The Boolean point 'Exists' is set to 'TRUE' if a file called 'C:\TEST.TXT' exists.

Note: "FileName" is a text point which can be set to any string value.

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-9-5 MoveFile

Syntax

```
returnstate = MoveFile("sourcename", "destname")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
sourcename	string	Pathname of file to be moved.
destname	string	Pathname of destination of move.

Typical Example

```
MoveFile("c:\autoexec.bat", "c:\autoexec.old")
```

The file "c:\autoexec.bat" is moved to the file "c:\autoexec.old".

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-9-6 OpenFile

Syntax

```
returnstate = OpenFile("filename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be opened.

Typical Example

```
OpenFile("c:\filename")
```

The file "c:\filename.csf" is opened and able to be accessed by the Read() and Write() script commands. Only one file can be open at a time. A file is created if it doesn't already exist. Files can be shared (for instance located on a network drive, and accessed by several running CX-Supervisor applications simultaneously - this can be used for data exchange).

Note: An extension ".csf" will always be added to the filename so it must not be specifed as part of the argument.

5-9-7 PrintFile

Syntax

```
returnstate = PrintFile("filename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be printed.

Typical Example

```
PrintFile("c:\autoexec.bat")
```

The file "c:\autoexec.bat" is sent to the currently configured printer.

Script commands that have textual arguments can take either literal strings within quotes or text points.

Note: CX-Supervisor uses the OLE registration information (file extension associations) to decide how to print a file. It invokes the parent application associated with a particular file extension, instructing the application to start minimised and passing the "print" command. For example, if the file extension .txt is associated with Notepad, then Notepad is invoked to print the file.

5-9-8 Read

Syntax

```
returnstate = Read(RecordId, pointname, ...)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
RecordId	integer	An index into the file.
Pointname	point	Name(s) of point(s) to be updated with the data read from the open file.

Typical Examples

```
Read(1, value)
```

The point 'value' is loaded with the value read from the currently open file using the value of 1 as an index into the file.

```
ReadOK = Read(indexno, value1, value2, value3)
```

The points 'value1', 'value2', 'value' are loaded using the value of indexno as an index into the file. Pass or fail status is stored in 'ReadOK'.

Note: It is advisable to use a RecordId less than 1024 whenever possible, in order to optimise file access time (records 0 to 1023 are cached).

5-9-9 ReadMessage

Syntax

```
returnstate = ReadMessage ("filename", offset,
textpoint, noofchars)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be read.
Offset	integer	An offset from the beginning of the file (in characters) indicating where to start reading from.
Textpoint	text point	The text point which holds the characters read from the file.
Noofchars	integer	The number of characters to read from the file.

Typical Example

```
ReadMessage ("C:\CX-SUPERVISOR\TESTFILE.TXT", 0,
TextPoint, 20)
```

The first 20 characters are be read from the file "C:\CX-SUPERVISOR\TESTFILE.TXT" and stored in the point 'TextPoint'.

Note: Text points can hold up to 256 characters therefore a maximum of 256 characters can be read from the file.

5-9-10 SelectFile

Syntax

```
filename = SelectFile (filter, path)
```

Remarks

Argument	Type	Description
Filename		Text string returned. Contains fully qualified filename including drive and path if OK was selected from OpenFile comms dialog, otherwise contains empty string.
Filter	string	Optional argument. If omitted, will show all files. This argument must be supplied if path is specified i.e. set to "". Specifies the filter string used by the 'Files of type' list. The string should contain 1 or more filters separated with a '|' (pipe) character and end with 2 characters i.e. '||'. Each filter should have some user text and 1 or more file specs separated with a semicolon. No spaces should be used, except within the user text.
Path	string	Optional argument. Specifies the path to show initially. If omitted, the dialog shows the current working directory.

Typical Example

```
TFile = SelectFile()
```

The 'File Open' dialog will be displayed, showing all files in the current working directory. The users choice will be stored in tFile.

```
TFile = SelectFile("Text Files (*.txt)|*.txt||")
```

The 'File Open' dialog will be displayed, showing just files with a .txt extension in the current working directory.

```
TFile = SelectFile("Text Files (*.txt;
*.csv)|*.txt;*.csv||")
```

The 'File Open' dialog will be displayed, showing files with either a .txt or .csv extension in the current working directory.

```
TFile = SelectFile("Text Files (*.txt;
*.csv)|*.txt;*.csv|Document Files (*.doc)|*.doc||")
```

In this example, the 'Files of type' filter has 2 choices: one to show text files (i.e. both .txt and .csv files), and one to show document files (just .doc files).

```
TFile = SelectFile("", "C:\WINDOWS")
```

The 'File Open' dialog will be displayed, showing all files in the "C:\WINDOWS" directory.

5-9-11 Write

Syntax

```
returnstate = Write(RecordId, pointname, ...)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
RecordId	integer	An index into the file.
Pointname	point	Name(s) of point(s) containing data to write to the open file.

Typical Examples

```
WroteOK = Write(indexno, $Second)
```

The point '$Second' is written to the currently open file using the value of indexno as an index into the file. Pass or fail status is stored in 'WroteOK'.

```
Write(2, $Second, $Minute, $Hour)
```

The points '$Second', '$Minute', '$Hour' are written to the currently open file using the value 2 as an index into the file.

Note:It is advisable to use a RecordId less than 1024 whenever possible, in order to optimise file access time (records 0 to 1023 are cached).

5-9-12 WriteMessage

Syntax

```
returnstate = WriteMessage("filename", offset, "text",
linefeed)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
filename	string	Pathname of file to be written.
offset	integer	An offset from the beginning of the file (in characters) indicating where to start writing. If the offset is -1 then the message is appended to the end of the file.
text	string	The text to be written into the file.
linefeed	bool	A flag to indicate a carriage return and line feed should be appended.

Typical Example

```
WriteMessage("C:\CX-SUPERVISOR\TESTFILE.TXT",      0,
"Hello World", TRUE)
```

The text 'Hello World' is written at the start of the 'C:\CX-SUPERVISOR\TESTFILE.TXT' file and a carriage return and line feed is appended which moves and subsequent text to the start of the next line.

Note: When the text is written into the file it overwrites any existing text that may exist at this location.

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-10 Recipe Commands

5-10-1 DisplayRecipes

Syntax

```
returnstate = DisplayRecipes()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DisplayRecipes()
```

The current recipes is displayed.

References

Refer to the CX-Supervisor User Manual for details of recipes.

5-10-2 DownloadRecipe

Syntax

```
returnstate = DownloadRecipe("recipename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
recipename	string	The name of the recipe to be downloaded.

Typical Example

```
DownloadRecipe("recipe1")
```

The recipe 'recipe1' is downloaded.

References

Refer to the CX-Supervisor User Manual for details of recipes.

5-10-3 UploadRecipe

Syntax

```
returnstate = UploadRecipe("recipename", processed)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
recipename	string	The name of the recipe to be uploaded.
processed	bool	Flag set to true when operation has been completed.

Typical Example

```
UploadRecipe("recipe1",done)
```

The recipe 'recipe1' is uploaded, and point 'done' is set True when the upload is complete.

References

Refer to the CX-Supervisor User Manual for details of recipes.

5-11 Report Commands

5-11-1 GenerateReport

Syntax

```
returnstate =
GenerateReport(ReportTemplateFile,ReportOutputFile)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
ReportTemplateFile	string	Pathname of the report template file.
ReportOutputFile	string	Pathname of the report output file.

Typical Example

```
GenerateReport("report3.txt","output.txt")
```

The ReportTemplateFile report3.txt contains a predefined set of point names and text laid out exactly as the report reader likes to view them. The point names contained within enclosing characters are the CX-Supervisor names for the data that is required in the report.

The enclosing characters can be changed in the Project menu->Runtime Settings->Point Substitution Settings dialog box, and once set must be fixed for all reports generated by the project.

The template file can be written using any ASCII text editor, for instance a Text file (.TXT), a Rich Text file (.RTF) or a Hypertext file (.HTML).

The report template is processed, dynamically replacing the point names with current values, and saved as output.txt.

Note: Functions that create files on the target system could be abused to create malicious files that could be used as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-11-2 PrintReport

Syntax

```
returnstate = Printreport(ReportTemplateFile)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
ReportTemplateFile	string	Pathname of the report template file.
ReportOutputFile	string	Pathname of the report output file.

Typical Example

```
PrintReport("report3.txt")
```

The report template is processed, dynamically replacing the point names with current values, and printed to the default Windows printer.

5-11-3 ViewReport

Syntax

```
returnstate = ViewReport(ReportTemplateFile)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
ReportTemplateFile	string	Pathname of the report template file.

Typical Example

```
ViewReport("report3.txt")
```

Note: Functions that launch applications on the target system could be abused as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths.

5-11-4 EmailReport

Syntax

```
returnstate = EmailReport(To, Subject,
ReportTemplateFile, CC, BCC, Attachments)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
To	string	Email addresses that the generated report should be sent to. Multiple email addresses should be separated by a semi-colon (;).
Subject	string	The string that will be displayed in the Subject field of the email.
ReportTemplateFile	string	File name of the report template file. A full path can be specified by starting with the drive letter.
CC	string	Email addresses that the generated report should be copied to. Multiple email addresses should be separated by a semi-colon (;).
BCC	string	Email addresses that the generated report should be blind copied to. Multiple email addresses should be separated by a semi-colon (;).
Attachments	string	File name of the file that should be attached to the email. A full path can be specified by starting with the drive letter. Multiple files can be attached by separating them with a semi-colon (;).

Typical Example

```
EmailReport("user@Omron.com","Generated CX-Supervisor
Report","D:\Reports\ReportTemplate.txt","","","")
```

The report template is processed, dynamically replacing the point names with current values, and then emailed to the specified recipients - just user@Omron.com in this case.

Note: For more details on generating report Templates refer to User Manual Appendix B, How do I create reports and HTML reports?

5-12 Text Commands

5-12-1 BCD

Syntax

```
result = BCD (value)
```

Remarks

Argument	Type	Description
Value		Number to convert to Binary Coded Decimal (BCD).
result		String containing BCD representation of value.

Typical Example

```
BCDStr = BCD(39)
```

In this example, 'BCDstr' contains '00111001'.

5-12-2 Bin

Syntax

```
result = Bin (value)
```

Remarks

Argument	Type	Description
Value		Number to be converted to a binary number.
result		String containing binary representation of value.

Typical Example

```
BStr = Bin (20)
```

In this example, 'Bstr' contains '10100'.

5-12-3 Chr

Syntax

```
result = Chr (value)
```

Remarks

Argument	Type	Description
Value		Extended ASCII value to convert to a character.
result		String containing single character representation of value.

Typical Example

```
Char = Chr(65)
```

In this example, 'Char' contains 'A'.

5-12-4 FormatText

Syntax

```
textpoint = FormatText ("formattext", expression, ...)
```

Remarks

Argument	Type	Description
textpoint	text point	A text point which holds the formatted text.
formattext	string	The text (with appropriate formatting characters) that the result expression is inserted into.
expression	Integer / real	The value(s) or expression(s) that is inserted into formattext.

Typical Examples

```
TextPoint = FormatText ("Boiler temperature is %ld
degrees.",  BoilerTemp)
```

The value of the 'BoilerTemp' point is inserted into the specified text at the position marked by the formatting characters (%ld) and then stored in the point 'TextPoint'.

If the value of 'BoilerTemp' was 57 then the resultant text that is stored in 'TextPoint' is as follows:

```
"Boiler temperature is 57 degrees."
TextPoint = FormatText ("Boiler %ld temperature is %ld
degrees.",  BoilerNo, BoilerTemp)
```

The value of 'BoilerNo' point is inserted at the first '%ld' marker and the value of the 'BoilerTemp' point is inserted at the second '%ld' marker and the resulting string is stored in the point 'TextPoint'.

If the value of 'BoilerNo' was 7 and the value of 'BoilerTemp' was 43 then the resultant text stored in the 'TextPoint' is as follows:

```
"Boiler 7 temperature is 43 degrees."
```

Note: The formatting characters are standard 'C' formatting characters (as used by the C-language sprintf function). Some commonly used types are:

- %ld. Insert integer value;
- %f. Insert decimal value. Prefix with decimal point and number to control position (for instance '%.2f' for 2 decimal places);
- %s. Insert string;
- %lX. Insert hexadecimal value (upper case HEX characters, for instance 'FFFF');
- %lx. Insert hexadecimal value (lower case HEX characters, for instance 'ffff');
- %c. Insert character (can be used to convert value to character, for instance to insert control character).

With the text left aligned, and with a width field (for instance '%-6ld' to insert a value left aligned with a field 6 characters wide).

References

More complex expressions (for instance controlling justification, decimal places, number base, etc.) are also possible. Refer to any C language reference book for full details of the format used by the 'sprintf' function.

5-12-5 GetTextLength

Syntax

```
value = GetTextLength (textpoint)
```

Remarks

Argument	Type	Description
textpoint	text point	This is the point which has its text length counted.
returnpoint	Integer / real	This is the point that holds the return value.

Typical Example

```
textpoint = "Hello World"
count = GetTextLength (textpoint)
```

The number of characters in 'textpoint' is counted and the point 'count' is set to the value 11.

5-12-6 Hex

Syntax

```
result = Hex (value)
```

Remarks

Argument	Type	Description
Value		Number to be converted to a Hex number.
Result		String containing Hex representation of value.

Typical Example

```
HStr = Hex (44)
```

In this example, 'Hstr' contains '2C'.

5-12-7 Left

Syntax

```
lefttext = Left(textpoint,noofchars)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be manipulated.
noofchars	integer	The number of characters to extract from the start of the string.
lefttext	text	Text point containing the specified range of characters.

Typical Example

```
textpoint = "abcdefgh"
lefttext = Left(textpoint,3)
```

The text point 'lefttext' contains the string 'abc'.

5-12-8 Message

Syntax

```
Message("message")
```

Remarks

Argument	Type	Description
message	string	Contains the text string that is displayed in the message box.

Typical Example

```
Message("this is a message")
```

The message 'this is a message' is displayed in a Message Box.

5-12-9 Mid

Syntax

```
midtext = Mid(textpoint,offset,noofchars)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be manipulated.
offset	integer	The zero based index of the first character in the string that is to be included in the extract.
noofchars	integer	The number of characters to extract from the string.
midtext	text	Text point containing the specified range of characters.

Typical Example

```
textpoint = "abcdefgh"
midtext = Mid(textpoint,3,2)
```

The text point 'midtext' contains the string 'de'.

5-12-10 PrintMessage

Syntax

PrintMessage ("message")

Remarks

Argument	Type	Description
message	string	Contains the text string that is sent to the printer.

Typical Example

```
PrintMessage ("Print this message")
```

The message 'print this message' is printed to the configured 'Alarm/message printer', queued if operating in page mode, or printing has been disabled by the EnablePrinting command.

References

Refer to the CX-Supervisor User Manual for further details to configure the 'Alarm/message printer'.

5-12-11 Right

Syntax

```
righttext = Right(textpoint,noofchars)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be manipulated.
noofchars	integer	The number of characters to extract from the string.
righttext	text	Text point containing the specified range of characters.

Typical Example

```
textpoint = "abcdefgh"
righttext = Right(textpoint,3)
```

The text point 'righttext' contains the string 'fgh'.

5-12-12 TextToValue

Syntax

```
valuepoint = TextToValue(textpoint)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be converted into a number.
valuepoint	integer	A point containing the value returned after conversion from a string.

Typical Examples

```
textpoint = "10"
valuepoint = TextToValue(textpoint)
```

The value 10 is assigned to the point 'valuepoint'.

```
textpoint = "10.34"
realpoint = TextToValue(textpoint)
```

The real value 10.34 is assigned to the real point 'realpoint'.

5-12-13 ValueToText

Syntax

```
textpoint = ValueToText(value)
```

Remarks

Argument	Type	Description
value	integer	The number that is to be placed into the textpoint. A point name is also a valid parameter.
textpoint	text point	A text point containing the value converted into a string.

Typical Examples

```
textpoint = ValueToText(10)
```

The value 10 is put into a string and assigned to the text point 'textpoint'.

```
value = 10
```

```
textpoint = ValueToText(value)
```

This has the same effect as the previous example.

5-12-14 EmailText

Syntax

```
returnstate = EmailText(To, Subject, BodyText, CC,
BCC, Attachments)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
To	string	Email addresses that the generated report should be sent to. Multiple email addresses should be separated by a semi-colon (;).
Subject	string	The string that will be displayed in the Subject field of the email.
BodyText	string	The text message that should be used for the main body of the email. Point values can be embedded in the text by specifying them within double brackets (e.g. The date is (($Date)).
CC	string	Email addresses that the generated report should be copied to. Multiple email addresses should be separated by a semi-colon (;).
BCC	string	Email addresses that the generated report should be blind copied to. Multiple email addresses should be separated by a semi-colon (;).
Attachments	string	File name of the file that should be attached to the email. A full path can be specified by starting with the drive letter. Multiple files can be attached by separating them with a semi-colon (;).

Typical Example

```
EmailText("user@Omron.com","Sent by CX-Supervisor",
"Hello,\r\n\r\nThis email was sent via the ‘EmailText’
script function.","","","")
```

The body text will be processed (i.e. embedded Points will have their current values inserted into the text) and then the email will be sent to the specified recipients - just user@omron.com in this case.

Note: For more details on dynamic substitution of point values refer to User Manual chapter 2-13, Embedding Point Values in Text.

5-13 Event/Error Commands

5-13-1 ClearErrorLog

Syntax

```
ClearErrorLog()
```

Typical Example

```
ClearErrorLog()
```

The error list is cleared and the log deleted.

5-13-2 CloseErrorLog

Syntax

```
returnstate = CloseErrorLog()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
CloseErrorLog()
```

The list of all currently logged errors is closed.

5-13-3 DisplayErrorLog

Syntax

```
returnstate = DisplayErrorLog()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DisplayErrorLog()
```

A list of all currently logged errors is displayed in a dialog.

5-13-4 EnableErrorLogging

Syntax

```
returnstate = EnableErrorLogging(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	bool	A Boolean point.

Typical Example

```
EnableErrorLogging(flag)
```

Error Logging is enabled based on the Boolean point 'flag'. If 'flag' is 'TRUE', then error logging is enabled. If 'flag' is false, then error logging is disabled.

5-13-5 LogError

Syntax

```
returnstate = LogError("message", priority)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
message	string	Contains the text string that is displayed in the Error Log.
priority	integer	Priority assigned to the error. 0 - low 1 - medium 2 - high

Typical Example

```
LogError("This is an error", 1)
```

The message 'This is an error' appears as a medium priority error in the error log.

5-13-6 LogEvent

Syntax

```
returnstate = LogEvent("message")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
message	string	Contains the text string that is displayed in the Error Log.

Typical Example

```
LogEvent("this is an event")
```

The message 'this is an event' appears as an event in the error log.

5-14 Printer Commands

5-14-1 ClearSpoolQueue

Syntax

```
returnstate = ClearSpoolQueue()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
ClearSpoolQueue()
```

Any messages (typically printed alarms) that are queued up waiting to be sent to the CX-Supervisor Alarm/Message printer is discarded.

5-14-2 EnablePrinting

Syntax

```
returnstate = EnablePrinting(flag)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
flag	bool	0 to disable, 1 to enable.

Typical Example

```
EnablePrinting(FALSE) - Disables printing
EnablePrinting(TRUE) - Enables printing
```

While alarm printing is disabled, any new messages are stored but not printed. When alarm printing is re-enabled, any pending messages are printed (if in line mode) or added to the current page (if in page mode).

5-14-3 PrintActivePage

Syntax

```
returnstate = PrintActivePage(flag)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
flag	bool	Flag is to indicate whether the print setup dialog is to be displayed before printing.

Typical Example

```
PrintActivePage(TRUE)
```

The currently active page is sent to the printer. The flag 'TRUE' indicates that the print dialog is displayed. 'FALSE' causes the print dialog not to be shown.

5-14-4 PrintPage

Syntax

```
returnstate = PrintPage ("pagename", flag,
printheaderfooter)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pagename	string	The name of the page to be printed.
flag	bool	Flag to indicate whether the print setup dialog is to be displayed before printing.
printheaderfooter	bool	Optional. Flag to control if printout details are included in a header and footer.

Typical Example

```
PrintPage("page1", TRUE)
```

The CX-Supervisor page is sent to the printer. The flag 'TRUE' indicates that the print dialog is displayed first to allow for printer configuration. If 'FALSE' was specified instead of 'TRUE' then the print dialog is not shown, the page is just printed.

5-14-5 PrintScreen

Syntax

```
returnstate = PrintScreen(flag)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
flag	bool	Flag to indicate whether the print setup dialog is to be displayed before printing.

Typical Example

```
PrintScreen(FALSE)
```

All CX-Supervisor pages currently on view is printed. The flag 'FALSE' indicates that the print dialog is not displayed. A flag of 'TRUE' causes the print dialog to be shown, allowing the user to configure or choose the printer.

5-14-6 PrintSpoolQueue

Syntax

```
returnstate = PrintspoolQueue()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
PrintSpoolQueue
```

Any message (typically printed alarms) that are queued up waiting to be sent to the CX-Supervisor Alarm/Message printer is printed immediately.

5-15 Security Commands

5-15-1 Login

Syntax

```
returnstate = Login(username, password)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
username	text	Optional parameter with name of user to login. If omitted, the login dialog will be shown.
password	text	Optional parameter with password for user to login. If used, username must be specified, even if only empty i.e. "". If omitted, the login dialog will be shown.

Typical Examples

```
Login()
```

The Login dialog is displayed for user entry.

```
Login("Designer", "Designer")
```

The default 'Designer' user is logged in automatically using matching password.

References

Refer to the CX-Supervisor User Manual for details of Login.

5-15-2 Logout

Syntax

```
returnstate = Logout(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
pointname	bool	A Boolean that holds the required status of whether a confirmation message box should be displayed before logging out the user.

Typical Examples:

```
Logout()
Logout(FALSE)
```

The user will be logged out without displaying the confirmation message.

```
Logout(TRUE)
```

A message box will be displayed with the text ‘Are you sure you want to logout ‘User Name’? The user will be logged out only if the ‘Yes’ button is clicked.

5-15-3 SetupUsers

Syntax

```
returnstate = SetupUsers()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
SetupUsers()
```

The Setup Users dialog is displayed for user entry.

References

Refer to the CX-Supervisor User Manual for details of setting and modifying user details.

5-15-4 ChangeUserPassword

Syntax

```
ChangeUserPassword("username","old", "new")
```

Remarks

Argument	Type	Description
username	string	user whose password should be changed.
old	string	the existing password.
new	string	the new password.

Typical Example

```
ChangeUserPassword("Fred Smith","fred1", "fred2")
```

The ChangeUserPassword would change ‘Fred Smith’s’ Windows Logon password from ‘fred1’ to ‘fred2’.

References

Refer to the CX-Supervisor User Manual for details of setting and modifying user details.

5-16 Data Logging Commands

5-16-1 AuditPoint

Syntax

```
AuditPoint("pointname")
```

Remarks

Argument	Type	Description
pointname	string	Name of the point to be logged into the CFR database.

Typical Example

```
AuditPoint("MyInteger")
```

This command will cause the value of ‘MyInteger’ to be logged into the CFR database.

5-16-2 ClearLogFile

Syntax

```
ClearLogFile("datasetname")
```

Remarks

Argument	Type	Description
datasetname	string	Name of Data Set to clear as text point or constant.

Typical Example

```
ClearLogFile("Process 1")
```

This command will clear all data from the active (latest) log file for this data set, and add a 'Clear Event' indicator.

5-16-3 CloseLogFile

Syntax

```
returnstate = CloseLogFile("datasetname")
```

or

```
returnstate = CloseLogFile("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
datasetname	text	Name of Data Set to close as text point or constant.
databaselink	text	Name of Database link to close as text point or constant.

Typical Example

```
CloseLogFile("Process 1")
```

This command will close the active log file for the data set. Logging for this data set is automatically stopped.

5-16-4 CloseLogView

Syntax

```
CloseLogView("datasetname")
```

Remarks

Argument	Type	Description
datasetname	text	Name of Data Set to close as text point or constant.

Typical Example

```
CloseLogView("Process 1")
```

This command will close the Data Log Viewer, which is displaying the named data set.

5-16-5 ExportAndViewLog

Syntax

```
ExportAndViewLog ("datasetname", "item list",
"format", file, outputfile)
```

or

ExportAndViewLog ("datasetname", TextArray, "format", file, outputfile)

Remarks

Argument	Type	Description
datasetname	text	Name of Data Set to close as text point or constant.
item list	string	List of Items and/or Groups within the data set to export, separated by commas. Alternatively use "*" to export all.
TextArray	string array	A text point, which has an array size specified as 1 or more elements . Each element holds an Item or Group name.

Argument	Type	Description
format	string	Either "CSV" or "Text" to specify output format. May include suffix '-' followed by: B to exclude break information D to exclude the log date T to exclude the log time M to exclude to log milliseconds G to not Group 'On Change' data together
file	integer	Number of file to export where 0 is the latest (active) file, 1 is the previous file etc.
outputfile	string	File name for output file. May include full path, which will be created automatically if it does not exist.

All these arguments are optional, and may be omitted provided there are no further arguments i.e. to specify the 'format', 'datasetname' and 'item list' must be included but 'file' and 'output' may be omitted.

Typical Examples

```
ExportAndViewLog("Balloon", "*")
```

or

```
ExportAndViewLog("Balloon",
"Altitude,Fuel,Burning,Lift,Group 1", "CSV-BDTM", 0,
"output")
```

or

```
ItemList(0) = "Altitude"
ItemList(1) = "Fuel"
ItemList(2) = "Burning"
ItemList(3) = "List"
ItemList(4) = "Group 1"
ExportAndViewLog("Balloon", ItemList, "CSV-BDTM", 0,
"output")
```

All these commands will export all the data in the specified file, for the named data set to the named output file, in the format specified (as per ExportLog). It then launches an appropriate viewer to display the file, using the Windows file associations.

5-16-6 ExportLog

Syntax

```
ExportLog ("datasetname", "item list", "format", file,
outputfile)
```

or

```
ExportLog ("datasetname", TextArray, "format", file,
outputfile)
```

Remarks

Argument	Type	Description
datasetname	text	Name of Data Set to close as text point or constant.

Argument	Type	Description
item list	string	List of Items and/or Groups within the data set to export, separated by commas. Alternatively use "*" to export all.
TextArray	string array	A text point, which has an array size specified as 1 or more elements . Each element holds an Item or Group name.
format	string	Either "CSV" or "Text" to specify output format. May include suffix '-' followed by: B to exclude break information D to exclude the log date T to exclude the log time M to exclude to log milliseconds G to not Group 'On Change' data together
file	integer	Number of file to export where 0 is the latest (active) file, 1 is the previous file etc.
outputfile	string	File name for output file. May include full path, which will be created automatically if it does not exist.

All these arguments are optional, and may be omitted provided there are no further arguments i.e. to specify the 'format', 'datasetname' and 'item list' must be included but 'file' and 'output' may be omitted.

Typical Examples

```
ExportLog("Balloon", "*")
```

or

```
ExportLog("Balloon",
"Altitude,Fuel,Burning,Lift,Group  1"  "CSV-BDTM",  0,
"output")
```

or

```
ItemList(0) = "Altitude"
ItemList(1) = "Fuel"
ItemList(2) = "Burning"
ItemList(3) = "List"
ItemList(4) = "Group 1"
ExportAndViewLog("Balloon",  ItemList,  "CSV-BDTM",  0,
"output")
```

All these commands will export all the data in the specified file, for the named data set to the named output file, in the format specified.

5-16-7 OpenLogFile

Syntax

```
returnstate = OpenLogFile("datasetname")
```

or

```
returnstate = OpenLogFile("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
datasetname	text	Name of Data Set to open as text point or constant.
databaselink	text	Name of Database link to open as text point or constant.

Typical Example

```
OpenLogFile("Balloon")
```

This command will open the log file, ready to start logging. As the function is disk intensive it should not be called frequently.

5-16-8 OpenLogView

Syntax

```
OpenLogView("datasetname", "item list", sessionfile)
```

or

```
OpenLogView("datasetname", TextArray, sessionfile)
```

Remarks

Argument	Type	Description
datasetname	text	Name of Data Set to open as text point or constant.
item list	string	List of Items and/or Groups within the data set to view, separated by commas
TextArray	string array	A text point, which has an array size specified as 1 or more elements. Each element holds an Item or Group name.
sessionfile	string	Optional filename of session information file. The Data Log Viewer is shown with the session settings (e.g. Window position, size, colours, grid options etc. stored in the session file. If omitted, the previous settings are used.

Typical Example

```
OpenLogView("Balloon",
"Altitude,Fuel,Burning,Lift,Group 1")
```

or

```
ItemList (0) = "Altitude"
ItemList (1) = "Fuel"
ItemList (2) = "Burning"
ItemList (3) = "Lift"
ItemList (4) = "Group 1"
OpenLogView("Balloon", ItemList)
```

Both these commands will open the Data Log Viewer, and load the Balloon log file, and show the named items.

```
OpenLogView("Balloon",      ItemList,      "C:\Program
Files\Omron\CX-SUPERVISOR\App\MySessionInfo.txt")
```

This command will open the Data Log Viewer and Balloon log file as above but the Data Log Viewer will always appear in the same position, and with the same settings - not as it was last shown.

5-16-9 StartAuditTrail

Syntax

```
returnstate = StartAuditTrail(ErrorFlag)
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful and the audit trail database is opened and logging started. Otherwise it returns 0.
ErrorFlag	bool	Optional. At some period of time after execution, this flag may be set to 1 if an error occurs.

Typical Example

```
StartAuditTrail(AuditError)
```

This command will start audit trail logging of all items configured to be logged into the audit trail database, based on the chosen target (i.e. Microsoft Access or SQL). By default, data will be appended to the audit trail database if one already exists, otherwise a new database will be created. The 'Audit Trail Configuration' dialog can be used to configure how audit trail data is logged to a Microsoft Access or SQL database.

If at any time any audit instruction fails, for example the remote database becomes disconnected, then "AuditError" is set true and can be used to test or trigger other actions. If AuditError is reset to False then it will automatically be set True again on any further auditing error.

5-16-10 StopAuditTrail

Syntax

```
StopAuditTrail()
```

Typical Example

```
StopAuditTrail()
```

This command will stop the current audit trail logging and close the audit trail database.

5-16-11 StartLogging

Syntax

```
returnstate = StartLogging("datasetname")
```

or

```
returnstate = StartLogging("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0

Argument	Type	Description
datasetname	text	Name of Data Set to open as text point or constant.
databaselink	text	Name of Database link to start logging as text point or constant.

Typical Example

```
StartLogging("Process 1")
```

This command will start logging of all items in the named data set. If the file is closed it will be automatically opened.

5-16-12 StopLogging

Syntax

```
returnstate = StopLogging("datasetname")
```

or

```
returnstate = StopLogging("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
datasetname	text	Name of Data Set to open as text point or constant.
databaselink	text	Name of Database link to stop logging as text point or constant.

Typical Example

```
StopLogging("Process 1")
```

This command will stop logging of all items in the named data set.

5-17 Database Commands

5-17-1 DBAddNew

Description

Adds a new record to a Recordset. This function will fail if the Recordset is opened with a lock of 'Read Only'.

Syntax

```
returnstate = DBAddNew(level)
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This should be a field or recordset level.

Typical Examples

```
Result = DBAddNew("Northwind.Order Details")
```

Using a Recordset connection level, a new record is added with values from all fields associated with a property type 'Add'. Point 'Result' is set true if this was successful.

```
DBAddNew("Northwind.Order Details.OrderID")
DBAddNew("Northwind.Order Details.ProductID")
DBAddNew("Northwind.Order Details.Quantity")
DBAddNew("Northwind.Order Details.UnitPrice")
DBUpdate("Northwind.Order Details")
```

Using a Field connection level, each required field is added to the new record using multiple calls to DBAddNew(). When the record is complete, it is added by calling the DBUpdate() function

Note: To use DBAddNew() with a Recordset level the Recordset must be configured to perform this type of operation i.e. it will need to contain fields for any primary keys and 'non null' values required to create a new record. When used at Recordset level all fields associated with the Recordset with property type 'Add' are added (as if calling DBAddNew()) and the record is updated (as if calling DBUpdate()). Points associated with the 'Add' property can be array points, thus enabling you to add multiple records in one operation.

Note: When using a Field level connection, the operation may be cancelled at any stage before the DBUpdate() function is called by calling the DBExecute() command "CancelUpdate".

Note: Only Fields with a property type of 'Add' can be added to a Recordset. The value(s) of the associated points at the time DBUpdate() is called will be used to create the record.

Note: Depending on the ADO provider, the added record may not be visible until the Recordset is requeried. See DBExecute, parameter Requery for more information.

5-17-2 DBClose

Description

Closes a Connection or Recordset. Closing a Connection will automatically close all recordsets associated with it. Recordsets can be closed in isolation by selecting the appropriate level.

Syntax

```
returnstate = DBClose(level)
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This should be a field or recordset level.

Typical Examples

```
Result = DBClose("Northwind.Order Details")
```

Closes the 'Order Details' Recordset

```
Result = DBClose("Northwind")
```

Closes the connection to the Northwind database, and also any Recordsets which may be open.

5-17-3 DBDelete

Description

Deletes the specified number of records from the current record position. This function works only at the Recordset level. This function will fail if the Recordset is opened with a lock of 'Read Only'.

Syntax

```
returnstate = DBDelete(level, quantity)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This should be a field or recordset level.
quantity	int	Number of records to delete.

Typical Examples

```
Result = DBDelete("Northwind.Order Details", 10)
```

Delete the next 10 records in the recordset

```
DBMove("First")
Result = DBDelete("Northwind.Order Details", 10)
```

Delete the first 10 records.

5-17-4 DBExecute

Description

The DBExecute function allows the execution of miscellaneous commands and allows for future expansion by supporting new commands without the need to create more new DB functions.

Syntax

```
return = DBExecute(level, command, parameter)
```

Remarks

Argument	Type	Description
return		1 if the function is successful otherwise 0 except for "Find" and "FindNext" commands which return the record number if found or if not, set the current record to EOF and return -1.
level	text	A text point or constant specifying the connection level, which depends on the command specified.
command	text	Command to execute. May be one of the commands listed below.
parameter	text	Command parameter only required with certain commands. For "Connection", this parameter should hold the new connection string. For "Find" and "FindNext" this parameter should be the search criteria. For "Source" this is the Recordset source. For "Filter" this is the Recordset filter.

Typical Examples

```
Pos = DBExecute("Northwind.Order Details", "Find",
"UnitPrice > 14.00")
```

Find the next record satisfying the specified criteria, starting from the current position. Valid search criteria include: "ProductName LIKE 'G*' " wildcard search finds all records where ProductName starts with 'G', "Quantity = 5", "Price >= 6.99". Only single search values are allowed, using multiple values with 'AND' or 'OR' will fail.

```
DBExecute("Connection1.Recordset1", "Source",
"Table2")
```

Modify the Recordsets source to open a different table than configured.

```
DBExecute("Northwind.Shippers", "Filter",
"CompanyName = 'United Package'")
```

Apply a filter to display only records with a company name 'United Package'

```
DBExecute("Northwind.Shippers", "Filter", "")
```

Cancel an existing filter (by passing an empty string)

DBExecute Commands

Command	**Connection Level**	**Description**
Connection	Connection	Modify the connection string.
BeginTrans	Connection	Begins a new Transaction.
CommitTrans	Connection	Saves any pending changes and ends the current transaction.
RollbackTrans	Connection	Cancels any changes made and ends the transaction.
CommitTransAll	Connection	Saves all changes and ends all transactions.
RollbackTransAll	Connection	Cancels all changes and ends all transactions.
TransCount	Connection	Returns the number of pending transactions.
Requery	Recordset	Re-run the Recordset Query.
CancelUpdate	Recordset	Cancel a DBAddNew operation.
Find	Recordset	Find the specified criteria in a Recordset.
FindNext	Recordset	Combined DBMove("Next"), DBFind() operation.
Source	Recordset	Modify the Recordset source.
Filter	Recordset	Apply a filter to a Recordset.
Save	Recordset	Saves a Recordset in XML format.
SQL	Connection or Recordset	Valid SQL text to execute on this connection level.

5-17-5 DBGetLastError

Description

Returns the last error string generated by the Database provider, and displays it in a message box.

Syntax

```
returnstate = DBGetLastError(level, display)
```

Remarks

Argument	Type	Description
returnstate	text	The error message from the provider
level	text	A text point or constant specifying the connection level. This must be a Connection level.
display	bool	Optional flag. By default DBGetLastError will display the providers error message in a message box. Setting this flag to FALSE prevents this action.

Typical Examples

```
DBGetLastError("Northwind")
```

or

```
DBGetLastError("Northwind", TRUE)
```

Both the above lines will get and display the last error to occur for the Northwind connection.

```
ErrMsg = DBGetLastError("Northwind", FALSE)
```

The last error to occur for the Northwind connection is stored Text point 'ErrMsg', without displaying a message box.

5-17-6 DBMove

Description

The DBMove function enables you to navigate around a Recordset by moving the position of the 'current record' in the Recordset. When a Recordset is first opened the first record is the current record.

Syntax

```
returnstate = DBMove(level, direction, position)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.

Argument	Type	Description
direction	text	A text string indicating where to move to. May be one of: "First" "Last" "Next" "Previous" "Position" "FirstPage" "LastPage" "NextPage" "PreviousPage" "Page" "Bookmark"
position	int/real	This optional parameter is only required when directions of "Position", "Page" and "Bookmark" are used. When used with "Position" and "Page" this parameter must be an integer, and is the record or page number to move to. When used with "Bookmark" this parameter must be a real.

Typical Examples

```
DBMove("Northwind.Order Details", "First")
```

Go to the first record in the Recordset.

```
pos = 3
DBMove("Northwind.Order Details", "Position", pos)
```

Go to the third record in the Recordset.

```
DBMove("Northwind.Order Details", "Page", 6)
```

Go to the sixth page in the Recordset.

Note: Bookmarks are returned from the function 'DBProperty', they enable you to return to a 'marked' record, even after records have been added or deleted

Note: Some Providers do not support moving in the "Previous" direction i.e. cursors are 'Forward-Only'. Some 'Forward-Only' providers do allow moving "First", while some are strictly Forward-Only i.e. the Recordset has to be Re-queried effectively a combined Close then Open operation to reset the cursor back to the start of the Recordset. Some Providers that do support moving "Previous" do not support moving to "Position". However, in order to be consistent, CX-Supervisor ensures that that all operations (except "Bookmarks") will work for any connection to any provider but you need to bear in mind when designing applications that use 'Forward-Only' cursors, that there may be some 'long-winded' acrobatics being performed behind the scenes. See DBSupports() for details of how to check the type of cursor in force.

Note: Bookmarks will only work if specifically supported by the Provider.

5-17-7 DBOpen

Description

Opens a Connection or Recordset. Opening a Connection will automatically open all recordsets associated with it, that are marked as auto open. Recordsets can be opened in isolation by selecting the appropriate level.

Syntax

```
returnstate = DBOpen(level)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.

Typical Examples

```
DBOpen("Northwind")
```

Open the connection to the Northwind database, and automatically open any Recordsets set to open on connection.

```
done = DBOpen("Northwind.Order Details")
```

Just open a specific Recordset.

5-17-8 DBProperty

Description

Returns the requested property. This function operates on the Recordset and Field levels. The type of the value returned depends on the property requested.

Syntax

```
returnstate = DBProperty(level, property)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.
property	text	The name of the property to get. For details see the Recordset Properties and Field Properties tables.

Typical Examples

```
Page = DBProperty("CSV.Result", "CurrentPage")
```

Get the current page for the CSV.Result Recordset.

```
FieldSize = DBProperty("Northwind.Customers.Address",
"Size")
```

Get the size for the 'Address' field.

Note: The Recordset will only return valid properties when it is Open.

Recordset Properties

The properties of a Recordset are:

Property	Description	Return type
"CurrentRecord	"Current cursor position	Integer
"RecordCount	"Number of records in the Recordset.	Integer

Property	Description	Return type
"Bookmark	"Record marker.	Real
"PageCount	"Number of pages in the Recordset.	Integer
"PageSize	"Number of records in a page.	Integer
"CurrentPage	"Page in which the cursor position resides.	Integer
"Source	"Command or SQL that created the Recordset.	Text
"Sort	"Field name(s) the Recordset is sorted on.	Text
"FieldCount	"Number of fields(columns) in the Recordset.	Integer
"BOF	"Current position is at the start of the Recordset.	Bool
"EOF	"Current position is at the end of the Recordset.	Bool

Field Properties

The properties of a Field are

Property	Description	Return type
"Value	"Value of the field at the current position.	As type of field
"Name	"Name of the Field.	String
"Type	"The fields data type.	String
"Size	"Maximum width of the field.	Integer

5-17-9 DBRead

Description

Reads a record from a Recordset to the associated point(s), or if associated points are array points, reads a whole page of records. This function operates on both Recordset and Field levels. At the Field level the associated column values from the Recordsets current position will be copied into the Point (number of elements copied = number of elements in the Point, no paging applies at the Field level).

Syntax

```
returnstate = DBRead(level, reset)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.
reset	bool	This argument is optional and may be omitted. If omitted or TRUE, when the read is complete the record cursor is reset to the position prior to reading.

Typical Examples

```
DBRead("Northwind.Customers")
```

Read the next page of records from the 'Customers' Recordset.

```
DBRead("Northwind.Customers", FALSE)
```

Read the next page of records from the 'Customers' Recordset, and leave the cursor at the next record.

```
DBRead("Northwind.Customers.Address")
```

The Address field is read. If it is an array point, the Address is read from subsequent records until the array has been filled.

Note: Use with reset = TRUE is useful if the read operation is being combined with a subsequent Write operation i.e. you can read in a set of records - resetting the cursor, make modifications to some of the fields and then Write the changes back to the Recordset.

Note: Use with reset = FALSE will leave the current position at the start of the next set of records. This option can be of benefit if the Provider only supports forward moving cursors, or you simply want to step through the records a page at a time.

5-17-10 DBSchema

Description

Issues commands to read schema results or properties or set up new schema criteria. This function operates only at a Schema level.

Syntax

```
return = DBSchema(level, command, parameters...)
```

Remarks

Argument	Type	Description
return		Value returned by command. For some commands e.g. "RecordCount" this is an integer value, for other commands this is a text value.
level	text	A text point or constant specifying the connection level. This must be a Schema level.
command	text	The command must be one of the following: • "Read" - Transfers a schema page into the associated point • "Set" - Enables schema details to be modified • "Type" - Returns the current Schema Type • "Criteria"- Returns the current Schema Criteria • "Filter" - Returns the current Schema Filter • "RecordCount" - Returns the number of records in the current Schema • "PageCount" - Returns the number of pages in the current Schema • "CurrentPage" - Returns the current Schema page

Argument	Type	Description
parameters		Some commands require 1 or more extra parameters. "Read" takes an optional parameter 'Page Number' of type integer. If no 'Page Number' is supplied, this function will return page 1 when first called and automatically return the next page of schemas for each subsequent call, cycling back to the beginning when all pages have been returned. "Set" takes three text parameters for Schema 'Name', 'Criteria' and 'Filter'.

Typical Examples

```
NumberOfRecords = DBSchema("Invoice.Data Types", "RecordCount")
```

Read the Number of records in the Schema.

```
DBSchema("Invoice.Data types", "Read", 2)
```

Read Schema page 2 results into the associated point.

```
DBSchema("Invoice.Data Types", "Set", "Columns", "COLUMN_NAME", "")
```

Set a new Schema to return column names.

5-17-11 DBState

Description

Reports if the specified level is in the requested state.

Syntax

```
return = DBState(level, state)
```

Remarks

Argument	Type	Description
return	bool	1 if the specified level is in the requested state, otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Connection or Recordset level.
state	text	The requested state must be either "Open" or "Closed"

Typical Examples

```
State = DBState("Invoice", "Closed")
```

Checks if the Connection "Invoice" is currently closed.

```
State = DBState("Northwind.Customers", "Open")
```

Checks if the Recordset "Customers" is currently open.

5-17-12 DBSupports

Description

Returns TRUE if the specified Recordset supports the requested operation.

Syntax

```
return = DBSupports(level, operation)
```

Remarks

Argument	Type	Description
return	bool	1 if the specified level is in the requested state, otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Connection or Recordset level.
operation	text	The requested operation may be one of: "AddNew" "Bookmark" "Delete" "Find" "MovePrevious" "Update"

Typical Example

```
Result = DBSupports("CSV.Recordset1", "Delete")
```

Checks if records can be deleted in 'Recordset1'

Note: If the "MovePrevious" operation is not supported then only 'Forward-Only' cursor movements are supported.

5-17-13 DBUpdate

Description

Update the record being added in a Recordset. Used in conjunction with DBAddNew to commit a new record.

Note: DBUpdate is ONLY required when DBAddNew has been used at the Field level. When DBAddNew is used at the Recordset level an additional DBUpdate is not required as this is performed automatically.

Syntax

```
returnstate = DBUpdate(level)
```

Remarks

Argument	Type	Description
return	bool	1 if the specified level is in the requested state, otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Connection or Recordset level.

Typical Example

```
DBAddNew("Northwind.Order Details.OrderID")
DBAddNew("Northwind.Order Details.ProductID")
DBAddNew("Northwind.Order Details.Quantity")
DBAddNew("Northwind.Order Details.UnitPrice")
DBUpdate("Northwind.Order Details")
```

Each required field is added to the new record using multiple calls to DBAddNew(). When the record is complete, it is added to the Recordset by calling the DBUpdate() function.

5-17-14 DBWrite

Description

Writes a set of records into a Recordset from the associated point(s). This function operates on both Recordset and Field levels. At the Recordset level all the associated points values from the Points will be written into the Recordset starting at the current record (1 page of values will be written for each Point). At the Field level the associated values from the point are written into the Recordsets starting at the current position. The number of elements written = number of elements in the Point. This function will fail, if the Recordset is opened with a Lock of 'Read Only'.

Syntax

```
return = DBWrite(level, reset)
```

Remarks

Argument	Type	Description
return	bool	1 if the specified level is in the requested state, otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Connection or Recordset level.
reset	bool	This argument is optional and may be omitted. If omitted or TRUE, when the write is complete the record cursor is reset to the position prior to writing.

Typical Examples

```
DBWrite("Northwind.Customers")
```

Write all point values to the associated Customers fields.

```
DBWrite("Northwind.Customers.Address", FALSE)
```

Write the point values to the Address column, and leave the cursor at the next set of records.

5-18 Serial Port Functions

5-18-1 InputCOMPort

Description

Sets the serial communications port for receiving ASCII text messages. Any message received is placed in the text point. The boolean flag is set true to indicate that a message has been received. It is up to the user to reset this flag between receiving messages in order to indicate that a new message is present. This function need only be called once to receive multiple messages every time the termination character is recieved.

Syntax

```
ReturnState = InputCOMPort(PortNumber, Message,
MessagePresent, MaxLength)
```

Remarks

Argument	Type	Description
ReturnState	bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort and opened with OpenCOMPort.

Argument	Type	Description
message	Text	Text point to hold ASCII text message received through the port.
MessagePresent	Bool	Boolean point indicating that a message has been received.
MaxLength	Integer	Optional. Maximum length of transmission before input is terminated. Used where fixed length packets are received without termination characters.

Typical Example:

```
bState = InputCOMPort(1, Msg, bTransmission)
```

5-18-2 OutputCOMPort

Description

Sends an ASCII text message out through the designated serial communications port.

Syntax

```
ReturnState = OutputCOMPort(PortNumber, Message)
```

Remarks

Argument	Type	Description
ReturnState	bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort and opened with OpenCOMPort.
message	Text	Text point to hold ASCII text message to send through the port.

Typical Example:

```
bState = OutputCOMPort(1, Msg)
```

5-18-3 CloseCOMPort

Description

Closes the designated serial communications port on the PC. The port must have been configured and opened before it can be closed.

Syntax

```
ReturnState = CloseCOMPort(PortNumber)
```

Remarks

Argument	Type	Description
ReturnState	bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort and opened with OpenCOMPort.

Typical Example:

```
bState = CloseCOMPort(1)
```

5-18-4 OpenCOMPort

Description

Opens the designated serial communications port on the PC for transmitting or receiving data. The port must have been configured before it can be opened.

Syntax

```
ReturnState = OpenCOMPort(PortNumber)
```

Remarks

Argument	Type	Description
ReturnState	bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort.

Typical Example:

```
bState  = OpenCOMPort(1)
```

5-18-5 SetupCOMPort

Description

Configures the designated serial communications port on the PC for transmitting or receiving data.

Syntax

```
ReturnState = SetupCOMPort(PortNumber,
ConfigurationString, HandShaking, TerminationChar,
ControlCharFlag, TermMode)
```

Remarks

Argument	Type	Description
ReturnState	bool	True if successful else false.
PortNumber	Integer	A string indicating the desired Baud rate, Parity, number of data bits and stop bits.
HandShaking	Integer	The required handshaking protocol. Valid values are 0 - None 1 - XonXoff 2 - RTS 3 - RTS & XonXoff
TerminationChar	Integer	A character indicating the end of the message.
ControlCharFlag	Bool	A flag indicating that control characters contained in a received message should be Ignored.

Argument	Type	Description
TermMode	Integer	Optional. Flags to indicate how to use the termination character @ONINPUT (or value 1) - Function InputComPort expects Termination Character. This is the default value if omitted. @ONOUTPUT (or value 2) -Function OutputComPort appends Termination Character. @ONINPUT | @ONOUTPUT (or value 3) - both of the above.

Typical Example:

```
bState  = SetupCOMPort(2, "9600,N,8,1", 0, 0x0D, TRUE)
```

5-19 ActiveX Functions

Reading and writing property values from ActiveX objects, and executing methods, is achieved easily using the Dot (.) syntax with the object name.

5-19-1 Getting a property value

Syntax

```
propertyvalue = object.property
or
propertyvalue = object.property(...)
```

Remarks

Argument	Type	Description
propertyvalue	n/a	The value of the property. Type is dependent on the type of the property.
object	Text	The name of the ActiveX object to get the property of.
property	Text	The name of the property to get.
---	n/a	Any number of parameters for the property.

Typical Examples

```
OLE1Height = OLE1.Height
```

This will read the property 'Height' from the ActiveX object 'OLE1' and store it in the point 'OLEHeight'.

```
DM100Value = CXComms1.DM(100)
```

This will read the property 'DM' (with one parameter 100) from the ActiveX object 'CXComms1' and store it in the point 'DM100Value'.

5-19-2 Writing a property value

Syntax

```
object.property = value
or
object.property(...) = value
```

Remarks

Argument	Type	Description
object	Text	The name of the ActiveX object containing the property to change.
property	Text	The name of the property to put.
`---`	n/a	Any number of parameters for the property.
value	n/a	The value to write to the property. Type is dependent on the type of property. Can also be a number.

Typical Examples

```
OLE1.Left = NewLeftValue
```

This will write the value stored in the point NewLeftValue to the property 'Left' in the ActiveX object 'OLE1'.

```
CXComms1.DM(10) = NewValue
```

This will write the value stored in the point NewValue to the property 'DM' (with one parameter 10) in the ActiveX object 'CXComms1'.

```
Gauge1.Value = 25.2
```

This will write the value 25.2 to the ActiveX object 'Gauge1'.

5-19-3 Executing a method

Syntax

```
object.method
or
object.method(...)
```

Remarks

Argument	Type	Description
object	Text	The name of the ActiveX object.
method	Text	The name of the method to execute.
`---`	n/a	Any number of parameters for the method.

Typical Examples

```
OLE1.Start
```

This will call the method 'Start' on the ActiveX object 'OLE1'.

```
CXComms1.OpenPLC("MyPLC")
```

This will call the method 'OpenPLC' with one text parameter 'MyPLC' on the ActiveX object 'CXComms1'

5-19-4 Responding to events

Some ActiveX components are written to generate events on certain conditions, like mouse clicking or user input or error conditions. You can write a script to execute whenever any event occurs. These scripts are defined as subroutines in the page initialisation script as they may be called any time the page is open. To easily add these subroutines, from the ActiveX Property Browser, click the ‘Events’ tab. This shows all the event types for this control and any parameters the event may pass. Select the event name, to add or edit the script for, and click the square edit button.

5-19-5 ExecuteVBScriptFile

Description

Allows Visual Basic script stored in a text file to be executed. This uses the windows scripting host which must be installed. See chapter 5 for a list of supported functions.

Syntax

```
returnstate = ExecuteVBScriptFile(scriptfile)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
scriptfile	Text	The name of the file with the Visual Basic Script to execute.

Typical Examples

```
returnstate = ExecuteVBScriptFile("c:\vbscript.txt")
```

This will execute the Visual Basic Script stored in "c:\vbscript.txt".

Note: Functions that run dynamic script on the target system could be abused as part of a cyber attack. To restrict the scope, limit the parameters to hard coded static strings with fully qualified path and drive letter. Alternatively if a variable parameter is used, recognise this variable could be tampered at Runtime and therefore include sufficient processing and parsing before use to ensure it is limited and restricted to the expected drives and paths

5-19-6 GenerateEvent

Description

This command is only used in conjunction with a remote connection using a CX-Supervisor Communications control (see User Manual Chapter 22, Connecting to remote applications). This command allows the Server machine to post unsolicited data back to the client machine. This data is captured in the client's "OnEvent" handler.

The data for the parameters is entirely at the designer's discretion, depending on what the client needs to be informed of.

Syntax

```
returnstate = GenerateEvent(param1, param2, param3)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
param1	Text	Optional. Parameter of data to send
param2	Text	Optional. Parameter of data to send
param3	Text	Optional. Parameter of data to send

Typical Examples

```
returnstate = GenerateEvent ("Archive", "", "")
```

An 'Archive' event is sent to the client application that may force the client to perform some specified archive operation. The second and third parameters are not used.

```
returnstate = GenerateEvent ("[Alarm Set]", "Boiler
alarm", "95.5")
```

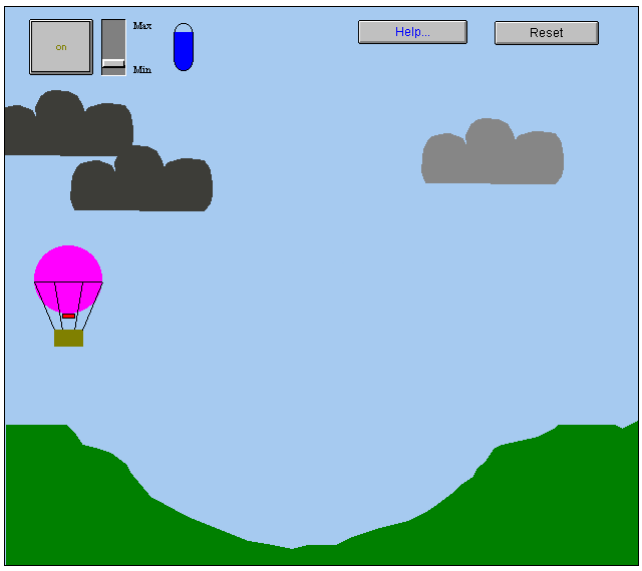
An event is sent to the client application which can be interpreted as 'The Boiler alarm has been set with a process value of 95.5'.

SECTION 6
Script Example

This chapter provides an example application for a script. The script is a typical script exercising the basic commands. It is described twice, once as a whole, and once on a line by line basis.

6-1 Balloon Script

The following script applies to a simple game.



The user must attempt to land the balloon on the plateau on the right, using the Max/Min slider control throughout the flight. Clicking Reset clears the current game and initialises a new game. Clicking the on/off pushbutton starts the game.

When the balloon is airborne, clouds move slowly horizontally and change colour slightly. Clicking Help at any time brings up a special help page; clicking Close from this help page returns the user to the game. The blue gauge shows the amount of fuel consumed and left.

The project consists of three page scripts and one object. The three page scripts are initiated at varied intervals: 10 milliseconds, 100 milliseconds and 1000 milliseconds.

The page script initiated at intervals of 10 milliseconds determines the position of each cloud, and the speed at which each cloud moves. The page script initiated at intervals of 1000 milliseconds determines how the balloon reacts to the conditions.

The page script initiated at intervals of 100 milliseconds provides the main configuration of the game, reacting to user input and moving the balloon accordingly. This page script is as follows:

```
IF burner AND alt > 400.0 THEN
    burner = FALSE
END IF
IF burner THEN
    fuel = fuel - rate
    IF fuel < 0.0 THEN
        fuel = 0.0
        burner = FALSE
```

```
    END IF
END IF

IF burner AND fuel > 0.0 AND rate > 0.0 THEN
    lift = lift + rate/5.0
ELSE
    IF alt > 140.0 THEN
        lift = lift - 0.2
    END IF
END IF
IF lift < -10.0 THEN
    lift = -10.0
END IF
alt = alt + lift
IF alt <= 140.0 THEN
    IF distance>630.0 AND distance<660.0 AND lift>=-
    3.0 THEN
        winner = TRUE
        burner = FALSE
    END IF
    IF lift < -3.0 then
        crash = TRUE
        burner = FALSE
    END IF
    lift = 0.0
END IF

speed = (alt-140.0 )/100.0
IF speed < 0.0 THEN
    speed = 0.0
END IF

distance = distance + speed
```

The following paragraphs describe the above script on a line by line basis.

```
IF burner AND alt > 400.0 THEN
    burner = FALSE
END IF
```

If the fuel burner is on, based on Boolean point 'burner' set to 'TRUE', and the altitude of the balloon, based on point 'alt', exceeds 400, then the fuel burner is turned off. Point 'alt' is measured in pixels between 140 and 1000, so the value of 400 is the height in pixels.

```
IF burner THEN
    fuel = fuel - rate
    IF fuel < 0.0 THEN
        fuel = 0.0
        burner = FALSE
    END IF
END IF
```

If the fuel burner is on, the amount of fuel remaining decreases by the rate of ascent. The rate of ascent, point 'rate' can be modified by moving the slider. If point 'fuel' currently has a value of less than 0, then there is no fuel left and the fuel burner is turned off.

```
IF burner AND fuel > 0.0 AND rate > 0.0 THEN
    lift = lift + rate/5.0
ELSE
```

```
        IF alt > 140.0 THEN
            lift = lift - 0.2
        END IF
    END IF
```

If the fuel burner is on, and there is still fuel left, and the rate of ascent exceeds 0 (the balloon has taken off) then point 'lift' is incremented by the rate of ascent divided by 5 to allow the balloon to climb. Otherwise the balloon must be descending and point 'lift' is decremented by 0.2.

```
IF lift < -10.0 THEN
    lift = -10.0
END IF
```

Once point 'lift' reaches -10, it is not allowed to go lower.

```
alt = alt + lift
```

The altitude of the balloon is incremented by point 'lift'.

```
IF alt <= 140.0 THEN
    IF distance>630.0 AND distance<660.0 AND lift>=-
    3.0 THEN
        winner = TRUE
        burner = FALSE
    END IF
```

If the balloon has hit the ground (point 'alt' equals 140), then provided it is on the plateaux (the position of the balloon in pixels defined by point 'distance' is between 630 and 660) and the rate of descent is not too fast (defined by point 'lift'), then the game is won.

```
  IF lift < -3.0 then
    crash = TRUE
    burner = FALSE
  END IF
```

If the balloon has hit the ground (point 'alt' equals 140), then if the rate of descent is not too fast (defined by point 'lift'), then the game is lost.

```
lift = 0.0
END IF
```

Point 'lift' is reset.

```
speed = (alt-140.0 )/100.0
IF speed < 0.0 then
    speed = 0.0
END IF
```

Point 'speed' is calculated based on the altitude.

```
distance = distance + speed
```

Point 'distance' is calculated based on the speed.

SECTION 7
Colour Palette

CX-Supervisor uses a colour palette whereby colours are based on ARGB values (Alpha, Red, Green and Blue). The Alpha value determines the transparency factor whereby 0 = 0% (fully transparent) and 255 = 100% (fully opaque). Within script, colours can be referenced in the following ways:

- By their name, for example:
  Colour “PageName”, “ObjectName”, **“Red”**

- By their ARGB value, for example:
  Colour “PageName”, “ObjectName”, **&HFFFF0000**

NOTE: the examples above show the VBScript syntax, whereby the colour names must be specified in quotes and the ARGB values must be prefixed with ‘**&H**’ and not ‘**0x**’. The equivalent code in CX-Supervisor script would be as follows:

objectname.colour = Red
objectname.colour = 0xFFFF0000

CX-Supervisor supports the following colours:

AliceBlue = 0xFFF0F8FF
AntiqueWhite = 0xFFFAEBD7
Aqua = 0xFF00FFFF
Aquamarine = 0xFF7FFFD4
Azure = 0xFFF0FFFF
Beige = 0xFFF5F5DC
Bisque = 0xFFFFE4C4
Black = 0xFF000000
BlanchedAlmond = 0xFFFFEBCD
Blue = 0xFF0000FF
BlueViolet = 0xFF8A2BE2
Brown = 0xFFA52A2A
BurlyWood = 0xFFDEB887
CadetBlue = 0xFF5F9EA0
Chartreuse = 0xFF7FFF00
Chocolate = 0xFFD2691E
Coral = 0xFFFF7F50
CornflowerBlue = 0xFF6495ED
Cornsilk = 0xFFFFF8DC
Crimson = 0xFFDC143C
Cyan = 0xFF00FFFF
DarkBlue = 0xFF00008B
DarkCyan = 0xFF008B8B
DarkGoldenrod = 0xFFB8860B

DarkGray = 0xFFA9A9A9
DarkGreen = 0xFF006400
DarkKhaki = 0xFFBDB76B
DarkMagenta = 0xFF8B008B
DarkOliveGreen = 0xFF556B2F
DarkOrange = 0xFFFF8C00
DarkOrchid = 0xFF9932CC
DarkRed = 0xFF8B0000
DarkSalmon = 0xFFE9967A
DarkSeaGreen = 0xFF8FBC8B
DarkSlateBlue = 0xFF483D8B
DarkSlateGray = 0xFF2F4F4F
DarkTurquoise = 0xFF00CED1
DarkViolet = 0xFF9400D3
DeepPink = 0xFFFF1493
DeepSkyBlue = 0xFF00BFFF
DimGray = 0xFF696969
DodgerBlue = 0xFF1E90FF
Firebrick = 0xFFB22222
FloralWhite = 0xFFFFFAF0
ForestGreen = 0xFF228B22
Fuchsia = 0xFFFF00FF
Gainsboro = 0xFFDCDCDC
GhostWhite = 0xFFF8F8FF
Gold = 0xFFFFD700
Goldenrod = 0xFFDAA520
Gray = 0xFF808080
Green = 0xFF008000
GreenYellow = 0xFFADFF2F
Honeydew = 0xFFF0FFF0
HotPink = 0xFFFF69B4
IndianRed = 0xFFCD5C5C
Indigo = 0xFF4B0082
Ivory = 0xFFFFFFF0
Khaki = 0xFFF0E68C
Lavender = 0xFFE6E6FA
LavenderBlush = 0xFFFFF0F5
LawnGreen = 0xFF7CFC00
LemonChiffon = 0xFFFFFACD
LightBlue = 0xFFADD8E6
LightCoral = 0xFFF08080
LightCyan = 0xFFE0FFFF
LightGoldenrodYellow = 0xFFFAFAD2
LightGray = 0xFFD3D3D3
LightGreen = 0xFF90EE90

LightPink = 0xFFFFB6C1
LightSalmon = 0xFFFFA07A
LightSeaGreen = 0xFF20B2AA
LightSkyBlue = 0xFF87CEFA
LightSlateGray = 0xFF778899
LightSteelBlue = 0xFFB0C4DE
LightYellow = 0xFFFFFFE0
Lime = 0xFF00FF00
LimeGreen = 0xFF32CD32
Linen = 0xFFFAF0E6
Magenta = 0xFFFF00FF
Maroon = 0xFF800000
MediumAquamarine = 0xFF66CDAA
MediumBlue = 0xFF0000CD
MediumOrchid = 0xFFBA55D3
MediumPurple = 0xFF9370DB
MediumSeaGreen = 0xFF3CB371
MediumSlateBlue = 0xFF7B68EE
MediumSpringGreen = 0xFF00FA9A
MediumTurquoise = 0xFF48D1CC
MediumVioletRed = 0xFFC71585
MidnightBlue = 0xFF191970
MintCream = 0xFFF5FFFA
MistyRose = 0xFFFFE4E1
Moccasin = 0xFFFFE4B5
NavajoWhite = 0xFFFFDEAD
Navy = 0xFF000080
OldLace = 0xFFFDF5E6
Olive = 0xFF808000
OliveDrab = 0xFF6B8E23
Orange = 0xFFFFA500
OrangeRed = 0xFFFF4500
Orchid = 0xFFDA70D6
PaleGoldenrod = 0xFFEEE8AA
PaleGreen = 0xFF98FB98
PaleTurquoise = 0xFFAFEEEE
PaleVioletRed = 0xFFDB7093
PapayaWhip = 0xFFFFEFD5
PeachPuff = 0xFFFFDAB9
Peru = 0xFFCD853F
Pink = 0xFFFFC0CB
Plum = 0xFFDDA0DD
PowderBlue = 0xFFB0E0E6
Purple = 0xFF800080
Red = 0xFFFF0000

RosyBrown = 0xFFBC8F8F
RoyalBlue = 0xFF4169E1
SaddleBrown = 0xFF8B4513
Salmon = 0xFFFA8072
SandyBrown = 0xFFF4A460
SeaGreen = 0xFF2E8B57
SeaShell = 0xFFFFF5EE
Sienna = 0xFFA0522D
Silver = 0xFFC0C0C0
SkyBlue = 0xFF87CEEB
SlateBlue = 0xFF6A5ACD
SlateGray = 0xFF708090
Snow = 0xFFFFFAFA
SpringGreen = 0xFF00FF7F
SteelBlue = 0xFF4682B4
Tan = 0xFFD2B48C
Teal = 0xFF008080
Thistle = 0xFFD8BFD8
Tomato = 0xFFFF6347
Transparent = 0x00FFFFFF
Turquoise = 0xFF40E0D0
Violet = 0xFFEE82EE
Wheat = 0xFFF5DEB3
White = 0xFFFFFFFF
WhiteSmoke = 0xFFF5F5F5
Yellow = 0xFFFFFF00
YellowGreen = 0xFF9ACD32

For compatibility with previous versions of CX-Supervisor the following names are also supported:

dark_blue
dark_green
blue_green
dark_grey
light_grey
pale_green
light_blue
off_white
grey
cherry
apple

Appendix A
OPC Communications Control

This appendix contains a list of the available component properties and gives details of the Visual Basic script interface. These properties can be set in run time by using a Visual Basic script command - for example: -

OMRONCXOPCCommunicationsControl1.ServerNodeName = "\\NAME"

The Script Interface defines the Visual Basic script interface for the OPC communications control. See ExecuteVBScript script functions for more information on running Visual Basic Script.

A.1 Component Properties

Property Title	Example	Description
DisplayErrors	True False	When set True, the object will display a message box for any errors. If set to False, error messages are not displayed.
ProjectName		Name of .OPC file containing the client setup.
ServerComputerName	"MyPC	"This is the name of the PC with the OPC Server.
ServerName		Name of the OPC Server to connect to. e.g. OMRON.OpenDataServer.1
ServerProjectName		Optional filename, which if specified causes the OPC Server to use the specified file, if supported by the server.

A.2 Script Interface

The Script Interface defines the methods for the OPC communications control.

A.3 Functions

Value	Function for getting and setting an OPC item value.
Read	Function to read the value of an OPC item.
Write	Function to write the value of an OPC item.

A.3.1 Value

Reads or writes the value of an OPC item.

Example 1 - Reading a value:

```
intVal =
 OMRONCXOPCCommunicationsControl1.Value
("MyGroup", "BoilerTemp")
```

In this example, the OPC item 'BoilerTemp' in the OPC group called "MyGroup" will be read from the OPC Server and will be stored in 'intVal'.

Example 2 - Writing a value:

```
OMRONCXOPCCommunicationsControl1.Value("MyGroup",
"BoilerTemp") = 50
```

In this example, the value 50 will be written to the OPC item 'BoilerTemp'.

Note: 'Value' is the default property so is assumed if omitted. Therefore, the following examples are the same:

```
intVal =
OMRONCXOPCCommunicationsControl1.Value("MyGroup",
"BoilerTemp")
```

and

```
intVal = OMRONCXOPCCommunicationsControl1 ("MyGroup",
"BoilerTemp")
```

A.3.2 Read

Reads the value of an OPC item.

Example of synchronous read:

```
intVal =
 OMRONCXOPCCommunicationsControl1.Read
("MyGroup", "BoilerTemp")
```

In this example, the OPC item 'BoilerTemp' in the OPC group called "MyGroup" will be read from the OPC Server and will be stored in 'intVal'. The script will wait for the read operation to complete before continuing to execute the next line. This is identical to the operation of the 'Value' method.

A.3.3 Write

Writes the value of an OPC item.

Example of synchronous write:

```
OMRONCXOPCCommunicationsControl1.Write
"MyGroup", "BoilerTemp", NewValue
```

In this example, 'NewValue' will be written to the OPC item 'BoilerTemp' in the OPC group called "MyGroup". The script will wait for the write operation to complete before continuing to execute the next line. This is identical to the operation of the 'Value' method.

Appendix B
CX-Server Communications Control

When the Project Settings->Advanced settings option "Allow advanced script access to PLC via 'CXServer' control" option is selected a CX-Server Communications Control is automaticalled created to allow script access to CX-Server functions. This ActiveX control is always named 'CXServer' (without any hyphen) and can always be used from any script.

This appendix contains a list of the available component properties and methods on the script interface.

B.1 Functions

Value	Function for getting and setting an area of memory in a PLC. This function allows logical names to be used. If an array is used, the first element is returned.
Values	Function for getting and setting an area of memory in a PLC. This function allows logical names to be used. If an array is used then a SAFEARRAY is returned with all values.
SetDefaultPLC	Function for setting the default PLC. This is primarily used when a project contains multiple PLCs.
OpenPLC	Opens the specific PLC for communications.
ClosePLC	Closes the specific PLC.
Read	Function to read the value of a PLC point
Write	Function to write the value of a PLC point
ReadArea	Function for reading a block of memory from the PLC.
WriteArea	Function for writing a block of memory to the PLC.
RunMode	Function for reading / writing the current mode of the PLC.
TypeName	Function for reading the PLC type (e.g. CQM1H).
IsPointValid	Checks a point name is valid.
PLC Memory Functions	A, AR, C, CIO, D, DM, DR, E, EM, G, GR, H, IR, LR, SR, ST, T, TC, TK, W. Functions for getting and setting the memory areas in the PLC.
ListPLCs	Property. Holds a list of all PLC names configured in the project file. This property is read only
ListPoints	Property. Holds a list of all point names configured in the project file. This property is read only.
IsBadQuality	Checks whether a point is currently indicating "bad quality".
ClockRead	Reads the PLC Clock
ClockWrite	Sets the PLC Clock
RawFINS	Function that enables raw FINS commands to be sent to a specified PLC.

Active	Function for returning the connection status of a specified PLC.
TCGetStatus	Function for returning the device status of a specified temperature controller
TCRemoteLocal	Function for switching a specified temperature controller into Remote or Local mode
SetDeviceAddress	Sets PLC Network, Node, and Unit number and IP address
SetDeviceConfig	Sets any element of device configuration
GetDeviceConfig	Gets any element of device configuration
UploadProgram	Uploads a program from a PLC
DownloadProgram	Downloads a program to a PLC
Protect	Protects (or releases protection on) program memory
LastErrorString	Description of last error that occurred

B.2 Value

Reads the value of an address from a PLC, or writes a value to an address in a PLC. This function allows logical names.

Example 1 - Reading a value from the PLC using a logical name.

```
intVal = CXServer.Value("BoilerTemp")
```

or

```
intVal = CXServer ("BoilerTemp")
```

In these examples, the PLC address associated with 'BoilerTemp' will be read from the PLC and stored in 'intVal'. "Value" is the default property and does not have to be specified.

Example 2 - Writing a value to the PLC using a logical name.

```
CXServer.Value("BoilerTemp") = 50
```

or

```
CXServer ("BoilerTemp") = 50
```

In these examples, the value 50 will be written to the PLC address associated with 'BoilerTemp'. "Value" is the default property and does not have to be specified.

B.3 Values

Reads an array of values from a PLC, or writes an array of values to a PLC. This function allows logical names. If an array is used then a SAFEARRAY is returned with all values.

Example 1 - Reading an array of values from the PLC using a logical name.

```
SomeArray  = CXServer.Values("BoilerTemps")
```

Example 2 - Writing an array of values to the PLC using a logical name.

```
CXServer.Values("BoilerTemps") = SomeArray
```

B.4 SetDefaultPLC

The 'SetDefaultPLC' function can be used to inform the script parser that a particular PLC is has been set as the default. Once a default PLC has been set, then it is not necessary (with some functions) to specify a PLC name. For example,

```
CXServer.SetDefaultPLC("MyPLC")
intVal = CXServer.Value("BoilerTemp1")
CXServer.Value("BoilerTemp1") = 75
intVal = CXServer.Value("DM50")
```

Each 'Value' function above will access data in the PLC called 'MyPLC'.

Note: If there is only 1 PLC in the project then it is not necessary to call the 'SetDefaultPLC' function. The first PLC in a project will automatically be set as the default PLC.

B.5 OpenPLC

Opens a PLC for communications. If no PLC is specified then the default PLC is opened.

Example 1:

```
CXServer.SetDefaultPLC("MyPLC")
CXServer.OpenPLC()
CXServer.DM(100) = 10
CXServer.DM(50) = 10
```

Example 2:

```
CXServer.OpenPLC("MyPLC")
CXServer.DM(100) = 10
```

B.6 ClosePLC

Closes a previously opened PLC. If no PLC is specified then the default PLC is closed.

Example:

```
CXServer.ClosePLC("MyPLC")
```

B.7 Read

Function to read the value of a PLC point.

Example of synchronous Read

```
intVal = CXServer.Read("MyPLC", "MyPoint", 0)
```

In this example, the Point 'MyPoint' will be read from the PLC 'MyPLC' and stored in 'intVal'. The script will wait for the read operation to complete before continuing to execute the next line due to the '0' parameter. This is identical to the operation of the 'Value' method.

Note: If the PLC is not open, then this command will cause it to be opened, and then closed after the read is complete. If more than one read or write operation is to be performed, it is considerably faster and more efficient to use the OpenPLC command first, do all the reading and writing, and then (if required) use the ClosePLC command to close the PLC.

B.8 Write

Function to write the value of a PLC point.

Example of synchronous write:

```
CXServer.Write("MyPLC", "MyPoint", NewValue, 0)
```

In this example, 'NewValue' will be written to the point 'MyPoint' in the PLC called 'MyPLC'. The script will wait for the write operation to complete before continuing to execute the next line due to the '0' parameter. This is identical to the operation of the 'Value' method.

Note: If the PLC is not open, then this command will cause it to be opened, and then closed after the write is complete. If more than one read or write operation is to be performed, it is considerably faster and more efficient to use the OpenPLC command first, do all the reading and writing, and then (if required) use the ClosePLC command to close the PLC.

B.9 ReadArea

Reads a specified block of memory from a PLC.

Examples of synchronous read:

```
MyVariant = CXServer.ReadArea("MyPLC/DM0", 12, vbString)
MyVariant = CXServer.ReadArea("BoilerTemp",10, vbInteger)
MyVariant = CXServer.ReadArea("BoilerTemp", 20)
```

In the first example, DM0 to DM11 will be read as characters (part of a string) from 'MyPLC' and will be stored in 'MyVariant'. The second example demonstrates that it is also possible to use a logical name for the start address, and that any VB variant types (such as vbInteger) can be used. The third example shows that the VB Variant type parameter is optional - if none is specified then vbInteger is assumed. The script will wait for the read operation to complete before continuing to execute the next line.

Note: If accessing from a CX-Supervisor script, the following integral values should be used for the return type:

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data
vbInteger	2	Integer subtype
vbLong	3	Long subtype
vbSingle	4	Single subtype
vbSingle	5	Double subtype
vbCurrency	6	Currency subtype
vbDate	7	Date subtype
vbString	8	String subtype
vbObject	9	Object
vbError	10	Error subtype
vbBoolean	11	Boolean subtype
vbVariant	12	Variant (used only for arrays of variants)
vbDataObject	13	Data access object
vbDecimal	14	Decimal subtype
vbByte	17	Byte subtype

Constant	Value	Description
vbArray	8192	Array

B.10 WriteArea

Writes a block of memory to a specified area in a PLC.

Examples of synchronous write:

```
MyString = "TestString"
CXServer.WriteArea "MyPLC/DM50", 10, MyString
Dim newValue(2)
newValue(1) = 0
newValue(2) = 1
CXServer.WriteArea "BoilerTemp",2,newValue
```

In the first example, the contents of 'MyString' will be written into DM50 to DM54. Any additional data in 'MyString' will be ignored (i.e. if 'MyString' is 15 characters in length then the first 10 characters will be written to DM50 to DM54 and the remaining 5 characters will be ignored - {Note: each PLC address holds 2 characters}). The second example shows that a logical name can be used. The script will wait for the write operation to complete before continuing to execute the next line.

B.11 RunMode

Reads the current operating mode of a PLC (Stop/Program, Debug, Monitor, Run), where 0=Stop/Program mode, 1=Debug mode, 2=Monitor mode and 4=Run mode.

Example

```
intMode = CXServer.RunMode("MyPLC")
```

In this example, the operating mode would be read from 'MyPLC' and stored in 'intMode'. If 'MyPLC' was in 'Monitor' mode then 'intMode' would be set to the value 2.

B.12 TypeName

Reads the PLC model name of a PLC (e.g. C200H, CQM1H, CVM1 etc).

Example

```
strPLCType = CXServer.TypeName("MyPLC")
```

In this example, the PLC model type will be read from 'MyPLC' and will be stored in 'strPLCType'.

B.13 IsPointValid

Checks if a Point name has been defined in the CX-Server project file.

Examples

```
bValid = CXServer.IsPointValid("MyPoint")
bValid = CXServer.IsPointValid("MyPoint", "MyPLC")
```

In both examples, the boolean variable bValid is set True if the point "MyPoint" has been defined.

B.14 PLC Memory Functions

(A, AR, C, CIO, D, DM, DR, E, EM, G, GR, H, IR, LR, SR, ST, T, TC, TK, W)

All PLC memory functions (e.g. A, AR, D, DM etc.) work in exactly the same way. The following examples use the DM function to get and set the value of a DM address in a PLC.

Example 1

```
intVal = CXServer.DM(100)
```

In this example, the contents of DM100 will be read from the PLC and stored in 'intVal'.

Note: These examples assume there is only 1 PLC in the CX-Server project file, or that the 'SetDefaultPLC' function has been used to select the required PLC. Refer to the 'SetDefaultPLC' function for details about using script with multiple PLCs in the project.

Example 2

```
CXServer.DM(100) = 75
```

In this example, the value 75 will be written to DM100 in the PLC.

Bit addressing, that is accessing data from individual memory bits, is also supported by these memory areas: IR, AR, HR and CIO.

Example 3

```
bVal = CXServer.IR("100.2")
```

In this example, the status of bit IR100.2 (i.e. bit 2 of IR100) will be read from the PLC and stored in 'bVal' (e.g. 'bVal' will be set to TRUE or FALSE).

Example 4

```
CXServer.IR("100.2") = True
```

In this example, bit IR100.2 (i.e. bit 2 of IR100) in the PLC will be set to True. Note that use of the quotes is optional, but is required to differentiate between 100.1 and 100.10

B.15 ListPLCs

Holds a list of all PLC names configured in the project file. This property is read only.

Example

```
Dim arrayOfPLCs
Dim nUbound, nLbound
arrayOfPLCs = CXServer.ListPLCs
nLbound = LBound(arrayOfPLCs)
nUbound = UBound(arrayOfPLCs)
For Count = nLbound To nUbound
    MsgBox arrayOfPLCs(Count)
Next
```

In this example, the list of PLC names in the project configured stored in 'arrayOfPLCs' and then each is displayed in a message box.

B.16 ListPoints

Holds a list of all point names configured in the project file or PLC. This property is read only.

Example

```
Dim arrayOfPoints
Dim nUbound, nLbound
arrayOfPoints = CXServer.ListPoints(sPLC)
nLbound = LBound(arrayOfPoints)
nUbound = UBound(arrayOfPoints)
```

```
For Count = 1 To UBound(arrayOfPoints)
    MsgBox arrayOfPoints (Count)
Next
```

In this example, the list of Points configured for the PLC name specified in text point sPLC is stored in 'arrayOfPoints' and each displayed in a message box.

Example 2

```
arrayOfPoints = CXServer.ListPoints
```

If ListPoints is used without a parameter then points from all PLCs are returned.

B.17 IsBadQuality

Checks whether a point is currently indicating "Bad Quality".

Example

```
Dim bBad
bBad = CXServer.IsBadQuality("MyPLC", "MyPoint")
```

Note: IsBadQuality will return True in situations where the quality is unknown, e.g. where no previous communications with a point has occurred.

B.18 ClockRead

Function that reads the PLC clock

Example

```
Dim NewDate
NewDate  = CXServer.ClockRead("PLC1")
'  dates  can  be  manipulated  via  standard  VBScript
methods (FormatDateTime,  DatePart etc.)
TextBox1 = NewDate      ' this uses a Microsoft Forms
Text Box to convert date to string
TextPoint1  = TextBox1 'this writes the date string to
a CX-Supervisor text point
```

B.19 ClockWrite

Function that sets the PLC clock. The expected format for the date is "dd/mm/yyyy hh:mm:ss".

Example

```
Dim NewDate
'set  time/date  value  here  using  standard  VBScript
methods (Date, Time, Now, CDate etc.)
NewDate = Now   ' This example sets the time to the
current PC time
CXServer.ClockWrite "PLC1",  NewDate
```

B.20 RawFINS

This function enables raw FINS commands to be sent to a specified PLC. This function is for advanced users familiar with the Omron FINS protocol only.

VBScript Example

```
Dim sFINS
Dim sResponse
sFINS = "0501"
sResponse = CXServer.RawFINS(sFins, sPLC)
txtFINSResponse = sResponse 'txtFINSResponse is a CX-
Supervisor point.
```

B.21 Active

Returns the connection status of a specified PLC.

VBScript Example

```
bActive = CXServer.Active("MyPLC") ' bActive is a CX-
Supervisor point
```

In this example, the connected status would be read from 'MyPLC' and stored in CX-Supervisor point 'bActive'. If 'MyPLC' is connected 'bActive' would be set to True.

B.22 TCGetStatus

Return status data for the specified temperature controller.

Example

```
Dim bTCStatusResponse
bTCStatusResponse = CXServer.TCGetStatus("E5AK")
'Heating output is bTCStatusResponse(21)
'Cooling output is bTCStatusResponse(22)
'Alarm 1 output is bTCStatusResponse(23)
'Alarm 2 output is bTCStatusResponse(24)
'Alarm 3 output is bTCStatusResponse(25)
'Stopped status is bTCStatusResponse(28)
'Remote status is bTCStatusResponse(30)
```

In this example, the device status is being read from "E5AK" as an array of bytes. The response from the temperature controller is stored as an array of bytes in bTCStatusResponse.

B.23 TCRemoteLocal

The TCRemoteLocal command will execute the Remote/Local command for the specified temperature controller:

Example - in this example, the "E5AK" device is being set to local mode:

```
'Set the device to local mode
CXServer.TCRemoteLocal "E5AK", 1
```

Example - in this example, the "E5AK" device is being set to remote mode:

```
'Set the device to remote mode
CXServer.TCRemoteLocal "E5AK", 0
```

B.24 SetDeviceAddress

This function can be used to set key elements of a device address (the network number, node number, unit number and Ethernet IP address). The numbers are in the range 0 to 255, with -1 being used to denote "ignore this parameter". This function is for advanced users only.

Note: this method does not interpret or verify the data passed, and it is possible to pass invalid data that will prevent a device communicating. Care should be taken to ensure that all data passed is valid. This method should not be used while a PLC is open and communicating.

Example:

```
NetworkNum = 1
NodeNum = 2
UnitNum = -1
iPAddress = "10.0.0.1"
bValid    =    CXServer.SetDeviceAddress(    "PLC1",
NetworkNum, NodeNum, UnitNum, IPAddress)
```

Note: The return Boolean value, bValid, is set to True if no errors were detected. However, this does not necessarily mean that all the parameters used were valid or appropriate for the PLC being used.

B.25 SetDeviceConfig

This is a function that can be used to set any element of CX-Server device configuration. All the data is passed in textual form. This function is for advanced users only.

Note: This method does not interpret or verify the data passed, and it is possible to pass invalid data that will prevent a device communicating. Care should be taken to ensure that all data passed is valid. This method should not be used while a PLC is open and communicating.

Example:

```
Device = "PLC1"
Section = "NET"
Entry = "IPADDR"
Setting = "10.0.0.1"
bValid = CXServer.SetDeviceConfig  Device,  Section,
Entry, Setting
```

Note: The return Boolean value, bValid, is set to True if no errors were detected. However, this does not necessarily mean that all the parameters used were valid or appropriate for the device being used.

Only the following Section, Entry and Setting parameter value combinations are currently supported:

- Section = "ADDRESS", Entry = "DNA", Setting = "0"..Setting = "255" - this can be used to set the network number
- Section = "ADDRESS", Entry = "DA1", Setting = "0"..Setting = "255" - this can be used to set the node number
- Section = "ADDRESS", Entry = "UNIT", Setting = "0"..Setting = "255" - this can be used to set the unit number
- Section = "ADDRESS", Entry = "IPADDR", Setting = "0.0.0.0"..Setting = "255.255.255.255" - this can be used to set the Ethernet IP address

Other parameter values may work, but should only be used on Omron advice.

B.26 GetDeviceConfig

This is a function that can be used to read any element of the CX-Server device configuration. All the data is passed (and received) in textual form. This function is for advanced users only.

Example:

```
Dim Setting
Device = "PLC1"
Section = "NET"
Entry = "IPADDR"
Setting = CXServer.GetDeviceConfig  Device,  Section,
Entry
```

Currently supported parameter values are as described for the SetDeviceConfig method.

B.27 UploadProgram

The UploadProgram function can be used to read a program from a PLC. The program is read in binary form, and stored in a user-specified file. This function should not be used at the same time as any other PLC communications. The project and PLC will automatically be opened if required. This function is for advanced users only.

Example:

```
Dim SourceFile
Dim DestinationFile
Sourcefile = ""
DestinationFile = "c:\test1.bin"
CXServer.UploadProgram      "PLC1",      SourceFile,
DestinationFile, 1, 0
```

The first parameter is the PLC name.

The second parameter is the source file name. To upload the current program this should be an empty string, but may also be set to the name of a file in the root directory of a memory card, e.g. "Example.obj".

The third parameter is the name of the local file to store the program. A '.bin' file extention is typical for a binary file.

Note: The 4th and 5th parameters are reserved, and should always be 1 and 0 respectively

B.28 DownloadProgram

The DownloadProgram function can be used to write a program to a PLC. This function should not be used at the same time as any other PLC communications. The project and PLC will automatically be opened if required. This function is for advanced users only.

Note: Care should be taken with this function to ensure that the program written is valid for the PLC to which it is downloaded.

Example:

```
bValid       =CXServer.DownloadProgram       "PLC1",
"c:\test2.bin", "", 1, 0
```

The first parameter is the PLC name.

The second parameter is the local source file name. A '.bin' file extention is typical for a binary file.

To download the current program the third parameter should be an empty string, but may also be set to the name of a file to download to the root directory of a memory card, e.g. "Example.obj".

Note: The 4th and 5th parameters are reserved, and should always be 1 and 0 respectively

B.29 Protect

The Protect function can be used to protect (or remove protection from) PLC program memory. This function should not be used at the same time as any other PLC communications. The project and PLC will automatically be opened if required. This function is for advanced users only.

Example 1 (sets protection for CS series PLC)

```
Dim SetProtection
Dim PasswordString
Dim PasswordNumber
EnableProtection = true
```

```
PasswordString = "Password"
PasswordNumber = 0
CXServer.Protect       "PLC1",       EnableProtection,
PasswordString, PasswordNumber
```

Example 2 (unsets protection for C series PLC)

```
Dim SetProtection
Dim PasswordString
Dim PasswordNumber
EnableProtection = false
PasswordString = ""
PasswordNumber = 12345678
CXServer.Protect       "PLC1",       EnableProtection,
PasswordString, PasswordNumber
```

The parameters of this command are, in order:

- PLC - Name of PLC.
- EnableProtection - true to set password protection, false to unset it
- PasswordString - Password as a string. For CS series PLCs this should be a string of up to 8 characters. For CV PLCs this should be a string of up to 8 characters containing a hexadecimal number, e.g. "12345678". For C series PLCs this should be a string of up to 4 characters containing a hexadecimal number, e.g. "1234".
- PasswordNumber - currently this is only used for C and CV series PLCs, and only when the password string is empty. In those circumstances it is simply a number representing the value of the 4 or 8 digit password. Please note that the password is entered in CX-Programmer as a hexadecimal string (as with the PasswordString parameter above), and that, for example, the value 1234 in decimal is the equivalent to "04d2" as a hexadecimal password string.

Additional C Series PLC notes: For C series the PLC program needs code (the first line of the application) in the PLC to enable password setting/release, and this fixes the password value.

```
e.g.  LD AR10.01
FUN49 0 0 #1234  (#1234 - password value in Hex)
```

When setting the password this value is used rather than the value passed - i.e. the password string or number is ignored. The correct password must be provided, however, when disabling the password protection.

B.30 LastErrorString

This property, which can be set as well as read, is a textual description of the last error that occurred. If none have occurred, it is blank.

Example:

```
txtError = CXServer.LastErrorString
CXServer.LastErrorString = ""
```

Appendix C
OMRON FH Vision Controls

This appendix provides details of the ActiveX properties, events and methods available on each Omron FH Vision control. For basic information about using these controls refer to User Manual chapter 19 Integrating FH Vision systems.

C.1 OMRON FH Image Window

The 'OMRON FH Image Window' control displays the image view from the main FH display. This shows the camera input live stream or last failure, and may overlay each unit measurement or response.

C.1.1 Properties

The following ActiveX properties are available with the 'OMRON FH Image Window' control. These can be accessed via CX-Supervisor's 'ActiveX Property Browser' and can be used in script code.

C.1.1.1 ConnectMode

Description

Set a connection target.

0 or Local: connected to FH/FZ5 simulation software.

1 or Remote: connected to FH/FZ5 Sensor Controller.

C.1.1.2 DispImageTransferSize

Description

Set the resolution of the display image.

When ConnectMode property is Remote, this setting is valid.

When ConnectMode property is Local, this setting is invalid.

The default value is 320.

Update of the image is not done after setting this property.

To reflect the change you need to update the image separately.

C.1.1.3 FzPath

Description

The folder location of where the Vision simulation software is installed (e.g. C:\Program Files (x86)\OMRON\FZ_FH_FJ).

When connecting to Sensor Controller, this setting must be specified.

Note: The CX-Supervisor application is 32 bit, so ensure you specify the folder to the 32 bit version of the simulator. The components will not work with the 64 bit version of the simulator.

C.1.1.4 ImageOrigin

Description

Set the upper left coordinate of a display image to window upper left coordinate.

C.1.1.5 ImageVisible

Description

Set a display presence of the window of image display.

0: window nondisplay

1: window display

C.1.1.6 IpAddress

Description

Set IP address of Sensor Controller of FH/FZ5 of connection target. When connecting to simulation software of FH/FZ5, it's ignored.

C.1.1.7 LineNo

Description

Where a vision controller is using multiple camera inputs to process multiple production lines, set the line number which becomes connection target.

The line number which can be connected will be as follows by operation mode.

* Multi-line random trigger mode: 0-7

* High-speed logging mode: 0

C.1.1.8 Magnification

Description

Set the display magnification of the image display.

The display magnification is designated by real number.

Example

-Set "0.5" when indicating it reducing half.

-Set "2.0" when expanding to double.

-When designating -1, It'll be the automatic magnification added to the window size.

C.1.1.9 SubNo

Description

Set the sub-number of the display unit. When designating "-1", it'll be location list timer mode from the designated processing unit to the processing unit just before the next image input relation, and made the display target.

C.1.1.10 UnitNo

Description

Set processing unit number (program step) of display item.

When designating "-1",the processing unit chosen at present on FH/FZ5 of connection item is displayed.

Note: UnitNo should be set at Design time (using ActiveX Property Browser) or at runtime only before ConnectStart method is called. Setting UnitNo property is not permitted after connection is started.

Tip: To change the Unit number of display after connection is started, execute the "SetDisplayUnitNo" macro.

Example to change to Unit 1 (VBScript):

```
if (PAGE_ImageWindow1.IsConnected()) then
   'Once connected, must set with Macro
   PAGE_ImageWindow1.Macro_DirectExecute("SetDisplayUnitNo ""1""")
else
   'Before connected, set start up property
   PAGE_ImageWindow1.UnitNo = 1
end if
```

C.1.1.11 UpdateImage

Description

Set the timing of a renewal of a display image.

The time of FREEZE: it is renewd each measurement time (freeze display)

NG_IMAGE: it is renewed when it's overall judgment result is NG.

THROUGH: It is always renewed (It's indicated through.)

C.1.1.12 WindowNo

Description

Set the window number of 0-63.

It's necessary to establish the peculiar number in the form.

Note: When the OMRON FH Panel Window and OMRON FH Image Window controls are used in combination a different ‘WindowNo’ should be used.

C.1.2 Methods

The following ActiveX methods are available with the ‘OMRON FH Image Window’ control. These can be accessed via the ‘ActiveX | Execute’ script function.

C.1.2.1 ConnectStart

Description

Connection is tried to FH/FZ5 simulation software ahead of the connection or Sensor Controller with setting factors of a property.

ImageWindow and TextWindow control components indicate measurement factors of FH/FZ5 after connection success.

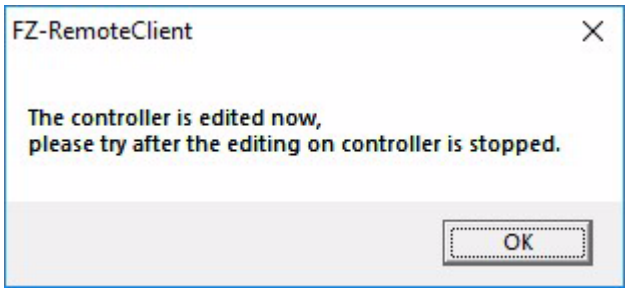
Returns

Connection enforcement:Success = 0

Designated FzPath does not exist:DirectoryNotFoundError = 1

Failed by connection processing:InvalidArgumentError = 3

Note: ConnectStart may fail with the following error. This can occur if the vision controller is running in “Continuous Measurement” mode. Stop the “Continuous Measurement” function on the vision controller in order to allow remote connection.



ConnectionStart may also fail if the target controller is not on the same IP network, even if the controller can be successfully pinged so make sure you are connecting from the same subnet.

C.1.2.2 Macro_DirectExecute

Description

Described macro in argument commandline is carried out on FH/FZ5.

When it connected to the plural lines by the same process, this method cannot use.

Arguments

Commandline: The macro description which carries out.

Note: For full details of the macro commands available on the FH Vision system and their operation refer to the FH/FZ5 User Manual and/or the .NET Control Components Operation Manual, which includes details of the macro command list that can be used in conjunction with the ‘Macro’ methods.

C.1.2.3 Macro_GetVariable

Description

The value of the macro symbol designated in argument variableName is acquired as a character string. It is used when acquiring the value of the macro symbol carried out in Macro_DirectExecute.

Arguments

variableName: Macro variable (name) of data acquisition target (input argument)

data: Acquisition result character string (output argument)

maxLength: The number of elements of the string array for acquisition result acquisition (input argument)

C.1.2.4 Macro_SetVariable

Description

Value is established to the value of the macro symbol designated in argument variableName.

Arguments

variableName: Macro variable (name) of data acquisition target (input argument)

data: setting subject data character string (input argument)

C.1.3 Events

The following ActiveX events are available with the 'OMRON FH Image Window' control. These can be accessed via CX-Supervisor's 'ActiveX Property Browser' and can be used to trigger script execution.

C.1.3.1 ErrorProc

Description

It occurs when the following error occurred on the FH/FZ5. Notified error contents and error code are as follows.

0: System error

1: System error (Fan voltage error)

10: Camera connection error

11: Connected camera has been changed

12: Detection of camera over current

13: Configuration error of light device connection

20: Loading error of image logging dick

30: Time out of parallel output

31: PLC link communication error

32: Detection of parallel I/O camera over current

40: Data load error

41: Data transfer error

42: Incorrect number of start-up Scene group

43: Incorrect number of start-up Scene

C.1.3.2 FzPathChanged

Description

It occurs when the FzPath property was changed.

C.1.3.3 MeasureDisp

Description

It occurs when a measuring result was displayed on the FH/FZ5. This event is triggered each time the controller completes a "Measurement". This is either a manual "Measure" action or "Continuous Measurement" setting.

C.1.3.4 MeasureInit

Description

It occurs when a measurement initializing process was carried out.

When BUSY signal becomes OFF from ON in the center such as the opening and shutting of setting screen and the execution time of command, a measurement initializing process occurs. But, it doesn't occur when measurement command was executed.

C.1.3.5 MeasureOut

Description

It occurs when measuring result was output.

C.1.3.6 OptionEvent

Description

When an optional event occurred on FH/FZ5 of the connection, it occurs.

C.1.3.7 ProcessStarted

Description

It occurs when connection with FH/FZ5 succeeded by the ConnectStart. This event is triggered after the “FZ-CoreRA” process has been successfully started on the host machine.

C.1.3.8 SceneChange

Description

It occurs when scene change processing was carried out on FH/FZ5. This event is triggered when the vision controller scene is changed on the controller. It is also triggered on first connection when the scene is updated.

C.2 OMRON FH Panel Window

The ‘OMRON FH Panel Window’ control provides remote controlling capability for the whole main screen. **NOTE**: the ‘OMRON FH Panel Window’ control and ‘OMRON FH Text Window’ control cannot be used together.

C.2.1 Properties

The following ActiveX properties are available with the ‘OMRON FH Panel Window’ control. These can be accessed via CX-Supervisor’s ‘ActiveX Property Browser’ and can be used in script code

C.2.1.1 FzPath

Description

The folder location of where the Vision simulation software is installed (e.g. C:\Program Files (x86)\OMRON\FZ_FH_FJ).

When connecting to Sensor Controller, this setting must be specified.

Note: The CX-Supervisor application is 32 bit, so ensure you specify the folder to the 32 bit version of the simulator. The components will not work with the 64 bit version of the simulator.

C.2.1.2 IpAddress

Description

Set IP address of Sensor Controller of FH/FZ5 of connection target.

C.2.1.3 LineNo

Description

Where a vision controller is using multiple camera inputs to process multiple production lines, set the line number which becomes connection target.

The line number which can be connected will be as follows by operation mode.

* Multi-line random trigger mode: 0-7

* High-speed logging mode: 0

C.2.2 Methods

The following ActiveX methods are available with the ‘OMRON FH Panel Window’ control. These can be accessed via the ‘ActiveX | Execute’ script function

C.2.2.1 ConnectStart

Description

Connection is tried to FH/FZ5 simulation software ahead of the connection or Sensor Controller with setting factors of a property.

ImageWindow and TextWindow control components indicate measurement factors of FH/FZ5 after connection success.

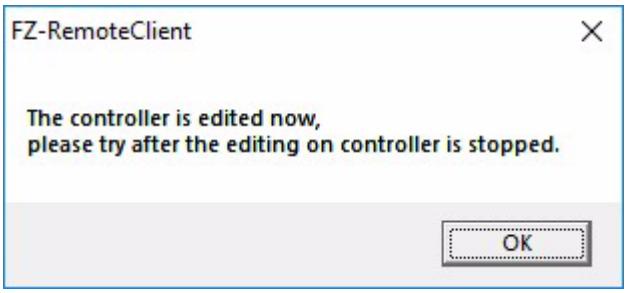
Returns

Connection enforcement:Success = 0

Designated FzPath does not exist:DirectoryNotFoundError = 1

Failed by connection processing:InvalidArgumentError = 3

Note: ConnectStart may fail with the following error. This can occur if the vision controller is running in “Continuous Measurement” mode. Stop the “Continuous Measurement” function on the vision controller in order to allow remote connection.



ConnectionStart may also fail if the target controller is not on the same IP network, even if the controller can be successfully pinged so make sure you are connecting from the same subnet.

C.2.3 Events

There a no events available for the ‘OMRON FH Panel Window’ control.

C.3 OMRON FH Text Window

The ‘OMRON FH Text Window’ control indicates the textual measured result of the designated FH Vision unit. **NOTE**: the ‘OMRON FH Panel Window’ control and ‘OMRON FH Text Window’ control cannot be used together.

C.3.1 Properties

The following ActiveX properties are available with the ‘OMRON FH Text Window’ control. These can be accessed via CX-Supervisor’s ‘ActiveX Property Browser’ and can be used in script code

C.3.1.1 ConnectMode

Description

Set a connection target.

0 or Local: connected to FH/FZ5 simulation software.

1 or Remote: connected to FH/FZ5 Sensor Controller.

C.3.1.2 DispImageTransferSize

Description

Set the resolution of the display image.

When ConnectMode property is Remote, this setting is valid.

When ConnectMode property is Local, this setting is invalid.

The default value is 320.

Update of the image is not done by setting this property.

To reflect the change you need to update the image separately.

C.3.1.3 FontSize

Description

As for these present conditions property, setting is ignored..

C.3.1.4 FzPath

Description

The folder location of where the Vision simulation software is installed (e.g. C:\Program Files (x86)\OMRON\FZ_FH_FJ).

When connecting to Sensor Controller, this setting must be specified.

Note: The CX-Supervisor application is 32 bit, so ensure you specify the folder to the 32 bit version of the simulator. The components will not work with the 64 bit version of the simulator.

C.3.1.5 IpAddress

Description

Set IP address of Sensor Controller of FH/FZ5 of connection target. When connecting to simulation software of FH/FZ5, it's ignored.

C.3.1.6 LineNo

Description

Where a vision controller is using multiple camera inputs to process multiple production lines, set the line number which becomes connection target.

The line number which can be connected will be as follows by operation mode.

* Multi-line random trigger mode: 0-7

* High-speed logging mode: 0

C.3.1.7 UnitNo

Description

Set processing unit number (program step) of display item.

When designating "-1",the processing unit chosen at present on FH/FZ5 of connection item is displayed.

Note: UnitNo should be set at Design time (using ActiveX Property Browser)

or at runtime only before ConnectStart method is called. Setting UnitNo property is not permitted after connection is started.

Tip: To change the Unit number of display after connection is started, execute the "SetDisplayUnitNo" macro.

Example to change to Unit 1 (VBScript):

```
if (PAGE_ImageWindow1.IsConnected()) then
   'Once connected, must set with Macro
   PAGE_ImageWindow1.Macro_DirectExecute("SetDisplayUnitNo ""1""")
else
   'Before connected, set start up property
   PAGE_ImageWindow1.UnitNo = 1
end if
```

C.3.2 Methods

The following ActiveX methods are available with the 'OMRON FH Text Window' control. These can be accessed via the 'ActiveX | Execute' script function

C.3.2.1 ConnectStart

Description

Connection is tried to FH/FZ5 simulation software ahead of the connection or Sensor Controller with setting factors of a property.

ImageWindow and TextWindow control components indicate measurement factors of FH/FZ5 after connection success.

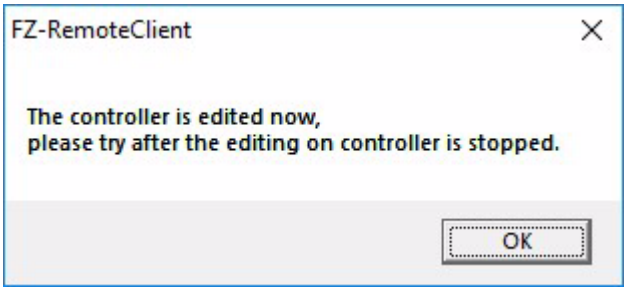
Returns

Connection enforcement:Success = 0

Designated FzPath does not exist:DirectoryNotFoundError = 1

Failed by connection processing:InvalidArgumentError = 3

Note: ConnectStart may fail with the following error. This can occur if the vision controller is running in "Continuous Measurement" mode. Stop the "Continuous Measurement" function on the vision controller in order to allow remote connection.



ConnectionStart may also fail if the target controller is not on the same IP network, even if the controller can be successfully pinged so make sure you are connecting from the same subnet.

C.3.2.2 Macro_DirectExecute

Description

Described macro in argument commandline is carried out on FH/FZ5.

When it connected to the plural lines by the same process, this method cannot use.

Arguments

Commandline: The macro description which carries out.

Note: For full details of the macro commands available on the FH Vision system and their operation refer to the FH/FZ5 User Manual and/or the .NET Control Components Operation Manual, which includes details of the macro command list that can be used in conjunction with the 'Macro' methods.

C.3.2.3 Macro_GetVariable

Description

The value of the macro symbol designated in argument variableName is acquired as a character string. It is used when acquiring the value of the macro symbol carried out in Macro_DirectExecute.

Arguments

variableName: Macro variable (name) of data acquisition target (input argument)

data: Acquisition result character string (output argument)

maxLength: The number of elements of the string array for acquisition result acquisition (input argument)

C.3.2.4 Macro_SetVariable

Description

Value is established to the value of the macro symbol designated in argument variableName.

Arguments

variableName: Macro variable (name) of data acquisition target (input argument)

data: setting subject data character string (input argument)

C.3.3 Events

The following ActiveX events are available with the 'OMRON FH Text Window' control. These can be accessed via CX-Supervisor's 'ActiveX Property Browser' and can be used to trigger script execution.

C.3.3.1 ErrorProc

Description

It occurs when the following error occurred on the FH/FZ5. Notified error contents and error code are as follows.

0: System error

1: System error (Fan voltage error)

10: Camera connection error

11: Connected camera has been changed

12: Detection of camera over current

13: Configuration error of light device connection

20: Loading error of image logging dick

30: Time out of parallel output

31: PLC link communication error

32: Detection of parallel I/O camera over current

40: Data load error

41: Data transfer error

42: Incorrect number of start-up Scene group

43: Incorrect number of start-up Scene

C.3.3.2 FzPathChanged

Description

It occurs when the FzPath property was changed.

C.3.3.3 MeasureDisp

Description

It occurs when a measuring result was displayed on the FH/FZ5. This event is triggered each time the controller completes a “Measurement”. This is either a manual “Measure” action or “Continuous Measurement” setting.

C.3.3.4 MeasureInit

Description

It occurs when a measurement initializing process was carried out.

When BUSY signal becomes OFF from ON in the center such as the opening and shutting of setting screen and the execution time of command, a measurement initializing process occurs. But, it doesn't occur when measurement command was executed.

C.3.3.5 MeasureOut

Description

It occurs when measuring result was output.

C.3.3.6 OptionEvent

Description

When an optional event occurred on FH/FZ5 of the connection, it occurs.

C.3.3.7 ProcessStarted

Description

It occurs when connection with FH/FZ5 succeeded by the ConnectStart. This event is triggered after the “FZ-CoreRA” process has been successfully started on the host machine.

C.3.3.8 SceneChange

Description

It occurs when scene change processing was carried out on FH/FZ5.

Appendix D
Obsolete Script Functions

This appendix provides a summary of script functions that are obsolete and have been removed from the standard documentation. Details are included here to assist maintaining old projects still using these features. These features should not be used in development of new solutions as it is likely support for the following features may and will be removed from the next or future releases. Please also refer to User Manual Appendix G for full details of all obsolete features.

D.1 Sleep

Description

Pause execution of a script for specified duration.

Syntax

```
Sleep (duration)
```

Remarks

Argument	Type	Description
Duration	- - -	Number of milliseconds to wait before continuing.

Typical Example

```
Sleep (1000)
```

CX-Supervisor waits 1 second.

Note: The sleep statement should be used with caution, as some other parts of the system may not be updated while a script is sleeping. It also uses multithreading which means some tasks like PLC communication may occur in parallel and behave unpredictably.

Note: In a well designed, truly event driven system use of the Sleep() statement should never be required. Always consider if the statements after the Sleep should be in their own script, executed when a Condition occurs.

Note: The Granularity (or intervals) differs between Operating Systems. In Windows NT (and 2000) expiration is checked every 10ms, so 'Sleep(100)' actually pauses for any time between 100.00 to 109.99 milliseconds depending on when it was started. For Windows 98 (and ME) the granularity is 55ms so 'Sleep(100)' actually pauses for 110 (2 times 55) to 164.99 milliseconds (nearly 3 times 55). For this reason, Sleep statements can act differently on different Operating Systems making the application OS dependent.

Note: Sleep should never be used as a delay for timing processes, for the following reasons:

- The actual time delay depends on the OS as described above
- There is always an error of 0 to 1 granularity, depending on when the action is started.
- The frequency can not be guaranteed as the OS may be busy, or handling other processes.

D.2 DDE Commands

DDE as a means for exchanging data has now been obsolete for some years. In fact for so long even its successor, OLE Automation is obsolete. DDE has also proved to be a poor technology, suffering from unfixed memory leaks both

in the native Operating Systems, and tools like Microsoft Excel. This technology has now been replaced and the CX-Supervisor Communications Control should be used instead.

The following DDE script commands are obsolete.

D.2.1 DDEExecute

Syntax

```
returnstate = DDEExecute(channel, {command})
```

Remarks

Argument	Type	Description
returnstate	Bool	Returnstate is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel based on the DDEInitiate() command must be open or an error is reported.
command	String	This is a command as recognised by the server application specified within the channel.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
DDEExecute(channelname,
{[OPEN("C:\EXCEL\WORK\SHEET2.XLS")]})
```

The file 'SHEET2.XLS' within path 'C:\EXCEL\WORK' is opened in Microsoft Excel, as specified by the Integer point 'channelname'. The file 'SHEET1.XLS' is already open in Microsoft Excel

D.2.2 DDEInitiate

Syntax

```
channel = DDEInitiate("server", topic")
```

Remarks

Argument	Type	Description
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command.
server	String	This contains the application that supports DDE as a DDE server. Typically, this is the name of the applications' *.EXE executable file without the filename extension. At runtime, the server application must be open or a value cannot be returned and an error is reported.

Argument	Type	Description
topic	String	This contains the name of the topic recognised by the server application. Typically, a topic is a document within an application. At runtime, the topic must be open or a value cannot be returned and an error is reported. The topic may be left empty, which enables documents to open remotely prior to making a specified connection. The topic name 'System' may be used to find out which other topics within the server application are available. However, this is dependent on the server application supporting this topic.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
```

The Integer point 'channelname' is provided with a DDE link to the application Microsoft Excel which is run by the executable filename 'EXCEL.EXE', and to the file 'SHEET1.XLS' within that application.

D.2.3 DDEOpenLinks

Syntax

```
returnstate = DDEOpenLinks(channel)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel in the DDEInitiate() command must be open or an error is reported.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
DDEOpenLinks(channelname)
```

The DDEOpenLinks command enables points which have been configured to communicate via DDE to begin data transfer. Data transfer between CX-Supervisor and the application Microsoft Excel is automatically maintained until the channel is closed either by Microsoft Excel or by the command DDETerminate() using the Integer point 'channelname', or the command DDETerminateAll().

D.2.4 DDEPoke

Syntax

```
returnstate = DDEPoke(channel, "item", pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Argument	Type	Description
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the in the DDEInitiate() command must be open or an error is reported.
item	string	This is an item as recognised by the server application. For instance, a cell is an item within a spreadsheet application. Likewise, a page is an item for a word processing application. It is wholly dependent on the server application.
pointname	point	This is a point whose attributes must include a DDE Access of 'Read/Only' or 'Read/Write'. The contents of this point are assigned to the server application.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
DDEPoke(channelname, "R2C5", data)
```

The content of point 'data' is sent to row 2, column 5 of 'SHEET1.XLS' in the Microsoft application. The Microsoft Excel application, and 'SHEET1.XLS' are specified by Integer point 'channelname'.

D.2.5 DDERequest

Syntax

```
pointname = DDERequest(channel, "item")
```

Remarks

Argument	Type	Description
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel in the DDEInitiate() command must be open or an error is reported.
item	string	This is an item as recognised by the server application. For instance, a cell is an item within a spreadsheet application. Likewise, a page is an item for a word processing application. It is wholly dependent on the server application.
pointname	point	This is a point whose attributes must include a DDE Access of 'Read/Write'.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
cellref = DDERequest("channelname", "R2C5")
```

The point 'cellref' is filled from a specific item, row 2, column 5 from 'SHEET1.XLS' from the Microsoft Excel application, specified by the Integer point 'channelname'.

D.2.6 DDETerminate

Syntax

```
returnstate = DDETerminate(channel)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel in the DDEInitiate() command must be open or an error is reported.

Typical Example

```
DDETerminate(channelname)
```

The server and topic specified by Integer point 'channelname' is closed.

D.2.7 DDETerminateAll

Syntax

```
returnstate = DDETerminateAll()
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DDETerminateAll()
```

All previously initiated DDE links are closed.

D.2.8 EnableDDE

Syntax

```
returnstate = EnableDDE(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Pointname	bool point	A Boolean point that holds the required enable/ disable state

Typical Examples

```
EnableDDE(result)
```

DDE functions are enabled based on the value of point 'result'. If 'point' is 'TRUE', then DDE is enabled, if 'point' is 'FALSE', then DDE is disabled.

```
EnableDDE(TRUE)
```

DDE functions can also be enabled directly without using a point to hold the desired status.

D.3 Graph Commands

D.3.1 ClearGraph

Syntax

```
returnstate = ClearGraph("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be cleared.
pagename	string	Optional parameter indicating the name of the page that the graph is on.

Typical Examples

```
ClearGraph("Graph_1", "TestPage1")
```

The trend or scatter graph on 'TestPage1' with the identifier 'Graph_1' has its data cleared.

```
ClearGraph ("Graph_2")
```

The trend or scatter graph on the current page, with the identifier 'Graph_2', has its data cleared.

D.3.2 StartGraph

Syntax

returnstate = StartGraph("graphid", "pagename")

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be started.
pagename	string	Optional parameter indicating the name of the page that the graph is on.

Typical Examples

```
StartGraph("Graph_1", "TestPage1")
```

The trend or scatter graph on 'TestPage1' with the identifier 'Graph_1' has its data logging started.

```
StartGraph("Graph_2")
```

The trend or scatter graph on the current page with the identifier 'Graph_2' has its data logging started.

Note: This command is provided for compatibility with SCS v2.0 applications. For newer applications the data logging facilities should be used in preference.

D.3.3 StopGraph

Syntax

```
returnstate = StopGraph("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be stopped.
pagename	string	Optional parameter indicating the name of the page that the graph is on.

Typical Examples

```
StopGraph("Graph_1", "TestPage1")
```

The trend or scatter graph on 'TestPage1' with the identifier 'Graph_1' has its data logging stopped.

```
StopGraph("Graph_2")
```

The trend or scatter graph on the current page with the identifier 'Graph_2' has its data logging stopped.

D.3.4 EditGraph

Syntax

```
returnstate = EditGraph("graphid")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be edited.

Typical Example

```
EditGraph("Graph_1")
```

The Edit Graph dialog is displayed offering options to view historical data for the chosen trend graph.

- Display Data loads the currently selected data sample i.e. either the current screen data or a snapshot of the data, into the trend graph.
- Snapshot stores the current data buffer associated with the trend graph. The snapshot is given a time stamped default description.
- Description provides the ability to change the description associated with the snapshot.
- Import Data provides the ability to load in a previously saved trend graph file.
- Export Data provides the ability to store a snapshot to a file, either in internal CX-Supervisor format, or as a text file that can be imported into other applications.
- Delete removes the currently selected snapshot.

Note: This command is provided for compatibility with SCS v2.0 applications. For newer applications the data logging facilities should be used in preference.

Note: This command can only be used if the trend is set to log to a file.

D.3.5 SaveGraph

Syntax

```
returnstate = SaveGraph("graphid")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be saved.
pagename	string	Optional parameter indicating the name of the page that the graph is on.

Typical Examples

```
SaveGraph("Graph_1", "TestPage1")
```

The trend graph on the page 'TestPage' with the identifier 'Graph_1' has its data saved to disc.

```
SaveGraph("Graph_2")
```

The trend graph on the current page with the identifier 'Graph_2' has its data saved to disc.

D.3.6 Snapshot

Syntax

```
returnstate = Snapshot("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	Returnstate is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to have the snapshot.
pagename	string	Optional parameter indicating the name of the page that the graph is on.

Typical Examples

```
Snapshot("Graph_1", "TestPage1")
```

The current data in trend graph 'Graph1' on 'TestPage1', is stored and is able to be viewed via the EditGraph command.

```
Snapshot("Graph_2")
```

The current data in trend graph 'Graph1' on the current page, is stored and is able to be viewed via the EditGraph command.

Note: This command is provided for compatibility with SCS v2.0 applications. For newer applications the data logging facilities should be used in preference.

D.4 Printer Commands

D.4.1 GetSpoolCount

Syntax

```
returnstate = GetSpoolCount()
```

Remarks

Argument	Type	Description
returnstate	int	Number of messages queued up waiting to be printed on Alarm/Message printer.

Typical Example

```
NumberMessages = GetSpoolCount()
```

The count of the number of messages (typically printed alarms) that are queued up waiting to be sent to the CX-Supervisor Alarm/Message printer is returned.

D.4.2 SetPrinterConfig

Syntax

```
returnstate StePrintConfig(Driver, Device, Port)
```

Remarks

Argument	Type	Description
returnstate	Bool	Returnstate is '1' if the function is successful, or '0' otherwise.
Driver	String	Name of printer device (e.g. "Epson9" for 9 pin Epson printers.
Device	String	Name of specific device (e.g. "Epson FX-870"). This is optional.
Port	String	Name of port or file(e.g. "LPT1.").
Line Terminator	String	Optional. Sets terminator (e.g. cr) to be added to end of each printed line.

Typical Examples

```
SetPrinterConfig("SCSPRN", "", "LPT1:")
```

This uses standard CX-Supervisor line print driver.

```
SetPrinterConfig("", "", "")
```

This uses default Windows printer driver.

```
SetPrinterConfig("Epson9", "", "LPT2:")
```

This uses Epson printer driver, attached to LPT2.

```
SetPrinterConfig(DriverNamePoint,     DeviceNamePoint,
PrintNamePoint)
```

This uses text points.

```
Terminator = FormatText("%c%c",13,10)
```

Character 10 is 'lf' (newline), character 13 is cr (carriage return).

```
SetPrinterConfig("Epson9","","LPT1:",Terminator)
```

D.5 JScript

The JScript script commands have been deprecated in CX-Supervisor 3.5. However, JScript code will continue to function in the same way it did in earlier versions of CX-Supervisor.

The following JScript script commands are obsolete.

D.5.1 ExecuteJScript

Description

Creates aliases allowing Java Script to be executed in line. See Appendix C for a list of supported functions and details of the Windows Scripting Host.

Syntax

```
@JSCRIPT
@ENDSCRIPT
```

Typical Examples

```
@JSCRIPT
    Point("PointName") = OLE_1.Height;
@ENDSCRIPT
```

This Java Script will write the value of the property 'Height' from the OLE object 'OLE1' into the Point named 'PointName'.

Note: The Java Script can not include the { or } characters. To use these, put the script in a text file and use the ExecuteJScriptFile function.

D.5.2 ExecuteJScriptFile

Description

Allows Java script stored in a text file to be executed. This uses the windows scripting host which must be installed. See Appendix C for a list of supported functions.

Syntax

```
returnstate = ExecuteJScriptFile(scriptfile)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
scriptfile	Text	The name of the file with the Java Script to execute.

Typical Examples

```
returnstate = ExecuteJScriptFile("c:\jscript.txt")
```

This will execute the Java Script stored in "c:\jscript.txt".

D.6 Atan

CX-Supervisor Script function ‘Atan’ is obsolete and has been replaced by VBScript equivalent ‘Atn’ function.

D.7 Sqrt

CX-Supervisor Script function ‘Sqrt’ is obsolete and has been replaced by VBScript equivalent ‘Sqr’ function.

D.8 GetPointValue / SetPointValue

The GetPointValue and SetPointValue functions, to get or set the element of an array, have been replaced by the use of square brackets which allows access within both scripts and now expressions. See section “4-6 Point Arrays within Script Commands and Expressions” for details on how to use.

Syntax

```
returnpoint = GetPointValue(pointname,offset)
```

Remarks

Argument	Type	Description
pointname	point	This is the name of the point whose contents are to be returned.
offset	integer	This specifies the offset into an array point. 0 if the point is not an array point.
returnpoint	point	Point that contains the return value. The type of data returned is dependent on the pointname specified.

Typical Example

```
pointname = 10;
returnpoint = GetPointValue(pointname,0)
```

The point 'returnpoint' contains the value 10. The offset is added to any offset specified for pointname. For example:

```
returnpoint = GetPointValue(a[10],10)
```

Causes the 21st element (offsets begin at zero) of array 'a' to be retrieved.

Note: It is often simpler to access an array element directly, e.g. returnpoint = a[20].

D.9 Colour Palette (ARGB Values Used in Script Code)

The new colour palette, described in section 1 of the User Manual, replaces the old colour palete, which is shown in the table below. When an older project is loaded into CX-Supervisor 3.4, or later, any references to a colour palatte index will be converted to the appropriate ARGB value.

No.	Colour	No.	Colour
0	black	12	purple
1	blue	13	olive
2	green	14	dark_grey
3	cyan	15	light-grey
4	red	16	pale-green
5	magenta	17	light-blue
6	yellow	18	off-white
7	white	19	grey
8	dark_blue	20	cherry
9	dark_green	21	silver
10	blue-green	22	apple
11	brown	23	orange
		24-65	Not used

Appendix E
CX-Supervisor Script Language

This chapter describes the CX-Supervisor script language syntax. It provides a detailed definition of the syntax of CX-Supervisor scripts that drive project, page and object actions, and CX-Supervisor expressions as used by objects and scripts. In conjunction with the script functions and methods described in Chapter 6, the CX-Supervisor script language provides a very powerful, compiled, fast and full featured programming language.

The following table describes the script language syntax at a glance.

Function Name	Function Type	Type	Remarks
&, |, ^, <<, >>	bitwise operators	All	Applies bitwise expressions
(objects)	statement	OP	Specifies an object name for modification or test.
(points)	statement	All	Specifies a point name for modification or test.
+, -, *, /, %, =, ++, --	arithmetic operators	All	Applies arithmetic expressions.
<, >, <=, >=, ==, !=	relational operators	All	Applies relational expressions.
AND	logical operators	All	Applies logical expressions.
CALL	statement	All	Call a subroutine
DO LOOP WHILE UNTIL EXIT DO	statement	Scr	Script segment to be repeated
FALSE	Boolean state	Scr	Applies Boolean expression.
FOR TO STEP NEXT EXIT FOR	statement	Scr	Script segment to be repeated
IFTHEN ELSE\ELSEIF ENDIF	statement	Scr	Applies a test to a script.
OR	logical operators	All	Applies logical expressions.
NOT	logical operators	All	Applies logical expressions.
REM	statement	Scr	Remarks on line or lines of script.
RETURN	statement	Scr	Stops sequential execution of script.
SELECT CASE/ END SELECT	statement	Scr	Applied to complex tests.
TRUE	Boolean state	Scr	Applies Boolean expression.

The 'Type' column refers to the types of script and expression the function can be applied to. 'All' refers to both expressions and scripts. 'Scr' refers to scripts only. 'OP' refers to Object and Page scripts only.

E.1 Points

E.1.1 Basic Point Assignment

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value.
expression	The value to be assigned to pointname. The expression may be of type Boolean, Integer, Real or Text.

Typical Examples

```
count = 100
```

The Integer or Real point 'count' is assigned the value 100.

```
result = TRUE
```

The Boolean point 'result' is assigned the state ''TRUE".

```
name = "Valve position"
```

The Text point 'name' is assigned the associated text, contained within quotation marks.

Note: When assigning Real (floating point) values to an Integer point the assignment uses the 'Symetrical Rounding Down' (towards 0) standard. This means a value of 4.1 would be assign a value 4. A value of -4.1 would asign a value of -4.

References

Refer to chapter 4, Punctuation for details of the use of quotation marks.

E.1.2 Further Point Assignment

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value.
expression	The value to be assigned to pointname. The expression may be of type Boolean, Integer or Real and can include other points, logical or arithmetical expressions. Mathematical precedence is applied as follows: • Parenthesis (highest). • Unary minus and NOT logical operator. • Multiplication, division and modulus. • Addition and subtraction. • Greater than, less than, greater than or equal to, and less than or equal to relational operators. • Shift Left (SHL) and Shift Right (SHR). • Equal to and not equal to relational operators. • Bitwise AND, XOR, OR. • AND logical operator, OR logical operator (lowest).

Typical Examples

```
lift = height + rate/5.0
```

The Integer or Real point 'lift' is assigned the value calculated by the value of point 'rate' divided by 5, plus the value of point 'height'. Precedence can be changed by the introduction of parenthesis.

```
lift = lift - 0.2
```

The Integer or Real point 'lift' is assigned the value calculated by the current value of point 'lift' minus 0.2.

```
distance = distance * time
```

The Integer or Real point 'distance' is assigned the value calculated by the current value of point 'distance' multiplied by point 'time'.

References

Refer to chapter 4, Logic and Arithmetic for details of the use of arithmetic and logic functions. Refer to chapter 4, Punctuation for details of the use of parenthesis.

E.2 Logic and Arithmetic

E.2.1 Arithmetic Operators

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value based on an arithmetical expression.
expression	The value to be assigned to pointname. The expression may include the following operators with points and constants: • Addition '+'. • Subtraction '-'. • Multiplication '*'. • Division '/'. • Modulus '%'. • Increment '++'. • Decrement '--'.

Typical Examples

```
result = 60 + 20/5
```

The Integer or Real point 'result' is assigned the value calculated by the value of 20 divided by 5, plus 60.

```
lift = height + rate/5.0
```

The Integer or Real point 'lift' is assigned the value calculated by the value of point 'rate' divided by 5, plus the value of point 'height'. Precedence can be changed by the introduction of parenthesis.

References

Refer to chapter 4, Punctuation for details of the use of parenthesis.

E.2.2 Bitwise Operators

Syntax

```
pointname = expression
```

or

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
pointname	The pointname to be assigned a value based on the bitwise operation.
expression	The value to be assigned to pointname, or to be evaluated as a Boolean expression. The expression can include the following operators with points and constants: • Bitwise AND, 'BITAND' or '&'. • Bitwise OR, 'BITOR' or '|'. • Bitwise XOR, 'XOR' or '^'. • Bitwise Shift Left, 'SHL' or '<<'. • Bitwise Shift Right, 'SHR' or '>>'.

Typical Examples

```
MSB = value & 128
```

The Boolean point 'MSB' is set 'TRUE' if the binary representation of 'value' has the bit set which is worth 128.

```
Pattern = value << 2
```

The binary representation of 'value' is shifted left twice, and stored in 'pattern'. Each Shift Left operation has the effect of doubling the value, so two shifts quadruple the value.

E.2.3 Logical Operators

Syntax

```
pointname = expression
```

or

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
Pointname	The point name to be assigned a value based on a logical expression.

Argument	Description
Expression	The Boolean value to be assigned to pointname or the Boolean value forming a conditional statement. The expression includes the following operators with points and constants: • And 'AND'. • Or 'OR'. • Not 'NOT'.

Typical Examples

```
flag = temp AND speed
```

The Boolean point 'flag' is assigned a value based on the logic of point 'temp' AND point 'speed'. If 'temp' and 'speed' are both not zero, 'flag' is set to 1, or ''TRUE''. A value of zero in either 'temp' or 'speed' supplies 'FALSE' or 0 to 'flag'.

```
IF flag AND temp AND speed THEN
    flag = FALSE
ENDIF
```

The Boolean point 'flag' is assigned 'FALSE', on the condition that 'flag' AND point 'temp' AND point 'speed' are all not zero. If the condition fails, then 'flag' is not assigned 'FALSE'.

References

Refer to chapter 4, Control Statements for details of the use of the IF THEN ELSE/ELSEIF ENDIF statements.

E.2.4 Relational Operators

Syntax

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
Expression	The value forming a conditional statement. The expression may include the following operators with points and constants: • Greater than '>'. • Less than '<'. • Greater than or equal to '>='. • Less than or equal to '<='. • Not equal to '!='. • Equal to '=='.

Typical Example

```
IF fuel < 0 THEN
    fuel = 0
ENDIF
```

The point 'fuel' is assigned the value 0 on the condition that currently, 'fuel' is less than 0. If 'fuel' is not less than 0, then it is not assigned the new value.

References

Refer to chapter 4, Control Statements for details of the use of the IF THEN ELSE/ELSEIF ENDIF statements.

E.3 Control Statements

E.3.1 Simple Conditional Statements

Syntax

```
IF condition THEN
    statementblock1
ENDIF
```

or

```
IF condition THEN
    statementblock1
ELSE
    statementblock2
ENDIF
```

Remarks

Argument	**Description**
Condition	The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string.
Statementblock1	One or more statements which are performed if the condition is met.
Statementblock2	One or more statements which are performed if the condition is not met.

Typical Examples

```
IF fuel < 0 THEN
    fuel = 0
ENDIF
```

Provided Integer point 'fuel' is less than 0, then it is assigned the value 0.

```
IF burner THEN
    fuel = fuel - rate
ENDIF
```

Provided Boolean point 'burner' is ''TRUE'', then Integer point 'fuel' is assigned a new value. It is also possible to apply 'IF burner == TRUE THEN' as the first line, with identical results.

```
IF distance > 630 AND distance < 660 AND lift >= -3
THEN
    winner = TRUE
    burner = FALSE
ENDIF
```

Provided that Integer point 'distance' is greater in value than 630 AND 'distance' is less in value than 660 (i.e. 'distance' is a value between 630 and 660) AND point 'lift' is greater than or equal to -3, then Boolean points 'winner' and 'burner' are assigned new values.

```
IF burner AND fuel > 0 AND rate > 0 THEN
    fuel = fuel - rate
ELSE
    lift = 0
    altitude = 0
ENDIF
```

Provided that Boolean point 'burner' is ''TRUE'' AND points 'fuel' and 'rate' are greater in value than 0, then 'fuel' is assigned a new value. Otherwise points 'lift' and 'altitude' are assigned a new value.

References

Refer to chapter 4, Punctuation, Indentation for details on the layout of code.

E.3.2 Nested Conditional Statements

Syntax

```
IF conditionA THEN
    statementblock1
    IF conditionB THEN
        statementblock3
    ENDIF
ELSE
    statementblock2
ENDIF
```

or

```
IF conditionA THEN
    statementblock1
    IF conditionB THEN
        statementblock3
    ELSE
        statementblock4
    ENDIF
ELSE
    statementblock2
ENDIF
```

or

```
IF conditionA THEN
    statementblock1
ELSEIF conditionB THEN
    statementblock3
ENDIF
```

or

```
IF conditionA THEN
    statementblock1
ELSE
    statementblock2
    IF conditionB THEN
        statementblock3
    ELSE
        statementblock4
    ENDIF
ENDIF
```

Remarks

Argument	Description
conditionA	The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string.
conditionB	This condition is nested in the first condition, either on a successful or unsuccessful evaluation of conditionA. The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string. There is no limit to the number of nested conditional statements.
statementblock1	One or more statements which are performed if conditionA is met.
statementblock2	One or more statements which are performed if conditionA is not met.
statementblock3	One or more statements which are performed if conditionB is met.
statementblock4	One or more statements which are performed if conditionB is not met.

Typical Examples

```
IF burner AND fuel > 0 AND rate > 0 THEN
    lift = lift + rate/5
ELSE
    count = 1
    IF altitude > 140 THEN
        lift = lift - 0.2
    ENDIF
ENDIF
```

Provided a successful evaluation has been made to points 'burner' AND 'fuel' AND 'rate', point 'lift' is updated with the current value of rate divided by 5 plus 'lift'. Otherwise, a further evaluation is required on point 'altitude'. If 'altitude' is currently greater than 140, then 'lift' is decremented by 0.2.

```
IF burner AND fuel > 0 AND rate > 0 THEN
    lift = lift + rate/5
ELSE
    IF altitude > 140 THEN
        lift = lift - 0.2
    ENDIF
ENDIF
IF burner AND fuel > 0 AND rate > 0 THEN
    lift = lift + rate/5
ELSEIF altitude > 140 THEN
    lift = lift - 0.2
ENDIF
```

These two examples are identical. The use of the ELSEIF statement combines the ELSE statement and the IF/ENDIF statements for brevity. It is acceptable to have more than one ELSEIF statement in an IF THEN ELSE/ ELSEIF ENDIF construct.

References

Refer to chapter 4, Punctuation for details of the use of indentation.

E.3.3 Case Select

Syntax

```
SELECT CASE expression
    CASE expression
        statementblock1
    CASE expression
        statementblock2
    CASE expression
        statementblock3
END SELECT
```

or

```
SELECT CASE expression
    CASE expression
        statementblock1
    CASE expression
        statementblock2
    CASE ELSE
        statementblock3
END SELECT
```

Remarks

Argument	**Description**
expression	The expression may be a point, or a calculation of constants and/or points that produces a result.
statementblock1	One or more statements that are only performed if the preceding CASE expression is met.
statementblock2	One or more statements that are only performed if the preceding CASE expression is met.
statementblock3	One or more statements that are only performed if the preceding CASE expression is met.

Typical Examples

```
SELECT CASE colourvalue
    CASE 1
        colour (blue)
    CASE 2
        colour (green)
    CASE 3
        colour (cyan)
    CASE ELSE
        colour (0)
END SELECT
```

This example shows the assignment of a colour according to the value of a point. The value of Integer point 'colourvalue' is evaluated and compared with each case until a match is found. When a match is found, the sequence of actions associated with the CASE statement is performed. When 'colourvalue' is 1, the colour given to the current object is blue, when 'colourvalue' is 2, the colour given to the current object is green, when 'colourvalue' is 3, the colour given to the current object is cyan. If 'colourvalue' falls outside the integer range 1-3, then the colour given is 0 (black). Like ELSE and ELSEIF, the CASE ELSE statement is optional.

```
SELECT CASE TRUE
    CASE temperature > 0 AND temperature <= 10
        colour (blue)
    CASE temperature > 10 AND temperature <= 20
        colour (green)
    CASE temperature > 20 AND temperature <= 30
        colour (red)
    CASE ELSE
        colour (white)
ENDSELECT
```

In this example, instead of using a point as the condition as with the previous example, the value is the condition - in this case Boolean state "TRUE" - with the integer point 'temperature' being tested at each case. If it is "TRUE" that 'temperature' is between 0 and 10, then the current object is set to blue, or if it is "TRUE" that 'temperature' is between 11 and 20, then the current object is set to green, or if it is "TRUE" that 'temperature' is between 21 and 30, then the current object is set to red. If none of these CASE statements are met, then the current object is set to white. Like ELSE and ELSEIF, the CASE ELSE statement is optional.

References

Refer to chapter 6, Object Commands for details of applying attributes to an object and for the use of the Colour object command. Refer to chapter 8, Colour Palette for details of the Colour Palette colour designation.

E.3.4 FOR... NEXT Loop

Syntax

```
FOR pointname = startpt TO endpt STEP steppt
    statementblock1
NEXT
```

Remarks

Argument	Description
pointname	The pointname to be used as the loop counter.
startpt	The initial setting of pointname, and the first value to be used through the loop.
endpt	The last value to be used. The loop ends when pointname exceeds this value.
steppt	Amount to increase pointname by every pass of the loop. Steppt can be negative to count backwards providing startpt is larger than endpt. The STEP keyword and variable may be omitted in which case pointname is incremented at each pass of the loop (identical to adding STEP 1).

Typical Examples

```
FOR loopcount = 0 TO 100
    Ellipse_1.vertical%fill = loopcount
NEXT
```

In this example, 'Ellipse_1' is gradually filled 100 times.

```
FOR loopcount = 100 TO 0 STEP -5
    Ellipse_1.vertical%fill = loopcount
NEXT
```

In this example, the fill for 'Ellipse_1' is gradually removed 20 times (100 times/-5).

Note: Loop statements should be used with caution, as they consume processor time while they are running and some other parts of the system may not be updated.

E.3.5 DO WHILE/UNTIL Loop

Syntax

```
DO WHILE expression
    statementblock
LOOP
```

or

```
DO
    statementblock
LOOP WHILE expression
```

or

```
DO UNTIL expression
    statementblock
LOOP
```

or

```
DO
    statementblock
LOOP UNTIL expression
```

Remarks

Argument	Description
expression	The expression may be a point, or a calculation of constants and/or points that produces a result.
statementblock	One or more statements to be executed multiple times depending on expression.

Typical Example

```
DO WHILE dooropen == TRUE
    Message ("You must shut the door before
    continuing")
LOOP
DO
    nextchar = Mid (Mystring, position, 1)
    position = position + 1
LOOP UNTIL nextchar = "A"
```

Note: Loop statements should be used with caution, as they consume processor time while they are running and some other parts of the system may not be updated.

E.4 Subroutines

E.4.1 Call

Syntax

```
CALL subroutine (arguments)
```

Remarks

Argument	Description
subroutine	The name of the subroutine defined at project level.
arguments	The list of arguments required by the subroutine separated by commas. Each argument may be a pointname, constant, arithmetical or logical expression or any valid combination.

Typical Example

```
CALL MySub ($Second, "Default", 2 + Int1)
```

E.4.2 Return

Syntax

```
RETURN
```

```
Typical Example
IF limit > 1000 THEN
    RETURN
ELSE
    value = limit
ENDIF
REM final part of script
POLYGON_1.COLOUR = red
ELLIPSE_5.WIDTH = value
```

The integer point 'limit' is tested for its value. If its value exceeds 1000, then the condition is met, and the RETURN command is executed. All statements after the RETURN command are ignored. If the value of integer point 'limit' does not exceed 1000, then the RETURN command is not executed, and statements after the RETURN command are performed.

References

Refer to the CX-Supervisor User Manual for the use of the RETURN statement for Recipe validation.

E.5 Punctuation

E.5.1 Command String Delimiters

Description

Alternative string delimiters allowing string to contain quote " characters.

Syntax

```
{Some "string" text}
```

Typical Example

```
Message({Error: "Invalid Function" occurred})
```

The '{' and '}' braces inserted around the whole strings allows the actual text in the string to contain quotes which will be displayed normally. They can be used in any situation where quotes can be used whether or not embedded quotes are required. However, for clarity the quote characters should be used by preference.

E.5.2 Indentation

Typical Examples

```
IF burner AND fuel > 0 AND rate > 0 THEN
lift = lift + rate/5
```

```
ELSE
IF altitude > 140 THEN
lift = lift - 0.2
ENDIF
ENDIF
IF burner AND fuel > 0 AND rate > 0 THEN
    lift = lift + rate/5
ELSE
    IF altitude > 140 THEN
        lift = lift - 0.2
    ENDIF
ENDIF
```

Both examples provide identical functionality, but the use of indentation, either spaces or tabs to show the construction of the statements aids readability.

The use of the ELSEIF statement in this example was omitted for clarity.

E.5.3 Multiple Commands

Typical Examples

```
count = 75
result = log(count)
count = 75 : result = log(count)
```

Both examples provide identical functionality, but the use of the colon between statements allows both to reside on the same line.

E.5.4 Parenthesis

Typical Examples

```
result = 20 + 30 * 40
```

The result is 1220.

```
result = (20 + 30) * 40
```

The values in parenthesis are calculated first. The result is 2000.

References

Refer to chapter 4, Logic and Arithmetic, Arithmetic Operations for further details.

E.5.5 Quotation Marks

Typical Examples

```
name = "Valve position"
```

The Text point 'name' is assigned associated text, contained within quotation marks. Quotation marks must be used in this instance.

```
Message("This text to be displayed as a message.")
```

Passing static text as arguments to functions.

```
BlueCarsAck = IsAlarmAcknowledged("BLUEPAINT")
```

The point 'BlueCarsAck' is assigned a Boolean state based on the alarm 'BLUEPAINT'. Quotation marks must be used for an alarm name.

E.5.6 Remarks

Syntax

```
REM | rem comment
```

or

```
'comment
```

Remarks

Argument	Type	Description
Comment	---	Descriptive text.

Typical Examples

```
REM The following statement adds two numbers
result = 45 + 754
result = 45 + 754  'add two numbers
```

E.6 Indirection within Script Commands and Expressions

It is possible to use text points directly or indirectly in place of literal string arguments within scripts and expressions. For instance, each of the following commands has the same effect:

- Using a string literal;

```
PlayOLE("ole_1", 0)
```

- Using a textpoint directly;

```
textpoint = "ole_1"
PlayOLE(textpoint, 0)
```

- Using a textpoint indirectly via the '^' notation.

```
text = "ole_1"
textpoint = "text"
PlayOLE(^textpoint, 0)
```

Note: This is only applicable to CX-Supervisor scripts and expressions, and is not supported by VBScript.

It is possible to use text points indirectly in place of point name arguments within script commands. For instance, each of the following commands has the same effect:

- Using a point name directly;

```
verbnumber = 0
PlayOLE("ole_1", verbnumber)
```

- Using a textpoint indirectly via the '^' notation.

```
verbnumber = 0
textpoint = "verbnumber"
PlayOLE("ole_1", ^textpoint)
```

An example using Indirection

The value of point indirection can be seen in a situation where it is necessary to dynamically change the pointname that an object is linked to. In the following example a toggle button is configured to control the Boolean state of one of four points:

- The four Boolean points to be controlled are called 'motor1', 'motor2', 'motor3' and 'motor4'.
- The text point 'textpoint' is used to store the name of the Boolean point to be controlled.
- The text point 'text' is used to store the string value of the integer point 'index'
- The integer point 'index' (which has a range 1-4) is used to dynamically change the point being controlled.
- Access to any of the four Boolean points 'motor1', 'motor2', 'motor3', 'motor4' can be achieved by applying indirection to 'textpoint' using the '^' notation and changing the contents of 'textpoint'.

For instance, in order to dynamically change the Boolean point a toggle button is linked to follow these steps.

1, 2, 3...

1. Link the toggle button to a textpoint using indirection e.g. ^textpoint.
2. Link the following script code to run as required. e.g. on clicking a button.
   - Text = ValueToText(index)
   - TextPoint = "motor" + text
3. The ValueToText function converts the integer value of the point 'index' into a string held in the textpoint 'text'. Therefore the point 'text' contains either '1', '2', '3' or '4'. The expression 'motor' + text appends the contents of the point 'text' to the literal string 'motor'. Therefore 'textpoint' contains either 'motor1', 'motor2', 'motor3' or 'motor4' dependent on the value of 'index'. Change the value of the 'index' to determine which Boolean point to control. e.g. via the Edit Point Value (Analogue) animation.

E.7 Point Arrays within Script Commands and Expressions

It is possible to access the elements of a point array directly or indirectly from within scripts or expressions.

- Setting the value of an array point directly;

```
arraypoint[2] = 30
```

- Getting the value of an array point directly;

```
value = arraypoint[2]
```

- "Setting the value of an array point using indirection;

```
textpoint = "arraypoint"
^textpoint[2] = 30
```

- Getting the value of an array point using indirection;

```
textpoint = "arraypoint"
value = ^textpoint[2]
```

An example using Point Arrays

The value of array points can be seen in a situation where it is necessary to dynamically change the pointname that an object is linked to. In the following example a toggle button is configured to control the Boolean state of one of four elements of an array point.

The Boolean array point 'motor' is configured to contain 4 elements.

The integer point 'index' (which has a range 0-3) is used to dynamically change the element of the point being controlled.

In order to dynamically change the element of a Boolean point that a toggle button is linked to follow these steps.

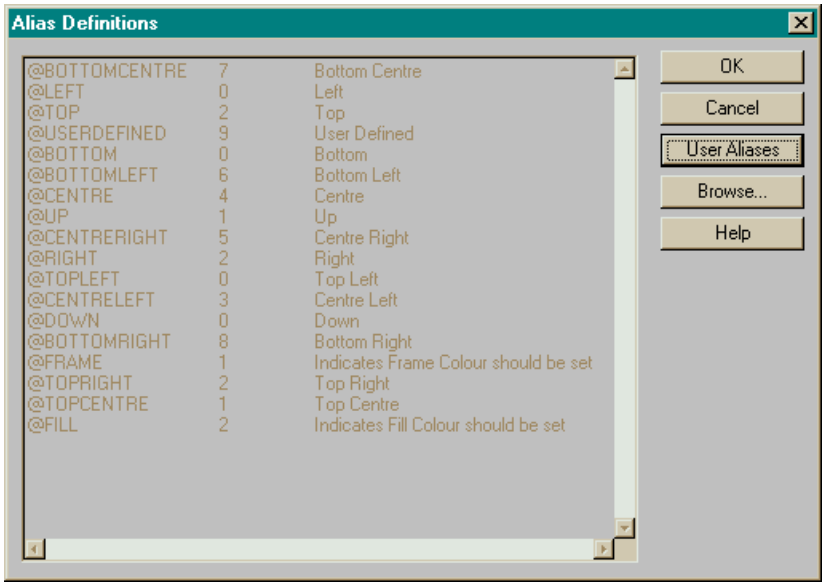
1, 2, 3...

1. Link the toggle button to an array point. e.g. 'motor[index]'.
2. Change the value of the 'index' to determine which element of the Boolean point to control. e.g. via the Edit Point Value (Analogue) animation.

E.8 Using Aliases

This facility is used to declare an alias - that is, to define a text string that can be used in place of another text string or a number within any script or expression. The Alias Definitions dialog is displayed by selecting the "Alias Definition..." option from the Project menu. It can also be displayed if "Aliases..." is selected from the script editor. The dialog displays either the User defined aliases or the preset System aliases and is toggled between these two displays by pressing the User/System Alias button.

The following illustration shows the Alias Definitions dialog displaying a number of User defined aliases. The System aliases are pre-defined and can not be edited or added to.



Syntax:

```
@AliasNameAlias definition 'optional comment
```

Remarks:

Argument	Type	Description
@AliasName	string	The string name of the alias
Alias definition	string	This is a string representing the actual text or expression of the expanded alias.
' comment	string	This is an optional comment.

The @ symbol at the beginning of each line initiates each alias command. For example, the text string @SomePoint could be used to represent any sequence of characters in a script or expression - e.g. it could be defined as:

```
@SomePoint = InArray[1]
```

or even

```
@SomePoint = Inarray[1] + Inarray[2] / 2
```

This is an easy way of identifying the individual members of array points. It can also be used to associate names with numbers, for example,

```
@SecondsPerDay = 86400
```

Alias definitions are stored in a simple text file in the project directory, called <project name>.pre. The format of the file consists of any number of lines such as:

```
@Test1 = InArray[12] * 10
```

i.e. an @ symbol followed by the name of the alias, then an equals sign (or space), followed by the definition of the alias. Anything that follows the last apostrophe (') symbol on a line is interpreted as a comment. Any line which does not start with the @ symbol is also assumed to be a comment.

Typical Examples

```
Declare boiler temperatures
@BoilerTemp1 = InArray[0] ' for boiler room 1
@BoilerTemp2 = InArray[1] ' for boiler room 2
```

```
@SecondsPerMinute  = 60   ' sets duration
```

Aliases may also be used to create a complicated expression such as

```
@HYPOTENUSEsqrt(Opposite  *  Opposite  +  Adjacent  *
Adjacent) 'Calculates length of Hypotenuse
```

This can be used in a script in the following way:

```
Opposite = 8.45
Adjacent = 9.756
length = @HYPOTENUSE
```

where Opposite, Adjacent and length are all REAL points.

Note: Changing an alias definition after it has been used in an expression or script will not automatically change the result in the script. The appropriate script or expression where that alias is used must be accessed and recompiled by pressing the OK button in order to apply the changes.

Appendix F
Glossary of Terms

ADO

ADO stands for Active Data Objects and is data access technology which uses OLE-DB to access data sources in a uniform way e.g. MS-Access databases, MS-Excel spreadsheets and Comma Separated Variable files.

AND

A logic operator used to interrogate Boolean type points. AND returns 'TRUE' if all arguments are 'TRUE'. An example of AND is that if a is a statement and b is a statement, AND returns 'TRUE' if both a and b are 'TRUE'. If one or both statements return 'FALSE' then AND returns 'FALSE'.

Application

A software program that accomplishes a specific task. Examples of applications are CX-Supervisor, CX-Server and Microsoft Excel. CX-Supervisor and its development environment allows the creation and testing of new applications through a Graphical User Interface (GUI).

Arguments

Words, phrases, or numbers that can be entered on the same line as a command or statement to expand or modify the command or statement within the CX-Supervisor script language. The command acts on the argument. In essence the command is a verb, and the argument is the object of the **verb**. An example of an argument in CX-Supervisor is "`DDETerminate(channel)`" where `DDETerminate` is a command within the script language, and `channel` is the argument upon which the command will act.

ASCII

An old standard, defining a set of characters. Officially using only 7 bits allows definitions for only 127 characters, and does not include any accented characters.

Bitmap

The representation of an image stored in a computer's memory. Each picture element (pixel) is represented by bits stored in the memory. In CX-Supervisor a bitmap image can be installed as a single object.

Boolean type

A type of point where the value of the point can be one of two states. Essentially the two states are '0' and '1', but these states can be assigned a meaningful designation. Examples are:

State	Example	Example	Example	Example
0	OFF	FALSE	OUT	CLOSED
1	ON	TRUE	IN	OPEN

COM

COM is a Microsoft technology that allows components used to interact.

Communications Driver	The relevant communications management system for OMRON PLCs in conjunction with Microsoft Windows, providing facilities for other SYSMAC software to maintain PLC device and address information and to communicate with OMRON PLCs and their supported network types.
Constant	Within CX-Supervisor, a constant is a point within the script language that takes only one specific value.
Control Object	In CX-Supervisor, a control object is applied in the development environment and can be a pushbutton, a toggle button, a slider, a trend graph, a rotational gauge or a linear gauge. Essentially a control object can be a complex graphic object consisting of a number of primitive graphic objects, which provides user interaction.
CX-Server	An advanced communications management system for OMRON PLCs providing facilities for software to maintain PLC device and address information and to communicate with OMRON PLCs and their supported network types. CX-Server supports CS-Series PLCs.
Database connection	A Database connection (or Connection for short) contains the details used to access a data source. This can either be via Data Source Name (DSN), filename or directory.
Database Connection Level	A Database Connection Level is a string which determines what level in the database tree hierarchy is to be operated on. Some examples are listed below:

"Northwind"	Connectionlevel
"CSV.Result"	Recordset level
"Northwind.Order Details.OrderID"	Field level
"Invoice.Data Types"	Schema level

Database Recordset	A Database recordset (or Recordset for short) is a set of records. This could either be an actual Table in the database, or a table that has been generated as a consequence of running a Query.
Database Schema	A Database Schema (or Schema for short) obtains database schema information from a Provider.
Database Server Query	A Database Server Query (or Server Query for short) is a query that is stored in the actual Database. They are pre-defined and added by the database designer which means they are 'fixed' for the duration of a project. Server Queries may have pre-defined 'Parameters', which allow criteria to be passed to the query at runtime e.g. values to filter, allowing one query to be used to produce different results. Each pre-defined parameter must have a Parameter Association defined. Because these queries are stored in a compiled and tested form they are more efficient and therefore preferential to running a SQL Query.

Database SQL Query	A Database SQL Query (or SQL Query for short) is interpreted dynamically at runtime. The SQL Text can be modified at runtime, enabling different Queries to be run for varying situations however, the SQL Text has to be compiled on the fly every time it is executed and consequently is less efficient than a Server Query.
DBCS	DBCS stands for Double Byte Character Set and is a Microsoft extension of ASCII which uses 2 bytes (16 bits) to define character codes. With this larger range it can include accented characters, extended ASCII characters, Nordic characters and symbols.
DCOM	DCOM is a distributed version of COM that allows components on different PCs to interact over a network.
DDE	Dynamic Data Exchange. A channel through which correctly prepared programs can actively exchange data and controls other applications within Microsoft Windows. DDE technology was notoriously unstable and was replaced with OLE technology. See also Item, Server, server application and Topic.
Development Environment	SCADA applications are created and tested using the development environment within CX-Supervisor. On completion, the finished application can be delivered as a final customer application to be run by the run-time environment.
DLL	Dynamic Link Library. A program file that although cannot be run stand-alone as an executable, can be utilised by one or more applications or programs as a common service. DLL files have a *.DLL extension. DLL's comprise a number of stand-alone functions. In CX-Supervisor, a DLL containing icons can be accessed to represent the display part of an OLE object. One such DLL, 'MORICONS.DLL', is provided in the standard Microsoft Windows installation.
Download	A recipe is downloaded during runtime. This process involves identifying the appropriate recipe and executing the validation code, if any exists. The download is complete when each ingredient has set its point to the target value.
Executable	A file that contains programs or commands of an application that can be executed by a user or another application. Executable files have a *.EXE file extension. CX-Supervisor provides two executable files, one for the development environment (CXSUPERVISORDEV.EXE), and one for the run-time environment (SCS.EXE).
Expressions	In the CX-Supervisor script language, expressions are a construct for computing a value from one or more operands. For instance, in the example "`lift = height + rate`", the expression is "`height + rate`" where the result yielded from the expression is used for the value of "`lift`".

	Outside of the script language, expressions consisting of operators and operands can be used to control objects, through actions.
Field association	A field association enables a link to be made between a CX-Supervisor Point and a particular field (i.e. column) within a recordset.
Graphic Object	In CX-Supervisor, a graphic object is created in the development environment, and can be a line, an arc, a polygon (including a square and rectangle), a round rectangle, an ellipse (including a circle), or a polyline. A complex object can exist as a combination of two or more graphic objects.
GUI	Graphical User Interface. Part of a program that interacts with the user and takes full advantage of the graphics displays of computers. A GUI employs pull-down menus and dialog boxes for ease of use. Like all Microsoft Windows based applications, CX-Supervisor has a GUI.
I/O type	Input/Output type. An attribute of a point that defines the origin and destination of the data for that point. The data for a point can originate (be input from) and is destined (is output to) to the internal computer memory or PLC.
Icon	Pictorial representations of computer resources and functions. The CX-Supervisor development environment and run-time environment are run from icons.
Ingredient	Each recipe consists of at least one ingredient. Each ingredient must be related to an existing point.
Integer type	A type of point where the value of the point can only be a whole positive or negative number.
Item	Within the CX-Supervisor script language, Item is a generic term for a point, OPC item or Temperature Controller item.
JScript	A Java style scripting language supported by Microsoft's Windows Scripting Host.
JVM	Java Virtual Machine.
Microsoft Excel	A spreadsheet application.
Microsoft Windows	A windowing environment that is noted for its GUI, and for features such as multiple typefaces, desk accessories (such as a clock, calculator, calendar and notepad), and the capability of moving text and graphics from one application to another via a clipboard. CX-Supervisor will run only under Microsoft Windows. DDE functions communicating with other applications supported by CX-Supervisor use Microsoft Windows as a basis.
Microsoft Word for Windows	A word processing application.

Nesting	To incorporate one or more IF THEN ELSE/ELSEIF ENDIF statements inside a structure of the same kind.
Network	1 - Part of the PLC configuration, based on the device type. The number of Networks available is dependent on the device type. 2 - A number of computers linked together with a central processing point known as a Server which is accessible to all computers. Networks affect CX-Supervisor in that further Network associated options are available if the computer is Network connected.
Non-Volatile	A point that is designated as 'non-volatile' is a point whose value is saved on disk and automatically reloaded when CX-Supervisor resumes execution.
NOT	A logic operator used to interrogate Boolean type points which produces the Boolean inverse of the supplied argument. An example of NOT is that if a is a statement and is 'FALSE', then NOT returns 'TRUE'. If a is a statement and is 'TRUE', then NOT returns 'FALSE'.
Object	In CX-Supervisor, an object can be text, graphics, a control, a bitmap, or ActiveX object as created in the development environment. A complex object can exist as a combination of two or more objects of any of the above types. Specifically, graphical objects can be categorised as a line, an arc, a polygon (including a square and rectangle), a round rectangle, an ellipse (including a circle), or a polyline. A control is essentially a complex graphic object and is specifically either a pushbutton, a toggle button, a slider, a trend graph, a rotational gauge or a linear gauge.
OLE-DB	OLE-DB is the underlying database technology, on which ADO relies. OLE-BD is designed to be the successor to ODBC.
Operand	The term used for constants or point variables.
Operator	A symbol used as a function, with infix syntax if it has two arguments (e.g. "+") or prefix syntax if it has only one argument (e.g. NOT). The CX-Supervisor script language uses operators for built-in functions such as arithmetic and logic.
OR	A logic operator used to interrogate Boolean type points. OR returns 'TRUE' if any of the supplied arguments are 'TRUE'. An example of OR is that if a is a statement and b is a statement, OR will return 'TRUE' if either a and b are 'TRUE'. If both statements return 'FALSE' then OR will return 'FALSE'.
Pages	The combination and manipulation of pages containing objects within projects forms the basis of CX-Supervisor. More than one page can exist for each project. The pages in a project provide the visual aspect of CX-Supervisor corresponding to a display with the objects contained in each page providing a graphical representation of the system being monitored.

Parameter Association	A Parameter Association enables values, either constant or stored in a point, to be passed to a Server Query.
Pixel	A single displayable point on the screen from which a displayed image is constructed. The screen resolution of the computer's Visual Display Unit (VDU) is defined by the number of pixels across and the number of pixels down (e.g. 1024 x 768). See also SVGA mode and VGA mode.
PLC	Programmable Logic Controller.
Point variable	A point within the CX-Supervisor script language that stores a value or string assigned to that point.
Point	A point is used to hold a value of a predefined type - Boolean, Integer, Text, etc. The contents of a point may be controlled by an object or I/O mechanism such as PLC communication. The contents of a point may control the action or appearance of an object, or be used for output via an I/O mechanism. See also Boolean type, Integer type, point variable, Real type and Text type.
Project	A CX-Supervisor application will consist of one or a number of pages linked together. The pages may contain passive or active graphics, text or animations, and may be grouped together logically to form a project. A project may consist of many pages, or simply a single page. Projects may be built and tested within the CX-Supervisor development environment, and run stand-alone under the CX-Supervisor run-time environment. Only one project at a time may be open for editing within the CX-Supervisor development environment.
Real type	A type of point where the value of the point can be any number, including those containing a decimal point.
Recipe	A recipe is a set of pre-defined steps used to perform a particular task. A CX-Supervisor project may contain zero or more number of recipes. Recipes are defined in the development environment and executed, or downloaded, in the run-time environment.
Run-Time Environment	SCADA applications are run using the run-time environment of CX-Supervisor, following creation of the application in the CX-Supervisor development environment.
SCADA	Supervisory Control and Data Acquisition.
Server	A Server is the central processing point of a Network that is accessible to all computers. Networks affect CX-Supervisor in that further associated options are available if the computer Network is connected.
Server Application	An application that can be used to view or interact with, whilst currently within CX-Supervisor.

Statement	Within the CX-Supervisor script language, a statement is a command understood by the run-time environment. Statements are constructed of commands and arguments, which when combined, help to formulate a finished application to be used in the run-time environment.
String	The contents of a Text type point that can only contain literal alphanumeric characters. A string starts following an opening quotation mark, and ends before a closing question mark; in the example "name = "spot"", the point "name" holds the string spot.
SVGA mode	A mode of video display that provides 800 600 pixel resolution (or higher) with 16 or more colours and is supported on Super Video Graphics Adapter systems.
CX-Supervisor	A SCADA software application which creates and maintains graphical user interfaces and communicates with PLCs and other I/O mechanisms.
Target Value	An ingredient must specify a target value for its related point. This is the value to which the point will be set in runtime when the recipe is downloaded.
Taskbar	An integral part of Microsoft Windows which allows Microsoft Windows based applications to be started. CX-Supervisor is run from the Taskbar.
Text Object	In CX-Supervisor, a text object is a string on a page. Attributes such as typeface, point size, embolden, italicise, underline, left justify, flush right, and centre can be applied to enhance its presentation.
Text Type	A type of point that holds a string.
Unicode	A Multi-Byte Character Set, which not only includes European Characters like DBCS, but can also include global support including for Japanese, Chinese and Cyrillic fonts. However, Unicode is not supported on all Windows platforms.
Validation Code	Recipe validation code is VBScript or CX-Supervisor Script language which is used to check point values before downloading a recipe.
VBScript	A Visual Basic style scripting language supported by Microsoft's Windows Scripting Host.
VGA mode	A mode of video display that provides 640 480 pixel resolution with 16 colours and is supported on Video Graphics Adapter systems.
Windows Desktop	An integral part of Microsoft Windows which allows Microsoft Windows based applications to be started from icons and for all applications to be organised. CX-Supervisor can be run from Windows Desktop.
Windows Scripting Host	A scripting engine supplied by Microsoft to run VBScript or JScript. See http://msdn.microsoft.com/scripting

Wizard	Wizards are dialogs used by the CX-Supervisor development environment to take the user through complex operations in a simplified step-by-step process.

Revision history

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.

Cat. No. W09E-EN-12

The following table lists the changes made to the manual during each revision. The page numbers of a revision refer to the previous version.

Revision code	Date	Revised content
01	Sept. 2010	First version in the standard Omron format.
02	June 2011	Updated for CX-Supervisor 3.2 release.
03	March 2017	Updated for CX-Supervisor 3.3 release.
04	Oct. 2017	Updated for CX-Supervisor 3.4 release.
05	June 2018	Updated for CX-Supervisor 3.4.1 release.
06	Dec. 2018	Updated for CX-Supervisor 3.5 release.
07	Feb. 2020	Updated for CX-Supervisor 4.0 release.
08	June 2021	Updated for CX-Supervisor 4.1 release.
09	Jan. 2022	Updated for CX-Supervisor 4.2 release.
10	Sept. 2022	Updated for CX-Supervisor 4.2.1 release.
11	March 2023	Updated for CX-Supervisor 4.3 release.
12	April 2024	Updated for CX-Supervisor 4.4 release.

OMRON

Authorized Distributor:

Cat. No. W09E-EN-12

Note: Specifications subject to change without notice.

Printed in Europe