CX-Server Lite
User Manual

Getting Started
Version 2.1

Notice

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify precautions in this manual. Always heed the information provided in them. Failure to heed precautions can result in injury to people or damage to the product.

DANGER! Indicates information that, if not heeded, is likely to result in loss of life or serious injury.

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

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

OMRON Product References

All OMRON products are capitalised in this manual. The word “Unit” is also capitalised when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation “PLC” means Programmable Logic Controller and is not used as an abbreviation for anything else.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

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

1, 2, 3… Indicates lists of one sort or another, such as procedures, checklists etc.

Represents a shortcut on the Toolbar to one of the options available on the menu of the same window.

Indicates a program must be started, usually by clicking the appropriate option under the standard Windows 'Start' button.

Note: Indicates procedures that are specific to Visual Basic.

About this Manual

This manual describes the CX-Server Lite application and its ability to interface with OMRON CS, CV and C PLCs. It does not provide detailed information concerning the PLCs themselves, for this information the commercial manual for the device must be consulted.

This manual contains the following information:

- ***Getting Started with CX-Server Lite:*** This describes the CX-Server Lite software in general terms
- ***Tutorial:*** This is a quick tutorial for Excel and Visual Basic host applications.
- ***Appendix A Component Properties:*** This appendix summarises the available properties for the ActiveX components.
- ***Appendix B Script Interface:*** The Visual Basic script interface for the CX-Server communications control.
- ***Appendix C Temperature Controller Support:*** The supported functionality for temperature controller devices.

A ***Glossary of Terms*** and ***Index*** are also provided.

Warning:	Failure to read and understand the information provided in this manual may result in personal injury or death, damage to the product, or product failure. Please read each chapter in its entirety and be sure you understand the information provided in the chapter and related chapters before attempting any of the procedures or operations given.

Table of Contents

Getting Started with CX-Server Lite

This book introduces the CX-Server Lite application to a new user.

Welcome to CX-Server Lite

CX-Server Lite allows PLC data collected by the OMRON CX-Server communications software to be accessed from Microsoft Excel (97 and later) and Visual Basic (6.0 and later). It allows existing process data to be collected and analysed, plus it includes some graphical components allowing easy creation of simple MMI applications.

Included with CX-Server Lite is the CX-Server runtime system, plus a range of ActiveX components that can be dragged and dropped onto your Workbook or Form.

About this Manual

This manual helps a new user get started with CX-Server Lite, by describing the software installation and computer configuration, and by leading the user through the basics of CX-Server Lite. For the most up to date information see the on-line help or the release notes in the installed directory.

Separate OMRON manuals describe the related CX Automation Suite products; CX-Server, CX-Programmer and CX-Supervisor.

CX-Server Lite comes with a comprehensive on-line help system, which is designed to complement this manual, and provide a quick reference at any point in the CX-Server Lite application when the manual is not to hand. This general help system uses a fast 'hypertext' system that allows progressively more information to be obtained about any topic by selecting keywords within the descriptive text.

Throughout this manual, it is assumed that a working knowledge of Microsoft Windows is obtained, and that the user can:

- Use the keyboard and mouse.
- Select options from Windows menus.
- Operate dialog boxes.
- Locate, open and save data files.
- Edit, cut and paste text.
- Drag and drop.
- Start programs from the “START” button.

If Windows has not been used before, it is recommended that some time working with the Microsoft documentation is spent before using CX-Server Lite.

This introductory section deals with several important aspects of installing CX-Server Lite and setting it up for use. It is recommended that this entire section be read before installing the software.

System Requirements

CX-Server Lite operates on IBM compatible personal computers with 1 GHz+ Pentium central processor. It is designed to run in the Microsoft NT 4, 2000, XP and Vista environments.

Note: CX-Server Lite is not guaranteed to be compatible with computers running Windows emulation (e.g. Apple Macintosh).

Hardware Requirements

The following configuration is the **minimum** system requirements for running CX-Server Lite

- IBM PC compatible Pentium II processor
- 512 Mbytes of RAM
- 30 Mbytes available hard disk space,
- 800 x 600 Super VGA display.

The Recommended **minimum** is:

- IBM PC compatible Pentium 4 processor,
- 1024 Mbytes of RAM
- 50 Mbytes available hard disk space,
- 1024 x 768 Super VGA display.

Operating Systems and Environments

The operating systems on which this software will run are:

- Microsoft Windows NT 4.0 (Service Pack 5 and later),
- Microsoft Windows 2000 (Service Pack 2 and later)
- Microsoft Windows XP (Professional is recommended)
- Microsoft Windows Vista (Business or Ultimate Edition is recommended)

Containers in which this software will run are:

- Microsoft Excel 97 and later,
- Microsoft Visual Basic version 6.0 and later.
- Microsoft Visual C++ 6.0.
- CX-Supervisor v1.1 and later (v2.0 and later recommended)

The *recommended* Operating System is Microsoft Windows XP Professional or Vista Business.

For use in Visual Studio 2008 and Microsoft .Net, see the separate “Guide to Using CX-Server Lite in Microsoft .Net”

Interfaces to Hardware - PLC Communications

Interfaces to PLC hardware are via the CX-Server runtime system.

Interfaces to Network Service Boards (NSBs) are achieved using Fins Gateway, which is supplied and supported as part of the CX-Server product.

Interfaces to Hardware - Peripherals

Interfaces to PC hardware (printers, graphics, keyboard, mouse, Ethernet etc.) are supported by drivers installed and supported by Windows.

Installing/Uninstalling CX-Server Lite

The CX-Server Lite software is supplied on CD-ROM and is installed easily from within Windows.

To install CX-Server Lite

1, 2, 3…

1. Close all programs.
2. Insert the CD labelled **CX-Server Lite** into your CD-ROM drive. If Autorun is enabled on your system, the installation starts automatically, otherwise see the README.TXT on the CD-ROM for instructions to launch manually.
3. Follow the instructions on the screen.

By default, CX-Server Lite is installed in **C:\Program Files\Omron**. Additional Omron applications will be installed in a subfolder under Omron. Common components are installed into **C:\Program Files\Common Files\Omron\Components.**

To uninstall CX-Server Lite

When you uninstall, your CX-Server Lite project data remains intact – uninstall only removes program files. It is recommended, however, that you copy and move any projects you saved in the CX-Server Lite folder. Save a copy of your projects in another location on your hard drive (such as your My Documents folder) before uninstalling CX-Server Lite.

1, 2, 3…

1. From the **Start** menu, select **Settings – Control Panel**.
2. Double-click the **Add/Remove Programs**.
3. Click the **Install/Uninstall** tab.
4. From the list programs that you can remove, select **CX-Server Lite**.
5. **Click Add/Remove**.
6. At the prompt, select the **Remove** item and click **Next** and follow any further prompts.
7. Wait until the uninstall program indicates that the process is complete.

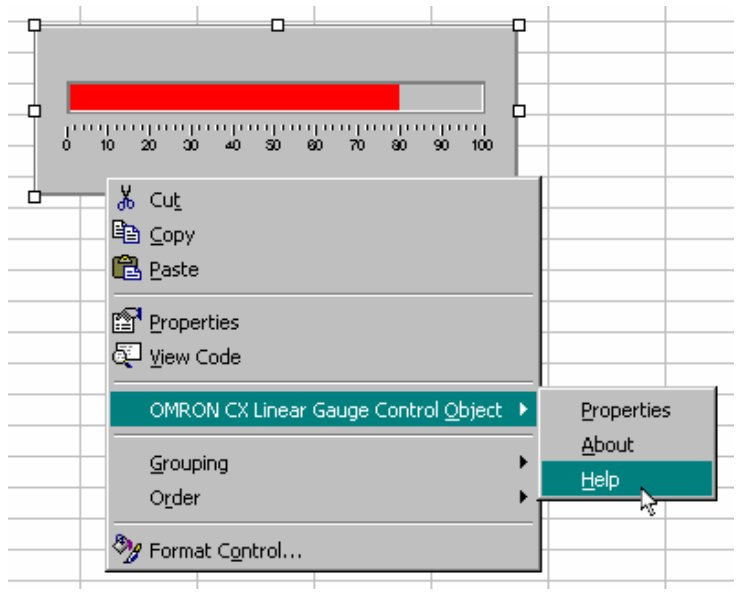
The Help system, and How to Access it

CX-Server Lite comes with a detailed help system. At any time while using the software, it is possible to get help on a particular point that is currently being worked on, or on general aspects of CX-Server Lite. This system is intended to complement the manual, by providing on-

line reference to specific software functions and how to use them. This manual is designed to provide tutorial information and discuss the various facilities offered by CX-Server Lite.

Help Topics

Select the *Help* option on the *Object Properties* menu, which can be accessed by clicking on an object using the right mouse button and then selecting the option which shows the object name.



The help system provides a standard look-up dialog under the Contents tab showing the contents of the CX-Server Lite Help file. Double-click on an item to read the associated information.

Index

Use the following procedure to retrieve on-line help from the *Index* tab of the Help dialog.

1, 2, 3…

1. Select the *Help* option from the Object Properties Menu.
2. Select the *Index* tab.
3. Enter a text query into the first step field. The second step field is refreshed according the to query entered in the first step field.
4. Select an entry in the second step field and select *Display* pushbutton, or double-click on the index entry.
5. If an entry is linked to two or more topics, the names of the topics are displayed in the Topics Found dialog. Select a topic and choose the *Display* pushbutton or double-click in the topic.

Find

Use the following procedure to retrieve on-line help from the *Find* tab of the Help Topics dialog.

1, 2, 3…

1. Select the *Help* option from the Object Properties Menu.
2. Select the *Find* tab.
3. Enter a text query into the first step field. The second step field is refreshed according the to query entered in the first step field. Previous text queries can be retrieved by selecting from the drop down list in the first step field.
4. Select a word that matches the query – some words may be automatically selected. More than one word can be selected by pressing Shift and selecting another word to extend the selection or by pressing Ctrl and selecting another word to add to the selection. The third step field is refreshed according to the word or words selected. The number of topics found is shown at the bottom of the dialog.
5. Select a topic from the third step field and select the *Display* pushbutton, or double-click on the topic from the third step field. Select the *Clear* pushbutton to restart the find operation.

The Find operation can be enhanced by the use of the *Options* pushbutton and Rebuild pushbutton. Refer to *Microsoft Windows documentation* for further information.

About CX-Server Lite

CX-Server Lite Components include an About dialog accessible from the properties menu. The About dialog supplies technical reference information about the application such as version and copyright information. It also contains essential version number information that is required for obtaining technical support. The CX-Server Communications Control also includes details of the version of CX-Server installed.

In addition, a brief description of CX-Server Lite and the CX-Automation Suite can be accessed from the main help contents dialog.

Technical Support

If the installation instructions for this application have been followed, no difficulties should be encountered.

If a problem occurs, check that it does not relate to a fault outside CX-Server Lite, for instance, with external components. Check the following:

- The PC is working correctly,
- The external system or application is working correctly,
- The communications system is set up correctly,
- Any errors are cleared in the associated PLCs.

When Customer Services need to be contacted, keep the following details to hand. A clear and concise description of the problem is required, together with the exact text of any error messages.

Note: Use the About dialog to obtain the version number of the application.

Objects Overview

CX-Server Communications Control

This control provides a seamless interface between the CX-Server Lite host application (Excel, Visual Basic) and Omron's communication software, CX-Server. Note that the control is only visible when the host application is in the Design mode.

The Properties dialog enables you to select the project you wish to open. Clicking the **Open…** button will open the Open Project dialog allowing you to navigate to the appropriate file.

Clicking the Edit Project button opens the CX-Server Project Editor dialog. This dialog is supported by CX-Server and follows the standard Windows Explorer format. The left hand pane shows the tree structure for the project. By expanding the tree the associated PLCs and Points etc. can be reviewed and edited as necessary. New PLCs and points can be added by right clicking in the right hand pane and selecting **New** from the menu. Consult the associated help file for more detailed information on editing.

Temperature controller devices are supported in this version of CX-Server LITE. See Script Interface support and Appendix C for detailed information regarding the available functionality.

7 Segment

The 7 Segment control displays a value in Binary, Decimal or Hexadecimal format. Leading zeros and unused segments can be hidden. The colour of the segments and the display background can be set independently. The 7 Segment control cannot be used to set a value.

Data Logging

The Data Logging control provides logging and trending functionality through use of the Data Log Viewer components currently used by other Omron software packages including CX-Supervisor and SYS-Config. The control is configured in design-mode to log data items and is controlled in runtime-mode using script commands. See the on-line help for further details regarding the script interface.

Display

The Display displays an analogue or text value. The Display only displays a value i.e. you cannot set a value using this display.

LED Indicator

The LED functions as a coloured on/off indicator. The colour of the indicator and the display background can be set independently while its shape can be round or square. In the off state, the chosen indicator colour is dimmed.

Linear Gauge

The Linear Gauge displays an analogue value by filling a rectangle to represent the actual value as a proportion of its expected maximum. The rectangle can be filled from bottom to top (like a thermometer) or from left to right (like a progress complete bar). There is also a configurable scale, enabling intermediate values to be estimated. The Linear gauge will only display a value, you cannot set a value with this gauge.

Linker

This control gives the ability to link COTS (commercial off the shelf) ActiveX components to any of the Omron communications controls, e.g. the CX-Server communications control. The control is configured in design-mode to select the ActiveX component (e.g. a Microsoft Forms V2.0 check box control) to which the control will link at runtime. In runtime mode data will be read from and written to the selected PLC item and the selected ActiveX component.

Note: In this version, the linker cannot link text points or array points only single element points.

Rotational Gauge

The Rotational Gauge displays an analogue value, similar to a speedometer. An indicator needle rotates according to the value. There is a configurable scale, enabling intermediate values to be estimated. The Rotational gauge will only displays a value, you cannot set a value with this gauge.

Rotary Knob

The Rotary Knob allows you to set an analogue value, similar to a volume knob. You can rotate the knob, e.g. by clicking and dragging the mouse, to set the pointer to a new position. There is a configurable scale, enabling intermediate values to be estimated. The pointer always reflects the current value e.g. on start-up, and will change position in response to an external influence.

Toggle

The Toggle allows you to toggle a Boolean bit between its 'On' and 'Off' state. This is as a switch that can be clicked to change its state. The current state is shown by the position of the switch. The switch position also reflects the current value e.g. on start-up, and will change position in response to an external influence.

Timer

The timer enables you to run a set of instructions repeatedly at regular intervals.

Thumbwheel

The Thumbwheel provides a set of input controls, similar to a hardware Thumbwheel Switch. By clicking minus and plus buttons, the various input digits can be set. There are two modes of operation; Commit and Direct. When the optional *Commit* button is enabled, digit values may be edited freely. The PLC will not receive an updated value until the Commit button is pressed. In *direct mode* [without the optional Commit button] changes to digit values are sent direct to

the PLC as they occur. Floating point is supported, and integer values can be represented in both decimal and hexadecimal.

Tutorial

Important: This tutorial describes operation in Visual Basic and Excel. For details of operation in Microsoft Visual C++ please see the separate CX-Server Lite Communications Control C++ Tutorial.

The following sections take you through the steps required to open your selected application, i.e. Excel or Visual Basic and create a working area. Using the short tutorial you can then continue and load a number of ActiveX objects, link them together and run a simulation.

As you became more practised in using CX-Server Lite you will find there is usually more then one way to perform an operation. The following procedures may not always be the quickest but have been written to show how the application works using the basic features.

If the ActiveX objects are not visible in the Visual Basic Toolbox they can be added as follows:

1, 2, 3…

1. Right click in the Toolbox and select the **Components…** option. This will open the Components dialog.
2. Find the CX-Server Lite controls in the list, all of which all start with **OMRON CX**, and tick each box.
3. Click the **OK** button. The objects are now displayed in the Toolbox.

Step 1: Viewing PLC Data using Omron Graphical Controls

Adding the Communications Control

Before the Graphical Controls objects of CX-Server Lite can communicate with a PLC the correct data source connections have to be set up for it. This is not necessary if the Graphical Control will be used stand alone and driven from script.

To add a Communications Control:

1, 2, 3…

1. Start the host application e.g. Microsoft Excel 97.
   Tip: If you intend having a large number of components on your desktop it is recommended you run Excel in full screen mode.
2. Ensure the **Control Toolbox** and **CX-Server Lite** toolbars are shown by selecting them from the **View**, **Toolbar** menu.

   Note: In Visual Basic, ensure the **Toolbox** is shown by selecting **Toolbox** from the **View** menu.

3. Ensure the host application is in design mode for example, in Excel by clicking the Design Mode button in the **Control Toolbox**.

4. In the **CX-Server Lite** toolbar click the **Add CX-Server Communications Control** button. The Communications Control object is drawn in the default position - top left hand corner of the work area.

Note: In Visual Basic, select the required component from the **Toolbox** and draw a rectangle at the desired position.
Tip: Double clicking the toolbox button inserts the component with a default size.

5. Using Drag and Drop the object can now be repositioned in the work area. Note that the object will not be visible in run mode.

Connecting the Communications Control to a PLC

The first step is to create a CX-Server project file (.CDM file) or select one which has been created by another application (e.g. CX-Programmer or SYSMAC-SCS). This file contains PLC configuration data and symbolic definitions.

The following procedure takes you through the steps required to load an existing .CDM file or create a new one.

1, 2, 3…

1. Right click on the CX-Server communications control. In the popup menu, select the **OMRON CX Communications Control Object** option.

   Note: In Visual Basic, the menu option is called **Properties**.

2. In the Communication Control Properties dialog 'Project' field enter or select the following.

   i. **To open an existing CX-Server project (.CDM) file:**

   - Click the **Open…** button and in the Open Project dialog navigate to the file you wish to open.
   - When you click the **Open** button, the full path name of the selected file will be entered into the Project field.

   Caution: When sharing a CDM file with another applications it is important to realise that any changes that are made to the CDM file may affect the other applications.

   ii. **To create a new CX-Server project (.CDM) file:**

   - Click the **New…** button and in the Create Project dialog navigate to the directory in which you wish to create the new file.
   - In the File Name field, enter the desired file name. When you click the **Save** button, the full path name of the new file will be entered into the Project field.

3. Projects can be edited by clicking the **Edit Project…** button and then making the required changes from the Project Edit Dialog. This is a CX-Server runtime utility – invoke help from within it for details of how to add and configure PLCs and Points.
4. Click the **OK** button to complete the configuration.

The communications control is now ready to connect to CX-Server, and retrieve data from PLCs. This data can be accessed using script commands (see Appendix B), or by adding a Graphical Component.

Adding a 7 Segment Display

1, 2, 3…

1. With the host application in design mode, add a 7 Segment control.
2. Right click on the graphical component and from the popup menu select the **OMRON CX 7 Segment Control Object** option.

   Note: In Visual Basic, the menu option is called **Properties**.

3. In the component properties dialog select the **Data Source** tab and enter the following information:
   - ***Server:*** - Select the name of the communications control to be used. If only one has been added, it is selected automatically. If the list is empty then you need to add one first.
   - ***PLC:*** - Select the required PLC. If the appropriate PLC is not in the list click the **>** button and select **Add PLC…**. The Add PLC dialog is part of the CX-Server runtime. For further information on adding PLCs refer to the CX-Server User Manual or the online help.
   - ***Item:*** - Select the point Item. If the appropriate item is not in the list click the **>** button and select **Add Item…**.. The Point Editor dialog is part of the CX-Server runtime. For further information on adding Points refer to the CX-Server User Manual or the online help. The Item field will also accept physical addresses e.g. “DM100” instead of defining logical addresses.
   - ***Update Rate:*** - Enter the rate, in seconds, at which the data is updated.

     Note: This value should be chosen carefully. If the update rate is set low it will increase the volume of data being transferred, and may cause the system to slow down or stop responding.
4. Click **OK** to complete the connection.

Running the Application

Click the Mode button to change to ‘Run’ mode. The communications control will disappear and will connect to the PLC. After a short delay, depending on the PLC communications medium, the 7 Segment display will show the PLC value.

Note: In Visual Basic, click  to switch to run mode, and  to return to design mode.

Step 2: Inserting PLC Data in Cells

Step 1 shows PLC data in a graphical control, but the PLC data can also be inserted directly into cells within Excel. This could be useful for further numerical analysis, like averaging or statistical control.

Note: In Visual Basic, there is no concept of cells, but the same technique could be used to set a Visual Basic variable with a PLC value.

Assuming Step 1 above has been completed:

1, 2, 3…

1. Decide when the data should be updated:
   i. For the user to control the data update, add a standard **Command Button** from the **Control Toolbox**.
   ii. For the value to constantly update on a regular interval, add the Omron timer control.
2. Double click the added object to access the script and add the line:

```
Cells(1, 1) = Comms1.Value("MyPointName")
```

   where MyPointName is the logical name of the item to read.
3. Close the Visual Basic editor, and run the application as shown in Step 1. The cell A1 (that is row 1, column 1) will show the required data.

Step 3: Adding Third Party ActiveX Controls

Step 1 shows PLC data in an Omron graphical control, but the PLC data can used by other ActiveX controls, like Graphical Control Libraries supplied by other manufacturers, or controls like Charts or Scroll Bars supplied with Microsoft products.

Assuming Step 1 above has been completed, the following steps show connecting the data to a standard Scroll Bar:

1, 2, 3…

1. Add a standard **Scroll Bar** from the **Control Toolbox**.
2. Select and resize the buttons as required then drag and drop the buttons in the desired position.

3. Double- click on the scroll bar buttons. This will reposition the cursor in the code sheet at the following entry.

```
Private Sub Scrollbar1_Change()
|
End Sub
```

Note: In Visual Basic, the default name for scroll bars is **HScroll1** or **VScroll1.**

4. Add the following syntax. The additional command instructs any new Scroll Bar value to be written to the PLC.

```
Private Sub Scrollbar1_Change()
    Comms1.Value("MyPointName") = Scrollbar1.Value
End Sub
```

Run the application as shown in Step 1. The communications control will disappear and the 7 Segment display will show the PLC value. When you click on the scroll bar buttons the value is sent to the PLC. This new value is then shown on the 7 segment display. Note that the maximum value is limited to 100. This is the default value of the scroll bar buttons.

This example shows a control setting a PLC value. Third Party controls can also display PLC values. For this, the syntax would be (depending on actual control):

```
ControlName.Value = Comms1.Value("MyPointName")
```

This script could also be added to a button or Timer control, as explained in Step 2. Script functions are described further in Appendix B.

Maximum Active Communications Controls

The number of active communications controls should be less than twenty (e.g. if communications controls have been added on a one-per-form basis, no more than nineteen forms should be open at any one time).

Improving Performance

When doing any reading or writing to or from a PLC, it is important to be aware of the fact that if the PLC is not open, then the read or write command will cause it to be opened, and then closed after the operation 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.

It is recommended that the number of communications controls that are active at any one time (e.g. on open forms in Visual Basic) is kept as low as possible. It is often possible to have just one per project. The main limitation with doing that is that the communications control needs to

be driven programmatically, and any graphical controls need to be driven directly (e.g. the value can be read by the program from the communications control, and then written to a graphical control). This is because graphical controls can only connect *directly* to a communications control on the same Visual Basic form.

Other Features

The following sections provide a brief overview of some of the more advanced features available in CX-Server Lite.

Event Driven Routines

Many of the script examples in this manual use asynchronous communications, that is communications are carried out on demand without synchronisation with the rest of the system. Asynchronous communications can be easily used to quickly create solutions that are easy to understand. As a solution grows however, asynchronous communications can prove inefficient and produce unpredictable updating, which is difficult to debug because multiple scripts may be demanding the same data at the same time.

The CX-Server Communications Control provides facilities for synchronous communications, that is communications and data updating are synchronised. The **GetData** and **StopData** script commands (see Appendix B for full details) control the generation of **OnData** events on regular intervals, which can be used to efficiently drive multiple controls, and is easier to debug.

Example:

1, 2, 3…

1. Add an **OMRON CX-Server Communications Control** and two standard **Command Buttons**.
2. Double click the communications control to add the following script. Note the script is in the Event **OnData**:

```
Private Sub Comms1_OnData(ByVal PLC As String, ByVal
Point As String, ByVal Value As Variant, ByVal
BadQuality As Boolean)
       If (Point = "MyPointName") Then
         'Data is from this point
         Cells(1, 1) = Value
       End If
End Sub
```

Every time **OnData** is called with data from the point **MyPointName** the value is written to the cell A1.

3. Double click the Command Buttons to add the following script:

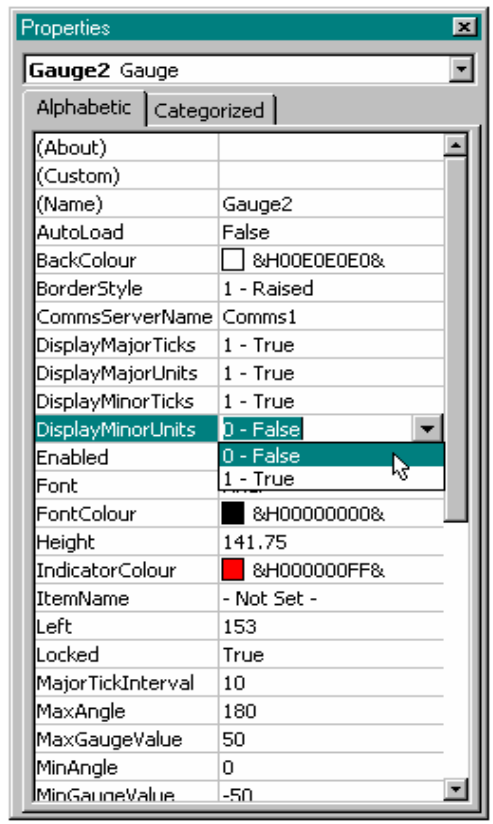
```
Private Sub CommandButton1_Click()
        Comms1.GetData("MyPLC", "MyPointName", 1.0, 0)
End Sub

Private Sub CommandButton2_Click()
        Comms1.StopData("MyPLC", "MyPointName)
End Sub
```

Run the application. Click CommandButton1 to start creating OnData events every second. Note the cell A1 updating. Click CommandButton2 to stop the updating.

Advanced Properties

When working with Visual Basic the advanced properties dialog is normally displayed on the right of the work form, although it can be docked in any position. In Excel it is opened by right clicking on an object and selecting the Properties option from the popup menu. The dialog allows you to scan through all the available options. Some options require you to enter specific information, others provide a choice of entries from a drop down menu.

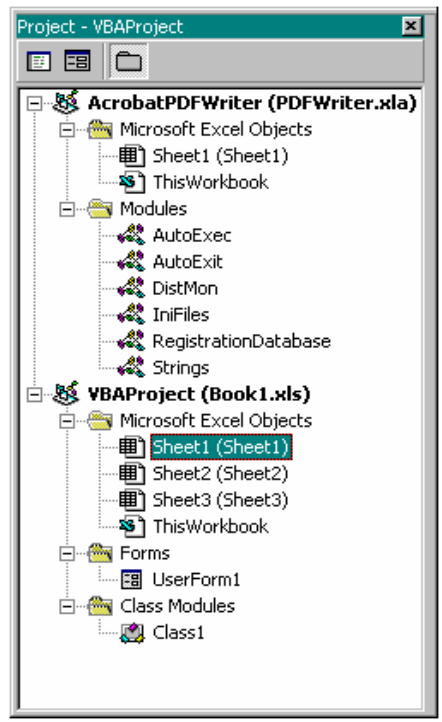
From the drop down menu at the top of the dialog select the object to be edited. This will display the full range of options available for that object, which can then be viewed either Alphabetically or Categorised. A full list of the options and their settings and ranges for the CX-Server Lite objects can be found in Appendix A – Component Properties.



Project Tree

Like Explorer the Project Tree provides a graphical representation of you application. In Visual Basic it is displayed to the right of the work form while in Excel it is shown on the left of the code sheet.

By expanding the tree you can see all your associated files and work sheets. It is possible to open any number of work sheets by simply double- clicking on them. Having multiple work sheets open in this way enables you to copy and paste between them saving you valuable time rewriting sections of code that already exist, and more importantly are known to work.



Controlling ActiveX Objects

A number of objects can be grouped together such as the 7 segment display and the spin buttons by selecting the objects you want to group while holding down the 'Shift' key as you select each object. When you have selected all the objects, right click on an object and select the 'Group' option from the drop down menu. Note however that objects must be ungrouped before their parameters can be edited.

Other drawing commands such as Bring to Front, Send to Back, Cut, Copy, Past etc. follow the standard windows conventions and are selected from the toolbar and/or drop down menus.

Appendix A
Component Properties

This appendix gives a list of the available properties. Each component supports a selection of these properties which can be set in design mode by using the properties dialog, or in the run time by using a Visual Basic script command – for example: - `Object1.Value = 10`

(Note: The container application may also display additional container-specific properties that apply to all objects, e.g. Excel provides a parent property. Consult the help for the container application for details of these properties.)

Property Title	Example Values	Description
About	None	Description of the object.
(Custom)	None	Opens the properties dialog for the object.
(Name)	Object Name	This is the system generated name for the object.
Autoload	True False	Switches the Autoload function On or Off. When set to On the ActiveX component value will autoload. True = On, False = Off.
AutoSize	0 to 1	Switches the fonts auto size option on or off. 0 = Off, 1 = On.
BackColour	&H00E0E0E0&	The code determines the background colour of the object. Click the browse button to display the colour palette.
BorderStyle	0 to 3	The value determines the visual appearance of the border that surrounds the object. 0 = None, 1 = Raised, 2 = Sunken, 3 = Single_Line
ButtonStyle	0 to 6	The value determines the button style. 0 = Toggle Switch, 1 = Colour Button, 2 = In/Out Button, 3 – Rotary Switch 4 = Rocker Switch, 5 = Indicator Button, 6 = Blank Button.
CommsServerName	Comms1 to n	This is the name of associated communications control.
ControlName	CheckBox1	Selects the control name that will be linked to the PLC item at runtime
DatasetName	MyLogFile	Determines the name of the dataset and the file to which the log data will be saved
DecimalPlaces	2	Number of decimal places for 7 Segment. Only applies when NumberBase = 10
DisplayErrors	True False	Determines whether or not error messages will be displayed by the communications control in a message box (set to True), or returned as an error code to the client application (set to False), in which case the Visual Basic Error Object should be used to display the error.
DisplayFont	Arial	This is the system name of the font used for the display. Use the browse button to display the font dialog. Arial is the default.
DisplayFontColour	&H00000000&	The code determines the colour of the display font. Click the browse button to display the colour palette.
DisplayFormat	0 – Dec 1 – Hex 2 – Scientific	Determines the format used to display the analogue display information. 0 = Decimal, 1 = Hexadecimal, 2 = Scientific.
DisplayMajorTicks	False True	Shows or hides the major tick marks of the scale. True = Show, False = Hide.
DisplayMajorUnits	False True	Shows or hides the major units markers of the scale. True = Show, False = Hide.
DisplayMinorTicks	False True	Shows or hides the minor tick marks of the scale. True = Show, False = Hide.
DisplayMinorUnits	False	Shows or hides the minor units markers of the scale.

	True	True = Show, False = Hide.
DisplayType	0 – Analogue 1 – Digital 2 – Text	Determines the type of display. 0 – Analogue: as a numeric value, 1 – Digital: as textual value, 2 – Text: as text.
Enabled	True False	Switches the object on or off. True = On, False = Off.
Font	Arial	This is the system name of the font used for the object title. Use the browse button to display the font dialog. Arial is the default.
GroupOrPLCName	MyPLCName	The name of the group or PLC to which the control is linked at runtime.
Height	141	Sets the overall height of the object in pixels.
IndicatorColour	&H00FF0000&	The code determines the colour of the indicator. Click the browse button to display the colour palette.
ItemName	Output Flow No1	This is the reference name given to the object.
KnobBorderPercentage	25	This value determines the size of the knob border as a percentage of the overall size.
KnobColour	&H02C4723E&	The code determines the colour of the knob. Click the browse button to display the colour palette.
KnobStyle	0 – Integer 1 – Real	Determines how the numerical value of the knob is interpreted 0 = as an integer or 1 = as a real number.
LeadingZeros	False True	Turns On or Off the leading zero's of the display. False = On, True = Off.
LEDColour	&H00FF0000&	The code determines the default or inactivated colour of the LED indicator. Click the browse button to display the colour palette.
Left	153	Determines the position of the object from the left hand edge of the work area in pixels.
Locked	True False	Locks the actions of the ActiveX object. True = Locked, False = Unlocked.
MajorTickInterval	10	Sets the number of major tick intervals.
MaxAngle	180	Determines the length of the arc for rotary gauges. Range $\pm 360^0$.
MaxCharacters	6	Determines the number of digits to be included in the display.
MaxDecimalPlaces	2	Determines the position of the decimal point in the display.
MaxGaugeValue	50	Sets the maximum value of the gauge.
MaxValue	999.99	Sets the maximum value that can be shown by a display.
MinAngle	0	Determines the length of the arc for rotary gauges. Range $\pm 360^0$.
MinGaugeAngle	50	Sets the minimum angle of the gauge.
MinGaugeValue	10	Sets the minimum value displayed on the gauge.
MinorTickInterval	2	Sets the number of tick intervals between each major tick interval.
MinValue	-999.99	Sets the minimum value that can be shown by a display.
NumberBase	10	Number Base for 7 Segment e.g. 2 for Binary, 10 for Decimal, 16 for Hexadecimal.
NumberOfDigits	3	Determines the number of digits to be included in the display.
Orientation	0 or 1	Sets the orientation of the gauge – Horizontal or Vertical. 0 = Vertical, 1 = Horizontal.
Path	C:\TEMP	Determines the location in which the control will save the log data
PLCName	PLC1	This is the name given to the PLC, not its type.
PrintObject	True False	Determines if this ActiveX object will be shown on a printout of the work sheet. True = Printed, False = Not printed.
ProjectName	C:\Test1.CDM	CX-Server Project Name
PropertyName	Value	The name of the property in the control to which the PLC item is linked at runtime
Rollovertime	1	Determines the frequency in hours at which the control will create a new log

		file
Round	0 or 1	Determines the shape of the LED indicator. 0 = Square, 1 = Round.
ScaleFont	Arial	This is the system name of the font used for the scale. Use the browse button to display the font dialog. Arial is the default.
ScaleFontColour	&H00000040&	The code determines the colour of the scale font. Click the browse button to display the colour palette.
SegmentColour	&H0000C000&	The code determines the colour of the display segments. Click the browse button to display the colour palette.
State0Colour	&H000000FF&	The code determines the colour of the toggle switch in the 0 state. Click the browse button to display the colour palette.
State0Text	Text	This is the text indicating the 0 state of the switch, i.e. Off, Stopped, Halt etc.
State1Colour	&H0000FF00&	The code determines the colour of the toggle switch in the 1 state. Click the browse button to display the colour palette.
State1Text	Text	This is the text indicating the 1 state of the switch, i.e. On, Running, Start etc.
StateFont	Arial	This is the system name of the font used to display the sate of the switch ro button. Use the browse button to display the font dialog. Arial is the default.
StateFontColour	&H00000000&	The code determines the colour of the title font. Click the browse button to display the colour palette.
Title	Text	This is the title that appears in the object.
TitlePosition	0 – Top 1 – Bottom	Determines where title will be placed in the object. Above or below the scale.
Top	230	Determines the position of the object from the top of the work area in pixels.
UnitPostion	0 – Inside 1 – Outside	Positions the dial units inside or outside the scale.
UpdateMethodID	-610	The ID of the method that will invoke the update to the PLC at runtime.
UpdateRate	10	The value sets the frequency, in seconds, at which the data passing to or from the object is updated.
Value	0	Determines the default value of the ActiveX object. This is typically (although not always) the default property for a control, so ControlName. = 10 is usually the same as ControlName.Value = 10
Visible	True False	Switches the object between visible and invisible in run mode. True = Visible, False = Invisible.
Width	98.25	Determines the overall width of the object in pixels.

Appendix B
Script / VB / C++ Interface

The Script Interface defines the Visual Basic script interface for the CX-Server communications control. (Note: The container application may also display additional container-specific methods that apply to all objects, e.g. Excel provides a BringToFront method. Consult the help for the container application for details of these methods.)

The same methods can also be used in C++ (although it is important to be aware of the fact that get and set are added before property names, so that, e.g., the D memory area property becomes SetD or GetD). Although this appendix primarily deals with script languages, and assumes that any C++ users are experienced and proficient, it does include examples in C++ where these should prove most helpful. The examples for ReadArea and WriteArea demonstrate supplying data as arrays of variants, and converting them back to other types. For more details of operation in Microsoft Visual C++ please see the separate CX-Server Lite Communications Control C++ Tutorial.

Important: For a guide to operation in Microsoft Visual Studio 2008, including use of these ActiveX controls, please see the "Guide To Using CX-Server Lite in Microsoft .Net". Note in particular that properties names in C# and VB.Net are prefixed with get_ and set_.

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.
Read	Function to read the value of a PLC point
Write	Function to write the value of a PLC point
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.
ReadArea	Function for reading a block of memory from the PLC.
WriteArea	Function for writing a block of memory to the PLC.
GetData	Function for starting OnData Events.
StopData	Function for stopping OnData Events.
OnData	Event for receiving notification of a change in data.

RunMode	Function for reading or writing the current mode of the PLC.
TypeName	Function for reading the PLC type (e.g. CQM1H).
IsPointValid	Checks whether a specified point (or address) exists and is valid.
IsBadQuality	Checks whether a point is currently indicating “bad quality”.
ListPLCs	Returns a list of the PLCs in a project.
ListPoints	Returns the list of points in a project (or in a PLC).
ClockRead	Reads the PLC Clock
ClockWrite	Sets the PLC Clock
RawFINS	Function that enables raw FINS commands to be sent to a specified PLC.
DisplayErrors	Function for control whether to display error in a message box.
ProjectName	Function for getting and setting the Name of the Project
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
PLC Memory Functions	A, AR, C, CIO, D, DM, DR, E, EM, G, H, HR, IR, LR, SR, ST, T, TC, TK, W. Functions for getting and setting the memory areas in the PLC.
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
DownloadProgram	Downloads a program to a PLC
UploadProgram	Uploads a program from a PLC
Protect	Protects (or releases protection on) program memory
InitCXServer	Initialises CX-Server (for advanced usage only)
NumErrors	Count of all errors occurred since Lite control was first run
LastErrorString	Description of last error that occurred
About	Brings up the about box
Help	Brings up help information

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.

Excel example 1 – Reading a value from the PLC using a logical name.

intVal = Comms1.Value("BoilerTemp")
or
intVal = Comms1("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 for a CX-Server communications control and does not have to be specified.

Excel example 2 – Writing a value to the PLC using a logical name.

Comms1.Value("BoilerTemp") = 50
or
Comms1("BoilerTemp") = 50

In these examples, the value 50 will be written to the PLC address associated with 'BoilerTemp'. "Value" is the default property for a CX-Server communications control and does not have to be specified.

Further Excel examples

Comms1("PLCName/DMO") = 1000
Comms1("DMO") = 1000
Intval = Comms1("DM100")

These examples are physical addresses. A PLC name can optionally be specified otherwise the current default PLC will be assumed.

VBScript (CX-Supervisor) example – Reading a value using a logical name.

intVal = Comms1.Value("BoilerTemp")

C++ example 1 – reads a value and converts it to text

```
COleVariant vaTemp;
CString csTextData;
COleVariant vaData;
vaData = m_CommsCtrl.GetValue(("BoilerTemp");
vaTemp.ChangeType(VT_BSTR, &vaData);
csTextData = vaTemp.bstrVal;
```

C+ example 2 – converts a text value to integer and writes it

```
CString csValue;
…
 // set csValue here to number input by user in string form, e.g. "1.2"
…
COleVariant var(csValue);
if(csValue.Find(".") != -1)
```

```
        var.ChangeType(VT_R8); // real number
else if(csValue == "TRUE")
{
        var.vt = VT_BOOL;
        var = (short)1;
}
else if (csValue == "FALSE")
{
        var.vt = VT_BOOL;
        var = (short)0;
}
else
        var.ChangeType(VT_I4); // integer
m_CommsCtrl.SetValue(PointName1, var);
```

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.

Note: in some programming languages properties are prefixed with Get or get_ (when reading the property) and Set or set_ (when writing the property) – e.g. Values would become get_Values or set_Values in C#.

Example 1 – Reading an array of values from the PLC using a logical name.

SomeArray = Comms1.Values("BoilerTemps")

Example 2 – Writing an array of values to the PLC using a logical name.

Comms1.Values("BoilerTemps") = SomeArray

C++ Example – see ReadArea for an illustration of how to handle values returned in a variant array.

Read

Reads the value of a PLC point.

Example of synchronous Read

IntVal = Comms1.Read("MyPLC", "MyPoint", WaitUntilComplete)

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. This is identical to the operation of the 'Value' method.

Example of asynchronous Read

Comms1.Read("MyPLC", "MyPoint", NoWaiting)

In this example, the point 'MyPoint' will be read from the PLC 'MyPLC'. The script will continue to execute the next line immediately, and when the data is read, it generates an OnData event.

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.

Write

Write the value of a PLC point.

Example of synchronous write:

Comms1.Write("MyPLC", "MyPoint", NewValue, WaitUntilComplete)

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. This is identical to the operation of the 'Value' method.

Example of asynchronous write:

Comms1.Write("MyPLC", "MyPoint", NewValue, NoWaiting)

In this example, 'NewValue' will be written to the point 'MyPoint' in the PLC called 'MyPLC'. The script will continue to execute the next line immediately.

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

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,

Comms1.SetDefaultPLC("MyPLC")
intVal = Comms1.Value("BoilerTemp1")
Comms1.Value("BoilerTemp1") = 75
intVal = Comms1.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.

VBScript (CX-Supervisor) Example

Comms1.SetDefaultPLC "MyPLC"

OpenPLC

Opens a PLC for communications. If no PLC is specified then the default PLC is opened. While it is generally not strictly necessary to open a PLC, because any attempt to communicate with it will cause it to automatically be opened, it is often considerably more efficient to do so (otherwise the PLC may be repeatedly automatically opened and closed each time any communications occurs, which can be very time consuming).

Example 1:

Comms1.SetDefaultPLC("MyPLC")
Comms1.OpenPLC()
Comms1.DM(100) = 10
Comms1.DM(50) = 10

Example 2:

Comms1.OpenPLC("MyPLC")
Comms1.DM(100) = 10

VBScript (CX-Supervisor) Example – Open the default PLC

Comms1.OpenPLC

Optional parameter: The name of the PLC being opened can be immediately followed by /force or /clearforce switch e.g.

*Comms1.OpenPLC("MyPLC**/force**")*
or *Comms1.OpenPLC("MyPLC**/clearforce**")*

If /force is used, then the PLC will not be closed until the ClosePLC method is called with the /force parameter (or the application exits). Note that the force status will be set even if the PLC was already open.

The /clearforce switch can be used to simply clear the forcing status, so that it is no longer forced open. If the PLC is currently not being forced then the /clearforce will have no effect.

If neither /force or /clearforce are used then any existing force status (or lack of force status) will remain unaffected.

ClosePLC

Closes a previously opened PLC. If no PLC is specified then the default PLC is closed.

Example:

Comms1.ClosePLC("MyPLC")

VBScript (CX-Supervisor) Example – Close the default PLC

Comms1.ClosePLC

Optional parameter: The name of the PLC being closed can be immediately followed by /force or /clearforce, e.g.

*Comms1.ClosePLC("MyPLC**/force**")*
*or Comms1.ClosePLC("MyPLC**/clearforce**")*

If /force is used, then the PLC will not be opened until the OpenPLC method is called with the /force parameter (or the application exits). Note that the force status will be set even if the PLC was already closed.

The /clearforce switch can be used to simply clear the forcing status, so that it is no longer forced closed. If the PLC is currently not being forced then the /clearforce will have no effect.

If neither /force or /clearforce are used then any existing force status (or lack of force status) will remain unaffected.

ReadArea

Reads a specified block of memory from a PLC.

Examples of synchronous read:

MyVariant = Comms1.ReadArea("MyPLC/DM0", 12, vbString, WaitUntilComplete)
MyVariant = Comms1.ReadArea("BoilerTemp", 10, vbInteger, WaitUntilComplete)
MyVariant = Comms1.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. In all these examples, WaitUntilComplete is specified (or assumed) so the script will wait for the read operation to complete before continuing to execute the next line.

Example of asynchronous read:

MyVariant = Comms1.ReadArea("BoilerTemp", 10, vbInteger, NoWaiting)

In this example, the Point 'BoilerTemp' will be read. The script will continue to execute the next line immediately, and when the data is read, it generates an OnData event.

VBScript (CX-Supervisor) Example – Synchronous read of an area using a logical name.

```
Dim data
data = Comms1.Read Area("myPointarray", 2, vbString + vbArray, 0)
'Access data by indexing the array.
sValue = data[0] ' For the first element of the array.
```

C++ Example:

This example converts an array of 10 integers based at D3000 to a string containing a comma-separated list of values.

```
COleVariant vaData;
COleVariant vaTemp;
CString strTextData;

CString csArea = "D3000"; // GetPointSelected();
vaData = m_CommsCtrl.GetReadArea(csArea,10,VT_INT,0);

// quick test for ints
SAFEARRAY* pArray = vaData.parray;
LONG lBound = 0, uBound = 0;
SafeArrayGetLBound( pArray, 1, &lBound );
SafeArrayGetUBound( pArray, 1, &uBound );
UINT elementsize = SafeArrayGetElemsize( pArray);

for (long i = lBound; i <= uBound; i++)
{
        HRESULT hr = SafeArrayGetElement(pArray, &i, &vaTemp);
        // This is just an example of converting array of integers to
        // comma-separated text
        vaTemp.ChangeType(VT_BSTR, &vaTemp);
        strTextData += vaTemp.bstrVal;
        if (i < uBound)
                strTextData += ",";
}
```

WriteArea

Writes a block of memory to a specified area in a PLC.

Examples of synchronous write:

```
MyString = "TestString"
Comms1.WriteArea "MyPLC/DM50", 10, MyString, WaitUntilComplete
Dim newValue(2) As Long
```

```
newValue(1) = 0
newValue(2) = 1
Comms1.WriteArea "BoilerTemp",2,newValue, WaitUntilComplete
```

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. In all these examples, WaitUntilComplete is specified (or assumed) so the script will wait for the write operation to complete before continuing to execute the next line.

Example of asynchronous write:

```
Comms1.WriteArea "Boiler Temp", 10, newValue, NoWaiting
```

In this example, the contents of 'newValue' will be written to the Point 'BoilerTemp'. The script will continue to execute the next line immediately, and the data write operation will continue in the background.

C++ Example

The following example writes the integers 101 to 110 to data locations from D3000 to D3009:

```
COleVariant vaData;
HRESULT hr = E_FAIL;
SAFEARRAYBOUND bound;
bound.lLbound = 1;
bound.cElements = 10;
SAFEARRAY* pArray = SafeArrayCreate(VT_I2, 1, &bound);
if (pArray)
{
        LONG vbIndex = 1;
        for(int i = 101; i <= 110; i++)
        {
                SafeArrayPutElement(pArray, &vbIndex, &i);
                vbIndex++;
        }
        vaData.vt = VT_I2 | VT_ARRAY;
        vaData.parray = pArray;
}
CString csArea = "D3000";
m_CommsCtrl.WriteArea(csArea,10,vaData,1);
```

GetData

Starts asynchronous data reading of the specified point at the requested update rate.

Example

Comms1.GetData "MyPLC", "MyPoint", nUpdateRate

In this example, MyPoint in MyPLC would be read at the rate of nUpdateRate (seconds). Data is then sent to the OnData routine.

An individual element of a point, defined as an array in a CDM file, can be accessed for reading or writing values.

Example

Comms1.GetData "MyPLC", "MyPoint[2], nUpdateRate

In this example, the third element of the point MyPoint (defined as plc1/DM500/10/USH) can now be accessed as MyPoint[2]. By default, if the array element is not specified, then the whole point (whole array of elements) is manipulated.

By using the 'OnChange' command as the fourth parameter, the GetData method will only return points when they change.

Example

Comms1.GetData "MyPLC", "MyPoint", nUpdateRate, OnChange

OnChange is of type *'UpdateSetting'*. By default, the UpdateSetting is set to *Continuous*, to remain compatible with previous versions.

VBScript (CX-Supervisor) Example – Get a single point

Comms1.GetData "MyPLC", "MyPoint", 1

StopData

Stops asynchronous data reading of the specified point.

Example

Comms1.StopData "MyPLC", "MyPoint"

In this example, the reading of MyPoint in MyPLC would be stopped.

VBScript (CX-Supervisor) Example

Comms1.StopData "MyPLC", "MyPoint"

OnData

This event is sent back to the container (e.g. Excel) when GetData has been called and new data is available.

Example

```
Private Sub Comms1_OnData(ByVal PLC As String, ByVal Point As String,
                          ByVal Value As Variant, ByVal BadQuality as Boolean)
    TextBox1 = Point
```

```
        Segment1 = Value
End Sub
```

In this example, the CX-Server 7 Segment component is set to the value of the point and a text box is set to display the current point.

If BadQuality is set to True then the value may be inaccurate e.g. from a device which has been disconnected.

The OnData routine can be enhanced to include logical expressions on the incoming Point name and then update the correct graphical object etc. for example:

```
Private Sub Comms1_OnData(ByVal PLC As String, ByVal Point As String,
                          ByVal Value As Variant, ByVal BadQuality as Boolean)
    If Point = "MyPoint" then
        Segment1 = Value
    Else if Point = "MyOtherPoint" then
        Cells(1,1) = Value
    End if
End Sub
```

VBScript (CX-Supervisor) Example

```
'In the OnData event of the CX-Lite control.

sPLC = PLC

sPoint = Point

sValue = Value
```

RunMode

Reads or writes 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.

Note: in some programming languages properties are prefixed with Get or get_ (when reading the property) and Set or set_ (when writing the property – e.g. RunMode will become get_RunMode or set_RunMode in C#.

Example 1

```
intMode = Comms1.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.

Example 2

```
Comms1.RunMode("MyPLC") = intMode
```

In this example the run mode is set to intMode

VBScript (CX-Supervisor) Example – Get the current PLC mode

Dim nMode

nMode = Comms1.RunMode("MyPLC")

TypeName

Returns from the CX-Server project file a PLC model name (e.g. C200H, CQM1H, CVM1 etc).

Example

strPLCType = Comms1.TypeName("MyPLC")

In this example, the PLC model type of 'MyPLC' will be read from the .CDM file and stored in 'strPLCType'.

VBScript (CX-Supervisor) Example

Dim sPLCType

sPLCType = Comms1.TypeName("MyPLC")

IsPointValid

Checks if a point (or address) exists and is valid

Examples

bIsValid = Comms1.IsPointValid("MyPoint", "MyPLC")

bIsValid = Comms1.IsPointValid("MyPoint")

The boolean variable bIsValid is set True if the point (or address) "MyPoint" exists or is a valid physical address. Note: Be careful when using this, the PLC parameter is optional and is therefore second, not first as is the case with with many other methods.

VBScript (CX-Supervisor) Example

bIsValid = Comms1.IsPointValid("MyPoint")

IsBadQuality

Checks whether a point is currently indicating "Bad Quality".

Example

bBad = Comms1.IsBadQuality("MyPLC", "MyPoint")

The boolean variable bBad is set True if the point "MyPoint" is indicating "Bad Quality" (e.g. the associated PLC is disconnected). It will also return True if the current quality is unknown - see note below.

VBScript (CX-Supervisor) Example

```
Dim bBad
bBad = Comms1.IsBadQuality("MyPLC", "MyPoint")
```

Note: In order to return a valid value, the point must have previously been read.

ListPLCs

Returns a list of PLCs in a project

Example

```
A = Comms1.ListPLCs
For Count = 1 To UBound(A)
   ComboBox1.AddItem (A(Count))
Next Count
```

This example fills a combo box with the names of PLCs in the current project.

VBScript (CX-Supervisor) Example

```
Dim arrayOfPLCs
Dim nUbound, nLbound
arrayOfPLCs = Comms1.ListPLCs
nLbound = LBound(arrayOfPLCs)
nUbound = UBound(arrayOfPLCs)
For Count = nLbound To nUbound
        'Do something with list here
Next
```

ListPoints

Returns a list of points in a project or PLC

Example 1

```
A = Comms1.ListPoints(ComboBox1.Text)
For Count = 1 To UBound(A)
   ComboBox2.AddItem (A(Count))
Next Count
```

This example fills ComboBox2 with the names of points in the PLC shown in ComboBox1.

Example 2

```
A = Comms1.ListPoints
For Count = 1 To UBound(A)
   ComboBox3.AddItem (A(Count))
Next Count
```

This example fills ComboBox3 with the names of points in the current project.

VBScript (CX-Supervisor) Example

```
Dim arrayOfPoints

Dim nUbound, nLbound

arrayOfPoints = Comms1.ListPoints(sPLC)

nLbound = LBound(arrayOfPoints)

nUbound = UBound(arrayOfPoints)

For Count = 1 To UBound(arrayOfPoints)

        'Do something with points list here

Next
```

ClockRead

Function that reads the PLC clock

VBA (e.g. Excel) Example

```
Dim NewDate as Date
NewDate  = Comms1.ClockRead("PLC1")
Text1 = NewDate     ' this is just an example of displaying the date in a text box
```

VBScript (CX-Supervisor) Example

```
Dim NewDate
NewDate  = Comms1.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
```

ClockWrite

Function that sets the PLC clock. The expected format for the date is “dd/mm/yyyy hh:mm:ss”.

VBA (e.g. Excel) Example

```
Dim NewDate as Date
```

```
'set time/date value here using standard VBA methods
NewDate = Text1  ' This example is getting the date from a text box, set earlier by ClockRead
Comms1.ClockWrite "PLC1",  NewDate
```

VBScript (CX-Supervisor) 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
Comms1.ClockWrite "PLC1",  NewDate

Comms1.ClockWrite “PLC1”, Now  ‘does the same as the example above using less script
```

RawFINS [Advanced function]

This function is for advanced users familiar with the Omron FINS protocol only. This function enables raw FINS commands to be sent to a specified PLC

Example

```
Dim sFINSResponse as String
sFINSResponse = Comms1.RawFINS(“0501”, “MyPLC”, AsString, PLC_CPU, False)
```

In this example, the FINS message “0501” (model read) would be sent to “MyPLC” as a string. The response from the PLC will be stored as a string in sFINSResponse.

The final two parameters are optional. The first of them, UnitNumber, allows the command to be sent to the PLC CPU or to a bus unit (e.g. BUS_UNIT1, the default is PLC_CPU), the second parameter specifies whether errors should be ignored (the default is False)

Example

```
Dim bFINSResponse() as Byte
Dim bFINSCommand(2) as Byte
bFINSCommand(0) = &H7
bFINSCommand(1) = &H1
bFINSResponse = Comms1.RawFINS(bFINSCommand, “MyPLC”, AsArrayOfBytes)
Count = 0
For Count = LBound(bFINSResponse) to UBound(bFINSResponse)
        MsgBox(bFINSResponse(Count))
Next Count
```

In this example, the FINS message “0701” (to read the PLC’s clock) would be sent to “MyPLC” as an array of bytes. The response from the PLC will be stored as an array of bytes in bFINSResponse.

By using this function it is possible to send any supported FINS command to a specified PLC. For further information regarding FINS commands please see the FINS commands reference manual.

VBScript (CX-Supervisor) Example

```
Dim sFINS
Dim sResponse


sFINS = "0501"
sResponse = Comms1.RawFINS(sFins, sPLC)
txtFINSResponse = sResponse 'txtFINSResponse is a CX-Supervisor point.
```

DisplayErrors

The DisplayErrors property can be used to control whether to display error in a message box. For example,

```
Comms1.DisplayErrors = True
```

Or

```
Dim IsDisplayingErrors as Boolean
IsDisplayingErrors = Comms1.DisplayErrors
```

ProjectName

The ProjectName property can be used to set the Name of the Project to a specific path. For example,

```
Comms1.ProjectName = C:\Program Files\OMRON\CX-Server Lite\LiteExample.cdm
```

Or to get the current one,

```
Dim ProjectPath as String
ProjectPath = Comms1.ProjectName
```

Active

Returns the connection status of a specified PLC.

Example

```
IntConnectedStatus = Comms1.Active("MyPLC")
```

In this example, the connected status would be read from 'MyPLC' and stored in 'intConnectedStatus'. If 'MyPLC' is connected 'intConnectedStatus' would be set to True.

VBScript (CX-Supervisor) Example

```
bActive = Comms1..Active(sPLC) ' bActive is a CX-Supervisor point
```

TCGetStatus

The .TCGetStatus command will return status data for the specified device.

Example

```
Dim bTCStatusResponse() As Byte
bTCStatusResponse = Comms1.TCGetStatus("E5AK")
'Heating output
Cells(1, 1) = bTCStatusResponse(21)
'Cooling output
Cells(2, 1) = bTCStatusResponse(22)
'Alarm 1 output
Cells(3, 1) = bTCStatusResponse(23)
'Alarm 2 output
Cells(4, 1) = bTCStatusResponse(24)
'Alarm 3 output
Cells(5, 1) = bTCStatusResponse(25)
'Stopped status
Cells(8, 1) = bTCStatusResponse(28)
'Remote status
Cells(10, 1) = 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.

By using this function it is possible to retrieve all supported status information from a specified temperature controller device.

See Appendix C for details regarding the supported device types and for details regarding which devices support which status bits.

TCRemoteLocal

The .TCRemoteLocal command will execute the Remote/Local command for the specified device:

Example - in this example, the “E5AK” device is being set to local mode:

```
'Set the device to local mode
Comms1.TCRemoteLocal "E5AK", 1
```

Example - in this example, the “E5AK” device is being set to remote mode:

'Set the device to remote mode
Comms1.TCRemoteLocal "E5AK", 0

PLC Memory Functions (A, AR, C, CIO, D, DM, DR, E, EM, G, H, HR, 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 = Comms1.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

Comms1.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

intVal = Comms1.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 'intVal' (e.g. 'value' will be set to TRUE or FALSE).

Example 4

Comms1.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

SetDeviceAddress [Advanced function]

This function is for advanced users only. 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". **Note: this method does not interpret 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.**

VBA (e.g. Excel) Example 1

NetworkNum = 1
NodeNum = 2
UnitNum = -1
iPAddress = “10.0.0.1”
bValid = Comms1.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.

SetDeviceConfig [Advanced function]

This function is for advanced users only. 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. **Note: this method does not interpret 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.**

Currently supported values are:

VBA (e.g. Excel) Example

Device = “PLC1”
Section = “ADDRESS”
Entry = “IPADDR”
Setting = “10.0.0.1”
bValid = Comms1.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.

CX-Supervisor script example:

CXServer.SetDeviceConfig "MyPLC", "ADDRESS", "IPADDR", "10.0.0.1"

This function is primarily intended for future use. 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.

GetDeviceConfig [Advanced function]

This function is for advanced users only. 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.

VBA (e.g. Excel) Example 1

Dim Setting As String

Device = “PLC1”
Section = “ADDRESS””
Entry = “IPADDR”
Setting = Comms1.GetDeviceConfig(Device, Section, Entry)

CX-Supervisor script example:

IPAddrPoint = CXServer.GetDeviceConfig("MyPLC", "ADDRESS", "IPADDR")

Currently supported parameter values are as described for the SetDeviceConfig method.

UploadProgram [Advanced function]

This function is for advanced users only. The UploadProgram function can be used to read a program (or other file) 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.

VBA (e.g. Excel) Example 1

Dim Source As String

Dim DestinationFile As String

Source = “”

DestinationFile = “c:\test1.bin”

Comms1.UploadProgram "PLC1", Source, DestinationFile, WaitUntilComplete, True

VBA (e.g. Excel) Example 2

BValid = Comms1.UploadProgram ("PLC1", "ALL", "c:\test1.bin", WaitUntilComplete, True)

Note: The parameters, in order, are the PLC name, the source (see below), the destination file, WaitUntilComplete (value 0) or NoWaiting (value 1), and whether to report on progress.

Possible values for the source parameter are as follows:

"" (i.e. an empty string) – will upload the program (object) file only

"PRG" – same as above, will upload the program (object) file only

"FBL" – will upload the Function Block Library file (The Function Blocks associated with a program)

"ALL" – will upload both the program file and the Function Block file. The Function Block file will be given the same name as the destination file except that a .FBL extension will be used. Note that currently use of this setting will force the wait setting to always be interpreted as "WaitUntilComplete" as the program and FB upload need to occur consecutively.

"CANCEL" – will cancel (abort) any current upload. The program is uploaded on a block-by-block basis, and the cancel status is checked after each block is uploaded, so it is possible that an upload will complete before the cancel takes effect. In that case the cancel command will be ignored.

Any other value – will be interpreted as the name of a file in the root directory of a memory card, e.g. "Example.obj"

The second example shows that this method returns a Boolean value, which is set to True if no errors are detected (but note that if the command is used with NoWaiting, then an error may occur after the method returns but before the operation is complete, and must be detected via the OnData routine - see below).

If progress reporting is set, then the progress updates will be provided via the OnData callback. The format of this is:

```
Private Sub Comms1_OnData(ByVal PLC As String, ByVal Point As String,
                          ByVal Value As Variant, ByVal BadQuality as Boolean)
End Sub
```

When used with program upload reporting, the parameters are used as follows:

PLC – Name of PLC

Point – "UPLOAD:START" or "UPLOAD:RECEIVED" or "UPLOAD:COMPLETE" or "UPLOAD:ERROR" or "UPLOAD:CANCELLED"

Value – For "UPLOAD:START" this is the total number of bytes to upload (i.e. file size)

For "UPLOAD:RECEIVED" and "UPLOAD:COMPLETE" and "UPLOAD:CANCELLED" this is the total bytes uploaded so far.

For "UPLOAD:ERROR" this is a CX-Server error code (e.g. 35339 which is 8A0B in hexadecimal, indicates the PLC is in the wrong mode)

DownloadProgram [Advanced function]

This function is for advanced users only. The DownloadProgram function can be used to write a program to a PLC. The PLC **must** be in program mode before DownloadProgram is used.

Care should be taken with this function to ensure that the program written is valid for the PLC to which it is downloaded. In addition, users should be aware that downloading the program object code on its own clears the associated function block area. If a program uses function blocks then the associated function block file *must* be downloaded to the PLC before the PLC program is run, or a PLC error condition will occur. Any attempt to download a function block to a PLC that does not support FBs will result in an error on download.

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.

VBA (e.g. Excel) Example 1

```
Dim SourceFile As String
Dim DestinationFile As String
SourceFile = "c:\test2.bin"
Destination = “ALL”
Comms1.DownloadProgram "PLC1", SourceFile, Destination, WaitUntilComplete, True
```

VBA (e.g. Excel) Example 2

```
bValid =Comms1.DownloadProgram("PLC1", "c:\test2.bin", "", WaitUntilComplete, True)
```

Note: The parameters, in order, are the PLC name, the source file, the destination (see below), WaitUntilComplete (value 0) or NoWaiting (value 1), and whether to report on progress.

Possible values for the destination parameter are as follows:

“” (i.e. an empty string) – will download the source file as a PLC program (object file) only. This clears the function block area - if the program references any function blocks then these *must* be downloaded (using the FBL parameter) before the PLC is run, or a PLC error condition will occur.

“PRG” – same as above, will download the source file as a PLC program (object file) only. This clears the function block area - if the program references any function blocks then these *must* be downloaded (using the FBL parameter) before the PLC is run, or a PLC error condition will occur.

“FBL” – will download the source file as a Function Block Library file (The Function Blocks associated with a program). Note: The PLC must support function blocks, and the downloaded function blocks must match the PLC program or an error will occur when downloading.

"ALL" – will download the program file and then any associated Function Block file. The Function Block file will be downloaded from a file with the same name as the source file except that a .FBL extension will be used. If the FBL file does not exist then only the program file will be downloaded. If a suitably named FBL file is found, then the PLC must support function blocks, and the downloaded function blocks must match the PLC program or an error will occur when downloading. Note that currently use of this setting will force the wait setting to always be interpreted as "WaitUntilComplete" as the program and FB download need to occur consecutively.

"CANCEL" – will cancel (abort) any current download. Care should be taken to ensure that a valid program is downloaded after using this command, otherwise the contents of the PLC are likely to be invalid. The program is downloaded on a block-by-block basis, and the cancel status is checked after each block is downloaded, so it is possible that a download will complete before the cancel takes effect. In that case the cancel command will be ignored.

Any other value – will be interpreted as the name of a file, e.g. "Example.obj", in which case a file of that name will be created or overwritten in the root directory of a memory card

The second example shows that this method returns a Boolean value, which is set to True if no errors are detected (but note that if the command is used with NoWaiting, then an error may occur after the method returns but before the operation is complete, and must be detected via the OnData routine - see below).

If progress reporting is set, then the progress updates will be provided via the OnData callback. The format of this is:

Private Sub Comms1_OnData(ByVal PLC As String, ByVal Point As String,
ByVal Value As Variant, ByVal BadQuality as Boolean)
End Sub

When used with program download reporting, the parameters are used as follows:

PLC – Name of PLC

Point – "DOWNLOAD:START" or "DOWNLOAD:SENT" or "DOWNLOAD:COMPLETE" or "DOWNLOAD:ERROR" or "DOWNLOAD:CANCELLED"

Value – For "DOWNLOAD:START" this is the total number of bytes to upload (i.e. file size)

For "DOWNLOAD:SENT" and "DOWNLOAD:COMPLETE" and "DOWNLOAD:CANCELLED" this is the total bytes downloaded so far.

For "DOWNLOAD:ERROR" this is a CX-Server error code (e.g. 35339 which is 8A0B in hexadecimal, indicates the PLC is in the wrong mode)

Protect [Advanced function]

This function is for advanced users only. 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.

VBA (e.g. Excel) Example 1 (sets protection for CS series PLC)

```
Dim SetProtection as Boolean
Dim PasswordString As String
Dim PasswordNumber As Long
EnableProtection = true
PasswordString = “Password”
PasswordNumber = 0
Comms1.Protect “PLC1”, EnableProtection, PasswordString, PasswordNumber
```

VBA (e.g. Excel) Example 2 (unsets protection for C series PLC)

```
Dim SetProtection as Boolean
Dim PasswordString As String
Dim PasswordNumber As Long
EnableProtection = false
PasswordString = “”
PasswordNumber = 12345678
Comms1.Protect “PLC1”, EnableProtection, PasswordString, PasswordNumber
```

Note: 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.

InitCXServer [Advanced function]

This function is for advanced users only. The InitCXServer function is **not** normally required. It is intended for use only in certain specialised situations, e.g. it may be of use in a multithreading environment, to initialise worker threads before the first CX-Server function in that thread is called. It should not be used in the main thread, and, in general, it should only be used on advice from Omron. An appropriate Microsoft COM initialisation method, e.g. CoInitialize or CoInitializeEx, may also need to be called within the thread before this method is called.

C++ examples:

m_CommsCtrl.InitCXServer(true); // Initialises CX-Server within a thread

m_CommsCtrl.InitCXServer(false); // Uninitialises CX-Server within a thread

The unitialise example should only be used when no further CX-Server functionality is required within a thread.

NumErrors

This property, which can be set as well as read, is a count of the number of errors that have occurred since the Lite control was first run.

Excel Examples:

Cells(2,2) = Comms1.NumErrors

Comms1.NumErrors = 0;

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.

Excel Examples:

Cells(2,2) = Comms1.LastErrorString

Comms1.LastErrorString = “No more yet”

AboutBox

Brings up the About Box.

Example

Comms1.AboutBox

Help

Brings up the Help information.

Example

Comms1.Help

Appendix C
Temperature Controller Support

Hardware support

CX-Server LITE supports all temperature controller devices supported by CX-Server, these are:

- E5AF-A (*)
- E5AF-AH (*)
- E5EF-A (*)
- E5EF-AH (*)
- E5EF-BA (*)
- E5EF-BAH (*)
- E5AX-A (*)
- E5AX-AH (*)
- E5AX-D (*)
- E5AX-LA (*)
- E5AX-MA (*)
- E5AX-PRR (*)
- E5AX-V (*)
- E5AJ-A (*)
- E5EJ-A (*)
- E5AK-AA
- E5AK-PRR

(*) These devices are currently being phased out

Parameter support

The following table outlines the temperature controller points available from the CX-Server points editor. Cross-reference this information with the support offered by your specified device.

Data Location	Word Addressing	Bit Addressing	Read/Write
SETPOINT	√	X	√
PROPORTIONAL	√	X	√
INTEGRAL	√	X	√
DERIVATIVE	√	X	√
OUTPUT	√	X	X
LOWER_LIMIT	√	X	X
UPPER_LIMIT	√	X	X
PROCESS	√	X	X
STRENGTH	√	X	√
SCALE1	√	X	√
SCALE2	√	X	√
ALARM	√	√	√
SHIFT	√	√	√
SHIFT_LOWER	√	√	√
SHIFT_UPPER	√	√	√
BURNOUT	√	√	√
HEATER	√	X	X
ALARM1	√	√	√
ALARM2	√	√	√
ALARM3	√	√	√
COOLING	--	--	--
DEADBAND	--	--	--
VALVE	--	--	--

To access these values via the CX-Lite component add the appropriate temperature controller device. Add a point using the value from the Data Location column in the table above. You can then access this information as you would with any point.

Device support

The following table outlines the temperature controller points available for each of the devices supported by CX-Server.

	E5AF-A	E5AF-AH	E5EF-A	E5EF-AH	E5EF-BA	E5EF-BAH	E5AX-A	E5Ax-AH	E5AX-D	E5AX-LA	E5AX-MA	E5AX-PRR	E5AX-V	E5AJ-A	E5EJ-A	E5AK-AA	E5AK-PRR
SETPOINT	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
PROPORTIONAL	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
INTEGRAL	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
DERIVATIVE	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
OUTPUT	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
LOWER_LIMIT	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
UPPER_LIMIT	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
PROCESS	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
STRENGTH	√	√	√	√	√	√	--	--	--	--	--	--	--	√	√	--	--
SCALE1	√	√	√	√	√	√	--	--	--	--	--	--	--	√	√	--	--
SCALE2	√	√	√	√	√	√	--	--	--	--	--	--	--	√	√	--	--
ALARM	√	√	√	√	√	√	√	√	√	√	√	√	√	--	--	--	--
SHIFT	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	--	--
SHIFT_LOWER	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√
SHIFT_UPPER	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√
BURNOUT	--	√	--	√	--	√	--	√	--	--	--	--	--	√	√	√	√
HEATER	--	√	--	√	--	√	--	√	--	--	--	--	--	√	√	√	√
ALARM1	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√	√	√
ALARM2	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√	√	√
ALARM3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√
COOLING	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√
DEADBAND	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√
VALVE	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	√	√

In addition to the previous table, a special value of PARMxx can be used to refer to any E5AK parameter, where xx is replaced by a number that corresponds to the number of the parameter given in the E5AK manual. For example, the special address PARM45 would refer to the 'SP ramp set value'.

Refer to the Temperature Controller manual for descriptions of each special address.

TCGetStatus Information

The following status information is provided by the TCGetStatus() command:

1. eControlMode (* Note 1)	28. bHBAOutput (* Note 3)
2. eOutput (* Note 1)	29. bStopped (* Note 3)
3. fInputShiftDelay (* Note 1)	30. bManual (* Note 3)
4. eDisplayUnit (* Note 1)	31. bRemote (* Note 3)
5. fPIDConstantDisplay (* Note 1)	32. bRSPMode (* Note 3)
6. eOutputType (* Note 1)	33. bAT (* Note 3)
7. eCoolingType (* Note 2)	34. bEmpty1 (* Note 3)
8. eOutput2 (* Note 2)	35. bEmpty2 (* Note 3)
9. eAlarm1 (* Note 1)	36. bEmpty3 (* Note 3)
10. eAlarm2 (* Note 4)	37. bEmpty4 (* Note 3)
11. nInputType (* Note 1)	38. nSettingLevel (* Note 3)
12. eOperationMode (* Note 1)	39. bRAMWriteMode (* Note 3)
13. eBackupMode (* Note 1)	40. nControlOutput1Type (* Note 3)
14. eAutoTuneMode (* Note 1)	41. nControlOutput2Type (* Note 3)
15. fOverFlow (* Note 1)	42. bEEPROM (* Note 3)
16. fUnderFlow (* Note 1)	43. bInputError (* Note 3)
17. fSensorMalfunction (* Note 1)	44. bADConvertorError (* Note 3)
18. fADConvertorFailure (* Note 1)	45. bCTOverflow (* Note 3)
19. fRAMAbnormality (* Note 1)	46. bCTHold (* Note 3)
20. fRAMMismatch (* Note 1)	47. bPotentiometerError (* Note 3)
21. fStatusWordsOnly (* Note 3)	48. bRSPInputError (* Note 3)
22. bHeatingOutput (* Note 3)	49. bEmpty5 (* Note 3)
23. bCoolingOutput (* Note 3)	50. bEmpty6 (* Note 3)
24. bAlarm1Output (* Note 3)	51. bEmpty7 (* Note 3)
25. bAlarm2Output (* Note 3)	52. bEmpty8 (* Note 3)
26. bAlarm3Output (* Note 3)	53. bEmpty9 (* Note 3)
27. bLBAOutput (* Note 3)	

(* Note 1) E5*F, E5*X, E5*J only

(* Note 2) E5AX-V, E5AX-D only

(* Note 3) E5*K, only

(* Note 4) E5*F, E5AX-A, E5AX-AH, E5AX-LA, E5AX-MA, E5AX-PRR, E5*J only

Appendix D
Microsoft Visual Studio .NET 2003

General Usage and Considerations

CX-Server LITE has been designed to work with Microsoft Visual C++ 6 IDE and Microsoft Visual Basic 6 IDE. CX-Server LITE continues to support the new IDE from Microsoft, Visual Studio .NET 2003, but issues can arise that may cause confusion.

Microsoft Visual Studio .NET 2003 moves away from separate IDEs for users and instead offers a single IDE for all. This means that Visual Basic and Visual C++, and the new C# programming languages are all supported from one single IDE.

Adding the CX-Server LITE control to a project is now identical for all programming languages.

1. In the toolbox windows right-click and select **Add/Remove Items…**.
2. In the **Customize Toolbox** dialog select the **COM Components** tab. (This can take some time.)
3. Sort by **Name** (the first column) and scroll down until you find the Omron CX series of components. These include the graphical objects – e.g. the Omron CX Knob control - and the communications components – the Omron CX Communications control and the Omron CX OPC Communications control.
4. Select any of the controls you will be interested in using by clicking in the box to the left of the component's name. A tick will appear.
5. When you select **OK** on the **Customize Toolbox** dialog, the dialog will close and the components will be added to the toolbox.

For Visual Basic and Visual C# users' usage will be largely the same as for Visual Basic. Drop a component onto the form and use the properties page and custom properties to tailor the component.

Visual C++ users will find it necessary to work in a slightly different manner. The standard **Properties** was not available in Visual Studio C++ 6. Right-clicking the component and asking for properties used to display the standard properties for selecting a CX-Server project. In Visual Studio .NET that command will default to the **Properties** window where you cannot do this. There are two methods to view the properties of the component:

1. Near the top of the Properties window is the **Property Pages** icon,  . Selecting this will provide the property pages familiar to Visual C++ developers.
2. Right-clicking the components will display a pop-up menu. Of the menu options available, there is one labelled **ActiveX – Properties**. Selecting this will provide the property pages familiar to Visual C++ developers.

Glossary of Terms

ActiveX	A component technology developed by Microsoft allowing components to communicate with **applications**.
Application	A software program that accomplishes a specific task. Examples of applications are **CX-Supervisor**, **CX-Programmer, CX-Server, Microsoft Word** and **Microsoft Excel**
Communications Driver	The relevant communications management system for OMRON PLCs in conjunction with **Microsoft Windows**, providing facilities for other CX Automation Suite software to maintain PLC device and address information and to communicate with OMRON PLCs and their supported network types.
Event	User action, e.g. mouse click or System action, e.g. timer tick which may cause a script to execute.
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.
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, a **PLC** or a target application.
Icon	Pictorial representations of computer resources and functions.
IDE	Integrated Development Environment.
Microsoft Excel	A spreadsheet **application**.
Microsoft Windows	The most common operating system used by Personal Computers. **CX-Server Lite** will run only under Microsoft Windows.
Microsoft Word	A word processing **application**.
OLE	Object Linking and Embedding. Used to transfer and share information between **Microsoft Windows** based **applications** and accessories.
PC	Abbreviation for Personal Computer.
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. 1024x768).
PLC	Abbreviation for Programmable Logic Controller.

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 **DDE**. The contents of a point may control the action or appearance of an object, or be used for output via an I/O mechanism.
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.
Windows Desktop	An integral part of **Microsoft Windows**that allows Microsoft Windows based **applications** to be started from **icons** and for all **applications** to be organised.

Index

T

V

W