# CX-Server Lite
# User Manual

# Guide to using CX-Server Lite in Microsoft .Net

# *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:

- ***Using CX-Server Lite in Microsoft .Net:*** This describes the use of the CX-Server Lite software within the .Net environment (e.g. Visual Studio 2008) in general terms
- ***Tutorial:*** This is a quick tutorial for use in Visual Studio host applications.
- ***Appendix A Design Mode Properties:*** This appendix summarises the component properties available within Visual Studio .Net.
- ***Appendix B Run Mode Interface:*** The Microsoft .Net interface for the CX-Server communications control.

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*

# 1. Using CX-Server Lite in Microsoft .Net

This book introduces the use of CX-Server Lite within a Microsoft .Net (e.g. Visual Studio 2008) environment to a new user. It is assumed that the reader is already familiar with CX-Server Lite in general terms, and is proficient at using Microsoft .Net technology.

**Important: See the "Getting Started" section of the main user manual for a general introduction to CX-Server Lite, for important system requirements and installation information, and for details of use in other, non .Net, applications.**

## 1.1 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), Visual Basic (6.0 and later), and Visual C++ 6.0, as well as Omron CX-Supervisor. From version 1.21 onwards CX-Server Lite also supported use in Windows applications created using the Microsoft Visual Studio .Net environment (use in an ASP.Net environment is not currently supported), using a development tool such as Microsoft Visual Studio 2008. In all of these applications, CX-Server Lite allows existing process data to be collected and analysed, via the CX-Server runtime system.

## 1.2 About this Manual

This manual helps a new user get started with CX-Server Lite in a Microsoft .Net environment, by leading the user through the basics of CX-Server Lite operation. 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.

Throughout this manual, it is assumed that the user has a working knowledge of Microsoft Windows and Microsoft .Net. development environments (e.g. Visual Studio 2008)

If Visual Studio .Net has not been used before, it is recommended that some time working with the Microsoft documentation is spent before using CX-Server Lite. Similarly, familiarity with the key Microsoft .Net concepts is also assumed.

## 1.3 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

There are several ways to access the help system from within the Visual Studio .Net development environment. One of the simplest is to right click on the object and then select the relevant help option from the popup menu. In the case of the CX-Server Lite ActiveX controls this is the “ActiveX –Help” option. The configuration dialogs for many of the controls also include a “Help” button, and “ActiveX-Help” can also be accessed from below the properties window.

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.

## 1.4 About CX-Server Lite

The CX-Server Lite ActiveX Components include an About dialog accessible from the object right-button menu (select the “ActiveX - About” option). 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.

## 1.5 Technical Support / Issues with old applications when upgrading

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

**If the problem occurred with an old application when upgrading to use a newer version of CX-Server Lite, please see Appendix C1 – Issues to be aware of when upgrading.**

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 of one of the ActiveX controls to obtain the version number of the application.

# 2. Using CX-Server Lite in Microsoft .Net Overview

## 2.1 CX-Server Lite Communications Control (ActiveX version)

CX-Server Lite includes a standard ActiveX communications control that acts as an interface to CX-Server within Excel, Visual Basic and C++ applications. This control can also be used in a .Net environment, using the standard .Net ActiveX interoperability support. All functionality is available, including automatic linking to the Omron Graphical Controls, meaning that it is ideal for many .Net Windows applications (e.g. Windows Form based applications).

The functionality of this control is described in the main CX-Server Lite user manual.

See section 3 for more details on using this control within Visual Studio .Net.

## 2.2 CX-Server Lite ActiveX Graphical Objects Overview)

CX-Server Lite includes a set of ActiveX Graphical Controls, which, like other ActiveX objects, can be used within a .Net environment. This section contains a brief overview of the available components. For full details of these objects see the main CX-Server Lite User Manual. For full details of ActiveX compatibility in a Microsoft .Net environment see the Microsoft documentation.

See section 3 for more details on using these controls within Visual Studio .Net.

**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 Run Mode 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. This control is not required within a .Net environment.

**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.

## 2.3 CX-Server Lite Communications Control (.Net version)

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

See section 4 for more details on using this control.

Note: This control is a .Net **Windows** Control intended for use within **Windows** applications. It is not a **Web** control, and is therefore not designed for use within a web server (e.g. IIS / ASP.NET) environment. If a web page is selected while the control is present on the Visual Studio toolbox, the control will appear greyed out.

# 3. Using the CX-Server Lite Active X controls in Microsoft .Net Applications

Visual Studio provides excellent support for “legacy” ActiveX objects. In many ways operation is similar to in previous development environments, such as Visual Basic 6.0.

As a result of this excellent support, it often makes sense to use the ActiveX controls even within a .Net environment, and even though a .Net Communications Control is available.

Users familiar with the ActiveX controls, or who just wish to construct simple Windows based applications that provide a graphical display of data, should consider using the CX-Server Lite Active X controls in preference to the CX-Server Lite .Net Communications Control, as they provide “quick and simple” automatic linking to the graphical controls.

In addition, some less commonly used functionality (e.g. interfacing to Temperature Controllers rather than PLCs) is only available in the CX-Server Lite ActiveX Communications Control.

To add one of the Omron ActiveX controls to the Visual Studio Toolbox do the following:

1. Click on the Toolbox (on the left side of the Visual Studio window)
2. Click on the Toolbox Components tab
3. Right-click on the background of the Toolbox window and select “Add / Remove Items” from the popup menu.
4. Scroll the list box down in the Com Components Tab, and select the components beginning with Omron CX (e.g. the Omron CX 7 Segment Control).
5. Select OK. The selected components will now be added to the Toolbox window.

To use one of the components

1. Drag from the Toolbox window and drop onto the form. Resize as desired.
2. To configure the component, right-click on the component and select Properties. This will bring up the properties dialog. The properties can then be configured using the properties dialog in the same way as in Excel or Visual Basic. By default the ActiveX component names will be prefixed by “ax”, e.g. “axKnob1”. To edit this select the (name) property to change the name, e.g. to “Knob1”.
3. Alternatively most properties of the selected control can be edited using the standard Visual Studio Property Editor Window. It is recommended, however, that editing of the project file name (i.e. selection of a project file) is done from the properties dialog.

To connect a graphical control to a communications control:

1. Add both controls to the form using the steps outlined above
2. Invoke the graphical control communications properties dialog, and configure the Data Source tab in the same way as for Excel and Visual Basic applications, i.e.

use the combo boxes to select the communications control, a device, and a point. If necessary use the ">" buttons to add new devices or points, or edit existing ones.

To drive the control from C# or Visual Basic .Net code:

1. Access the properties in the runtime in the usual way, e.g. the C# code to set a Knob control to the value 10 in C# is: Knob1.value = 10; **(Note: VB.NET will prefix ActiveX property names with either get_ or set_. As an example, ListPoints will become get_ListPoints)**
2. Access the events either by double-clicking on the control (e.g. for the ClickOn event), or, if using C#, by using the standard Visual Studio Event Editor (selected by clicking on the lightning icon in the Properties window). In VB.Net the events can be accessed by using the class name and method name drop-down list boxes which are displayed at the top of the code window (just below the tab for the VB source file).

# 4. CX-Server Lite Communications Control (.Net version)

The .Net version of the CX-Server Lite Communications Control is a lightweight object intended for use in any environment where .Net Windows Controls are supported. (Note: it is a **Windows** Control, not a **Web** Control, so is **not** intended for use in a web application.) Unlike the ActiveX version it does not include direct automatic support for interfacing to the graphical controls, although they can, of course, be driven directly from the application if desired (i.e. the value obtained from the .Net Communications Control and used to set the ActiveX graphical control, or vice-versa).

**Important: Please note that this version of the .Net control has been developed using, and is primarily intended for use in, the Visual Studio 2008 (and later) development environment.** The ActiveX controls (described earlier) can, however, still be used with earlier versions of the Visual Studio development environment.

It makes sense to use the CX-Server Lite .Net Communications Control in Windows applications that do not require the use of the Omron ActiveX graphical controls, or where there may be advantages to using a native .Net control within an application created using Visual Studio 2008.

The CX-Server Lite .Net Communications Control includes all commonly used functionality available in the ActiveX Control except for the Temperature Controller support.

For a full description of all design-mode and run-mode functionality supported by this control please see Appendices A and B. Please consult Appendix C for details of some common problems encountered when using Visual Studio (e.g. data events stopping working after a new version of software is installed).

Wherever practical, the same method and parameter names have been used as were used in the ActiveX control. However, in some cases, even where the functionality is quite similar, a different name has been chosen. This has been done for one or more of the following reasons:

a) To follow standard Microsoft .Net naming conventions

b) To better reflect use in a programming language rather than script-language based environment (the .Net component will be used with C# and VB.Net which have a very different syntax from VBScript, VBA and even VB6).

c) To standardise on a name used by the OPC .Net Communications Control

d) As with OPC, to use “device” rather than “PLC” because in future devices other than PLCs will be available for use with CX-Server.

To add the CX-Server Lite .Net Communications Control to the Visual Studio .Net Toolbox:

1. Click on the Toolbox (on the left side of the Visual Studio window)
2. Click on the Toolbox Components tab
3. Right-click on the background of the Toolbox window and select “Add/Remove Items” from the popup menu.

4. Scroll the list box down in the .Net Framework Components Tab, and select the "CXSLiteCtrl". *(Note: if for some reason the control is not visible, then use the Browse button on the dialog to browse to the directory where the control was installed (the filename is CXSLite.dll) and select it. The default installation directory is c:/program files/common files/Omron/Components)*
5. Select OK. The selected component will now be added to the Toolbox window.

To use the CX-Server Lite .Net Communications Control

1. Drag the CX-Server Lite Control ("CXSLiteCtrl") from the Toolbox window and drop onto the form.

To configure the component, use the standard Visual Studio Property Editor Window. Clicking the **…** button alongside the **ProjectName** property will open the Open Project dialog allowing you to navigate to the appropriate file.

Clicking the **…** button alongside the **Groups** property opens the CX-Server Project Editor dialog. This dialog is supported by CX-Server and follows the standard Windows Explorer format. The left 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.

To add events use the standard Visual Studio .Net technique (e.g. when creating a C# Windows Application use the event editor – accessed via the lightning-flash symbol at the top of the properties window). In VB.Net the events can be accessed by using the class name and method name drop-down list boxes which are displayed at the top of the code window (just below the tab for the VB source file).

# 5. Creating a CX-Server Lite Application in VS .Net

The following sections take you through the steps required to create a simple Windows form based application in Visual Studio .Net. Two applications will be created. The first application will use the CX-Server Lite Active X Controls, showing automatic linking between the graphical controls and the communications control. The second will show simple connection to an Omron PLC using the CX-Server Lite .Net interface component.

For a full description of all design-mode and run-mode functionality supported by this control please see Appendices A and B.

## 5.1 Example Viewing PLC Data using an Omron Graphical Control

The following sections take you through the steps required to open your selected application and create a working area using Windows Forms within Visual Studio .Net. 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.

### 5.1.1 Adding the Controls to the Toolbox

Start the Windows Form application and ensure the form is in design mode. If the ActiveX objects are not visible in the Toolbox they can be added as follows:

***1, 2, 3…***

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

### 5.1.2 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. Ensure the relevant form is active in design mode (e.g. by clicking on the relevant tab at the top of the screen, or by right-clicking on the name of the form in the solution explorer, and selecting "View Designer"

2. Select the CX-Server Communication Control component from the **Toolbox** and draw a rectangle at the desired position.

3. 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.

## 5.1.3 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 CX-Supervisor). 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 **ActiveX Properties** option.
2. In the Communication Control Properties dialog 'Project File' 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 other 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 programming commands (see Appendix B), or by adding a Graphical Component.

## 5.1.4 Adding a 7 Segment display

***1, 2, 3…***

1. With the target form in design mode, drag and drop a 7 Segment control onto it.
2. Right click on the graphical component and from the popup menu select the **Active X Properties** option.
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.

## 5.1.5 Running the Application

From the Debug Menu select the Start option. The project will be compiled and run (provided that there are no errors). The form will appear, and the (invisible) communications control will connect to the PLC. After a short delay, depending on the PLC communications medium, the 7 Segment display will show the PLC value.

## 5.2 Using CX-Server Lite Controls with C# and VB.NET

### 5.2.1 Step by Step example in C#

- Start a Visual Studio Windows Form application, and, using the steps outlined in the previous example, add an Omron CX Graphical Control to a form.
- Add a standard command button (e.g. double click on the icon in the Visual Studio toolbox), and then double click on it to bring up the Editor.
- Type the name of the CX Graphical Control (e.g. axGauge1) followed by a dot. If you have typed the name correctly then a list box will appear. Scroll down this list box to find the Value command (note: you can also just type v and the list box will change selection to the correct item; if the list box disappears, e.g. because you have switched back to your web browser to read these instructions, then just press backspace then retype the dot and it will reappear). Now type the = character and then a number, e.g. 10, followed by a semi-colon to terminate the line. The line of C# code now reads something like axGauge1.Value = 10 ;
- To run the application, click the select Start on the Debug menu.
- To test your code, press the command button. The gauge will change to display the value that you have set.

### 5.2.2 Step by Step example in VB.Net

One of the many benefits of Microsoft .Net is that the process for creating C# and VB.Net applications is almost identical. Follow exactly the same steps as in the previous example, but use VB syntax rather than C# syntax for the line of code (sometimes this is as simple as leaving off the semi-colon at the end of the line).

## 5.3 Using the CX-Server Lite Communications Control (.Net version)

The process for using the .Net version of the CX-Server Lite Communications control is very similar to that for the ActiveX version.

## 5.3.1 Adding the Control to the Toolbox

Start the Windows Form application and ensure the form is in design mode. If the control is not visible in the Toolbox it can be added as follows:

***1, 2, 3…*** 1. Right click in the Components tab of the Toolbox and select the **Add/Remove Items…** option. This will open the components selection dialog.

2. Find the CX-Server Lite controls in the .Net Framework Components Tab, and select the CX Server Lite Control ("CXSLiteCtrl") by ticking the box alongside the name.

3. Click the **OK** button. The objects are now displayed in the Toolbox.

## 5.3.2 Adding the Communications Control

To add the Communications Control to the form:

***1, 2, 3…*** 1. Ensure the relevant form is active in design mode (e.g. by clicking on the relevant tab at the top of the screen, or by right-clicking on the name of the form in the solution explorer, and selecting "View Designer"

2. Select the CX-Server Communication Control component from the **Toolbox** and draw a rectangle at the desired position.

3. 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.

## 5.3.3 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 CX-Supervisor). 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…***


5. Right-click on the CX-Server communications control. In the popup menu, select the **Properties** option.

6. In the Properties window 'Project Name' field enter or select the following.

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

   - Click the ellipsis (**…**) button alongside the ProjectName field 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 other 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:**

- ***Need to implement this functionality – it's missing!***
-

7. Projects can be edited by clicking the **Points…** 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.
8. If only editing of device configuration is required then the **Devices…** button can be used rather than the **Points…** button
9. Click the **OK** button to complete the configuration.

## 5.3.4 Accessing the Communications Control Runtime Interface

The Communications Control can be driven in the runtime via methods and events (although properties are also available these are normally set in design mode and not changed in runtime).

For a full description of all design-mode and run-mode functionality supported by this control please see Appendices A and B.

The following example provides a simple illustration of using the .Net version of the Communications Control to connect to a PLC. For a much fuller example see the separate Tutorial.

- Start a C# Visual Studio .Net Windows Form application, and, using the steps outlined in the previous example, add a Communications Control to a form.
- Add a standard command button (e.g. double click on the icon in the Visual Studio toolbox), and then double click on it to bring up the Editor.
- Add the following C# code (changing the name of the control, device and point name if required):

```
object Value = null;
bool BadQualityFlag = false;
bool ReadOK = cxsLiteCtrl1.Read(“MyDevice”, ”MyPoint”, out Value, out BadQualityFlag);
if (ReadOK)
        txtReadVal.Text = Value.ToString(); // displaying the value read in text box
```

- To run the application, click the select Start on the Debug menu.
- To test your code, press the command button.

# Appendix A
# Design Mode Properties

*Note: For details of the **ActiveX** design mode properties see the main CX-Server Lite User Manual.*

The following CX-Server Lite Communications Control configuration properties available are in design mode within Visual Studio. To edit them use the Visual Studio Properties Editor.

| Property Title | Example Values | Description |
|---|---|---|
| Devices | - | This is a string array that contains the list of devices (PLCs) in the CX-Server Project File. To edit the list of devices click on the ellipsis button (…) to invoke the CX-Server Device Editor. |
| Points | - | This is a string array that contains the list of points in the CX-Server Project File. To edit the list of points click on the ellipsis button (…) to invoke the CX-Server Project Editor. |
| ProjectName | "C:\CS1H.CDM" | CX-Server Project File Name. To select a project file click on the ellipsis button (…) to invoke the Windows file open dialog. |

The following is a list of some of the most useful standard properties available in design mode within Visual Studio. To edit them use the Visual Studio Properties Editor. For more information on these and other standard properties consult the Microsoft Visual Studio documentation.

| Property Title | Example Values | Description |
|---|---|---|
| (Name) | "cxsLiteCtrl1" | This is the name used for the Lite control. |
| Enabled | true | Enables or disables control in runtime |
| Size | 64,64 | Gets and sets the height and width of the control |
| Visible | true<br>false | Switches the object between visible and invisible in run mode.<br>True = Visible, False = Invisible. |

# Appendix B
# Run Mode Interface

*Note: For details of the **ActiveX** design mode properties see the main CX-Server Lite User Manual.*

The CX-Server Lite Interface defines the methods and properties made available by the .Net version of the CX-Server communications control. These methods can be used in any Microsoft .Net language (e.g. C#, VB.NET). Only the most important properties for running applications are discussed here; those properties normally only used in design mode are discussed in Appendix B.

*Note: Methods and events have generally been named in the same way as the ActiveX Lite Communications Control. However, in some cases, even where the functionality is quite similar, a different name has been chosen. This has been done for one or more of the following reasons:*

*a) To follow standard Microsoft .Net naming conventions*

*b) To better reflect use in a programming language rather than script-language based environment (the .Net component will be used with C# and VB.Net which have a very different syntax from VBScript, VBA and even VB6).*

*c) To standardise on a name used by the OPC .Net Communications Control*

*d) As with OPC, to use "device" rather than "PLC" because in future devices other than PLCs will be available for use with CX-Server*

*e) By the addition of device parameters where not previously required. This is necessary because in the future point names may not always be unique (e.g. the same point name could be used in different devices).*

| | |
|---|---|
| **Connect** | Connects to CX-Server project file |
| **Disconnect** | Disconnects from CX-Server |
| **OpenDevice** | Opens the specific device for communications. |
| **CloseDevice** | Closes the specific device. |
| **GetData** | Function for starting OnDataChange events. |
| **StopData** | Function for stopping OnDataChange events. |
| **IsDataGathered** | Indicates whether OnDataChange events are occurring for a point. |
| **OnDataChange** | Event for receiving notification of a change in data. |
| **RunMode** | Function for reading the current mode of the PLC. |
| **SetMode** | Function for setting the current mode of the PLC. |
| **TypeName** | Function for reading the PLC type (e.g. CQM1H). |
| **IsDevice** | Checks whether a specified device exists. |

| | |
|---|---|
| **IsPoint** | Checks whether a specified point exists. |
| **IsBadQuality** | Checks whether a point is currently indicating “bad quality”. |
| **ListDevices** | Returns a list of the devices (e.g. PLCs) in a project. |
| **ListPoints** | Returns the list of points in a project (or in a device). |
| **Read** | Reads a value |
| **Write** | Writes a value |
| **ReadAsync** | Reads a value asynchronously |
| **WriteAsync** | Writes a value asynchronously |
| **OnReadComplete** | Event that returns data read asynchronously |
| **ClockRead** | Reads the PLC Clock |
| **ClockWrite** | Sets the PLC Clock |
| **RawFins** | Enables raw FINS commands to be sent to a specified PLC |
| **GetLastError** | Returns the last error message (if any) |
| **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) |
| **GetNumErrors** | Count of all errors occurred since control was first run |
| **IsActive** | Returns if a device is active (i.e. open or active comms occurring) |

## B.1 Connect

Connects to CX-Server using current project file, or, optionally, specified project name. Generally it is not necessary to call this explicitly, because any attempt to communicate with will cause a connection to CX-Server to automatically be started, using current project name.

Note: If a project name other than that of the currently connected project is used, then the existing connection will be disconnected and a new one made using the newly specified project name. The current project name property “ProjectName” will also be updated to the new name.

Examples (in C#)

*bool ConnectedOK = cxsLiteCtrl1.Connect(); // uses current project name*

*bool ConnectedOK = cxsLiteCtrl1.Connect(“Project1.cdm”); // uses specified CX-Server project name*

Examples (in VB.Net)

*Dim ConnectedOK as Boolean*

*ConnectedOK = CxsLiteCtrl1.Connect() ‘ uses current project name*

Or

*Dim ConnectedOK as Boolean*

*ConnectedOK = CxsLiteCtrl1.Connect(“Project1.cdm”) ‘uses specified project name*

## B.2 Disconnect

Closes a connection to CX-Server.

Example (in C#)

*bool DisconnectedOK = cxsLiteCtrl1.Disconnect();*

Example (in VB.Net)

*Dim DisconnectedOK as Boolean*

*DisconnectedOK = CxsLiteCtrl1.Disconnect()*

## B.3 OpenDevice

Opens a device for communications. While it is generally not strictly necessary to open a device, because any attempt to communicate with it will cause the device to automatically be opened, it is often more efficient to do so (otherwise the device may be repeatedly automatically opened and closed each time any communications occurs).

Please note that device names should **not** have any spaces within them.

Example (in C#)

*bool OpenedOK = cxsLiteCtrl1.OpenDevice(“MyDevice”); // will open the device*

*If (OpenedOK)*

*cxsLiteCtrl1.Write(“MyDevice”,“MyPoint”,10); // MyPoint is within MyDevice*

Example (in VB.Net)

*Dim OpenedOK As Boolean*

*OpenedOK = CxsLiteCtrl1.OpenDevice("MyDevice 1") ‘ will open the device*

*If (OpenedOK = True) Then*

*CxsLiteCtrl1.Write("MyDevice", " MyPoint ", 99) ‘MyPoint is within MyDevice*

*End If*

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

*bool OpenedOK = cxsLiteCtrl1.OpenDevice("MyDevice**/force**");*

*or*

*bool OpenedOK = cxsLiteCtrl1.OpenDevice("MyDevice**/clearforce**");*

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

## B.4 CloseDevice

Closes a previously opened device.

Please note that device names should **not** have any spaces within them.

Example (in C#)

*bool Closed = cxsLiteCtrl1.CloseDevice("MyDevice");*

Example (in VB.Net)

*Dim Closed As Boolean*

*Closed = CxsLiteCtrl1.CloseDevice("MyDevice"")*

Note: if the device was already closed this method returns True

Optional parameter: The name of the device being closed can be 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.

## B.5 GetData

Starts asynchronous data reading of the specified point at the requested update rate. The returned values are returned in OnDataChange events.

Example 1 (in C#)

*cxsLiteCtrl1.GetData( "MyDevice", "MyPoint", UpdateRate);*

(in VB.Net)

*CxsLiteCtrl1.GetData( "MyDevice", "MyPoint", UpdateRate)*

In this example, MyPoint in MyDevice would be read at the rate of UpdateRate (seconds). Data is then sent to the OnDataChange event handler.

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

Example 2 (in C#)

*int UpdateRate = 2; // every 2 seconds*

*cxsLiteCtrl1.GetData("MyDevice", "MyPoint[2]", UpdateRate);*

(in VB.Net)

*Dim UpdateRate As Integer*

*UpdateRate = 2 'every 2 seconds*

*CxsLiteCtrl1.GetData("MyDevice", "MyPoint[2]", UpdateRate)*

In this example, the third element of the point MyPoint (defined in CX-Server 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.

Example 3 (in C#)

*cxsLiteCtrl1.GetData("MyDevice", "MyPoint", 0.5);*

(in VB.Net)

*CxsLiteCtrl1.GetData("MyDevice", "MyPoint", 0.5)*

In this example the point MyPoint in MyDevice will be read every 0.5 seconds and data returned in the OnDataChange routine.

## B.6 StopData

Stops asynchronous data reading of the specified point.

Example (in C#)

*cxsLiteCtrl1.StopData( "MyDevice", "MyPoint" );*

Example (in VB.Net)

*CxsLiteCtrl1.StopData( "MyDevice", "MyPoint" )*

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

## B.7 IsDataGathered

Indicates whether data is currently being obtained for the specified point (i.e. whether GetData has been called, and StopData had not been called, for that specified point)

Example (in C#)

*bool Gathered = cxsLiteCtrl1.IsDataGathered( "MyDevice", "MyPoint" );*

Example (in VB.Net)

*Dim Gathered As Boolean*

*Gathered = CxsLiteCtrl1.IsDataGathered( "MyDevice", "MyPoint" )*

## B.8 OnDataChange (Event)

This event is sent back to the application when GetData has been called and new data is available.

Example (in C#)

*Private void cxsLiteCtrl1_OnDataChange(object sender, CXSLite.CXSLiteCtrl.DataChangeArgs e)*

*{*

*string PointName = e.Point;*

*string DeviceName = e.Device;*

*txtVal.Text = e.ValueToString();*

*}*

Example (in VB.Net)

*Private Sub CxsLiteCtrl1_OnDataChange(ByVal sender As Object, ByVal e As CXSLite.CXSLiteCtrl.DataChangeArgs) Handles CxsLiteCtrl1.OnDataChange*

*Dim DeviceName As String*

*Dim PointName As String*

*Dim BadQuality As Boolean*

*DeviceName = e.Device*

*PointName = e.Point*

BadQuality = e.BadQuality

*TextBox1.Text = Convert.ToString(e.Value) ' TextBox1 is a text box*

*End Sub*

## B.9 RunMode

Reads the current operating mode of a PLC (Stop/Program, Debug, Monitor, Run), where 0=Stop/Program mode, 1=Run, 2=Monitor mode.

Example (in C#)

*int Mode = 0;*

*bool bOK = cxsLiteCtrl1.RunMode("MyDevice", out Mode);*

Example (in VB.Net)

*Dim Mode as Integer*

*Dim bOK as Boolean*

*bOK = CxsLiteCtrl1.RunMode("MyDevice", Mode)*

In this example, the operating mode would be read from 'MyDevice' and stored in 'Mode'. If 'MyDevice' was in 'Monitor' mode then 'Mode' would be set to the value 2.

## B.10 SetMode

Sets the current operating mode of a PLC (Stop/Program, Debug, Monitor, Run), where 0=Stop/Program mode, 1=Run, 2=Monitor mode.

Example (in C#)

*int Mode = 2;*

*bool bOK = cxsLiteCtrl1.SetMode("MyPLC", Mode);*

Example (in VB.Net)

*Dim Mode as Integer*

*Dim bOK as Boolean*

*Mode = 2*

*bOK = CxsLiteCtrl1.SetMode("MyPLC", Mode)*

In this example, the operating mode of 'MyPLC' would be set to the value 2 (monitor mode).

## B.11 TypeName

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

Example (in C#)

```
string PLCType = null;
bool bOK  = cxsLiteCtrl1.TypeName("MyDevice", out PLCType);
```

Example (in VB.Net)

```
Dim PLCType as String
Dim bOK as Boolean
PLCType = Nothing
bOK = CxsLiteCtrl1.TypeName("MyDevice", PLCType)
```

In this example, the PLC model type of 'MyDevice' will be read from the CX-Server (.CDM) project file and stored in 'PLCType'.

## B.12 IsDevice

Checks if a device exists (this will depend on the currently loaded project file)

Examples (in C#)

```
bool IsValid = cxsLiteCtrl1.IsDeviceValid("MyDevice");
```

Example (in VB.Net)

```
Dim IsValid as Boolean
IsValid = CxsLiteCtrl1.IsDeviceValid("MyDevice")
```

The boolean variable IsValid is set True if the device "MyDevice" exists.

## B.13 IsPoint

Checks if a point exists (this will depend on the currently loaded project file)

Examples (in C#)

```
bool IsValid = cxsLiteCtrl1.IsPointValid("MyDevice", "MyPoint");
```

Example (in VB.Net)

```
Dim IsValid as Boolean
IsValid = CxsLiteCtrl1.IsPointValid("MyDevice","MyPoint")
```

The boolean variable IsValid is set True if the point "MyPoint" exists.

## B.14 IsBadQuality

Checks whether a point is currently indicating “Bad Quality”.

Example (in C#)

```
bool IsBad = cxsLiteCtrl1.IsBadQuality(“MyDevice”, “MyPoint”);
```

Example (in VB.Net)

```
Dim IsBad as Boolean
IsBad = CxsLiteCtrl1.IsBadQuality(“MyDevice”,”MyPoint”)
```

The boolean variable IsBad is set true if the point “MyPoint” is indicating “Bad Quality” (e.g. the associated PLC is disconnected).

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

## B.15 ListDevices

Returns an array of devices in a project

Example (in C#)

```
string[] Names = null;
cxsLiteCtrl1.ListDevices(out Names);
```

Example (in VB.Net)

```
Dim Names as Array
CxsLiteCtrl1.ListDevices(Names)
```

## B.16 ListPoints

Returns an array of points in a project or device. To specify all points in the project, use “*”.

Example 1 (in C#)

```
string[] Names = null;
cxsLiteCtrl1.ListPoints(“MyDevice”, out Names);
```

Example 1 (in VB.Net)

```
Dim Names as Array
CxsLiteCtrl1 .ListPoints(“MyDevice”, Names)
```

Example 2 (in C#)

*string[] Names = null;*

*cxsLiteCtrl1.ListPoints("*", out Names); // return complete list of all points in project*

Example 2 (in VB.Net)

*Dim Names as Array*
*CxsLiteCtrl1.ListPoints("*", Names) ' return complete list of all points in project*

## B.17 Read

Function that reads a value from a device

Example (in C#)

*object Value = null;*

*bool BadQualityFlag = false;*

*bool ReadOK=cxsLiteCtrl1.Read("MyDevice", "MyPoint", out Value, out BadQualityFlag);*

*if (ReadOK)*

*txtReadVal.Text = Value.ToString(); // displaying the value read in text box*

Example (in VB.Net)

*Dim BadQualityFlag as Boolean*

*Dim ReadOK as Boolean*

*Dim Value as Object*

*Value = Nothing*

*BadQualityFlag = False*

*ReadOK = CxsLiteCtrl1.Read("MyDevice","MyPoint", Value, BadQualityFlag)*

*If (ReadOK = True) Then*

*TxtReadValue.Text = Value.ToString()*

*End If*

## B.18 Write

Function that writes a value to a device

Example (in C#)

*bool WrittenOK = cxsLiteCtrl1.Write("MyDevice", "MyPoint", Value);*

Example (in VB.Net)

*Dim WrittenOK as Boolean*

WrittenOK = CxsLiteCtrl1.Write*("MyDevice","MyPoint", Value)*

## B.19 ReadAsync

Function that reads a value from a device asynchronously. The value will be returned in the ReadComplete event.

Example (in C#)

*bool ReadOK = cxsLiteCtrl1.ReadAsync("MyDevice","MyPoint");*

Example (in VB.Net)

*Dim ReadOK as Boolean*

*ReadOK = CxsLiteCtrl1.ReadAsync("MyDevice","MyPoint")*

Example of callback event processing (in C#)

```
     private void cxsLiteCtrl1_OnReadComplete(object sender,
CXSLite.CXSLiteCtrl.DataChangeArgs e)
     {
             bool BadQuality = e.BadQuality;
             string Device = e.Device;
             string Point = e.Point;
             object Value = e.Value;
             label1.Text = Device;
             label2.Text = Point;
             label3.Text = Value.ToString();
      }
```

## B.20 WriteAsync

Function that writes a value to a device asynchronously.

Example (in C#)

int Value = 12;

*bool WritingOK = cxsLiteCtrl1.WriteAsync("MyDevice","MyPoint", Value);*

Example (in VB.Net)

```
Dim WritingOK as Boolean
Dim Value as Integer
Value = 12
WritingOK = CxsLiteCtrl1.WriteAsync("MyDevice","MyPoint", Value)
```

Note: The return flag indicates that the value has been successfully placed on the write queue – it does not indicate that the write has yet taken place or that it has been successful.

## B.21 OnReadComplete (Event)

This event is sent back to the application when ReadDataAsync has been called and new data is available.

Example (in C#)

```
Private void cxsLiteCtrl1_OnReadComplete(object sender, CXSLite.CXSLiteCtrl.DataChangeArgs e)
{
    string PointName = e.Point;
    string DeviceName = e.Device;
    txtVal.Text = e.Value.ToString(); // displays value as text
}
```

Example (in VB.Net)

```
Private Sub CxsLiteCtrl1_OnDataChange(ByVal sender As Object, ByVal e As
CX_SLite.CXSLiteCtrl.DataChangeArgs) Handles CxsLiteCtrl1.OnReadComplete
    txtVal.Text = e.Value.ToString()
End Sub
```

## B.22 ClockRead

Function that reads the PLC clock.

Example (in C#)

*DateTime PLCTime = new DateTime();*

*cxsLiteCtrl1.ClockRead("MyDevice", out PLCTime);*

Example (in VB.Net)

*Dim PLCTime As Date*

*CxsLiteCtrl1.ClockRead("MyDevice", PLCTime)*

## B.23 ClockWrite

Function that sets the PLC clock.

Example (in C#)

*DateTime ClockTime = new DateTime();*

*ClockTime = DateTime.Now;*

*bool WrittenOK = cxsLiteCtrl1.ClockWrite("MyDevice", ClockTime);*

Example (in VB.Net)

*Dim ClockTime as Date*

*Dim WrittenOK as Boolean*

*ClockTime = Date.Now*

WrittenOK = CxsLiteCtrl1.ClockWrite("MyDevice", ClockTime)

## B.24 RawFINS

Function that enables raw FINS commands to be sent to a specified PLC (Note: This is an advanced command, suitable for use only by those familiar with the Omron FINS protocol)

Example (in C#)

*String FINSResponse;*

*bool bReadOK = cxsLiteCtrl1.RawFINS("0501", "MyPLC",*
*CXSLite.UnitNumber.PLC_CPU, true, out FINSResponse);*

Example (in VB.NET)

*Dim FINSResponse As String*

*Dim OK as Boolean*

*OK = CxsLiteCtrl1.RawFINS("0501", "MyPLC", CXSLite.UnitNumber.PLC_CPU, True, FINSResponse)*

## B.25 GetLastError

Function that returns the last error as a string (blank if no error has occurred, or no suitable error message is available).

Example (in C#)

*String LastError = cxsLiteCtrl1.GetLastError();*

Example (in VB.Net)

*Dim LastError as String*

*LastError = cxsLiteCtrl1.GetLastError()*

## B.26 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 device is open and communicating.**

C# Example

*String NetworkNum = 1;*
*String NodeNum = 2;*
*String UnitNum = -1;*
*String iPAddress = "10.0.0.1";*
*Bool Valid = Comms1.SetDeviceAddress ("DEVICE1", NetworkNum, NodeNum, UnitNum, IPAddress);*

VB.Net Example

```
Dim strDevice As String
```

```
        Dim NetworkNum As Integer
        Dim NodeNum As Integer
        Dim UnitNum As Integer
        Dim strIPAddress As String
        Dim bSDASuccess As Boolean

        strDevice = comboListDevices.Text
        NetworkNum = 4
        NodeNum = 4
        UnitNum = 0
        strIPAddress = txtSetDeviceAddress_IPAddress.Text

        bSDASuccess = CxsLiteCtrl1.SetDeviceAddress(strDevice, NetworkNum,
NodeNum, UnitNum, strIPAddress)
```

Note: The return Boolean value 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.

## B.27 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 device is open and communicating.**

*C# Example:*

```
string config = "10.0.0.1";
bool Success = cxsLiteCtrl1.SetDeviceConfig("device1", "ADDRESS",
"IPADDR", config);
```

Note: The return Boolean value, Success, 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.

*VB.Net Example:*

```
        Dim strDevice As String
        Dim strSection As String
        Dim strEntry As String
        Dim strText As String
        Dim bSuccess As Boolean
```

```
        strText = "10.0.0.1"

        strDevice = comboListDevices.Text
        strSection = "ADDRESS"
        strEntry = "IPADDR"
        bSuccess = CxsLiteCtrl1.SetDeviceConfig(strDevice, strSection,
strEntry, strText)
```

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 = “A DDRESS”, Entry = “IPADDR”, Setting = “0.0.0.0”..Setting = “255.255.255.255” - this can be used to set the Ethernet IP address

Other parameter values may work, but should only be used on Omron advice.

## B.28 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.

C# Example

```
        bool Success = false;
        string config = cxsLiteCtrl1.GetDeviceConfig("device1", "ADDRESS",
"IPADDR", ref Success);
        if (Success)
            textBox1.Text = config;
```

VB.Net Example

```
        Dim strDevice As String
        Dim strSection As String
        Dim strEntry As String
        Dim strResult As String
        Dim bSuccess As Boolean
```

```
        strDevice = comboListDevices.Text
        strSection = "ADDRESS"
        strEntry = "DNA"
        strResult = CxsLiteCtrl1.GetDeviceConfig(strDevice, strSection,
strEntry, bSuccess)
```

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

## B.29 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 DEVICE communications. The project and PLC will automatically be opened if required.

C# Example:

```
bool Success = cxsLiteCtrl1.UploadProgram("device1",
"","d://test1.prg", false, true);
```

VB.Net Example

```
Dim strUploadDestination As String
Dim strUploadDevice As String
Dim strUploadSource As String
Dim bUploadWait As Boolean
Dim bUploadReportProgress As Boolean
Dim bUploadSuccess As Boolean

strUploadDevice = "PLC1_CJ1G_H"
strUploadSource = ""
strUploadDestination = "C:\Common\Test\PLC1\UploadCJ.bin"
bUploadWait = True
bUploadReportProgress = True
bUploadSuccess = CxsLiteCtrl1.UploadProgram(strUploadDevice,
strUploadSource, strUploadDestination, bUploadWait, bUploadReportProgress)
```

Note: The parameters, in order, are the PLC name, the source (see below), the destination file, whether to wait until complete, 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.

"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 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 without waiting for completion, 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 (in C#):

```
        private void cxsLiteCtrl1_OnDataChange(object sender,
CXSLite.CXSLiteCtrl.DataChangeArgs e)
        {
            bool BadQuality = e.BadQuality;
            string Device = e.Device;
            string Point = e.Point;
            object Value = e.Value;
         }
```

.. and in VB:

```
    Private Sub CxsLiteCtrl1_OnDataChange(ByVal sender As Object, ByVal e
As CXSLite.CXSLiteCtrl.DataChangeArgs) Handles CxsLiteCtrl1.OnDataChange
        Dim Devicename As String
        Dim PointName As String
        Dim BadQuality As Boolean

        Devicename = e.Device
        PointName = e.Point
        BadQuality = e.BadQuality
    End Sub
```

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

Device – 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)

## B.30 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.

C# Example

```
bool Success = cxsLiteCtrl1.DownloadProgram("device1",
“d:\\up1.prg", "", true, true);
```

VB.Net Example

```
Dim strDownloadDestination As String
Dim strDownloadDevice As String
Dim strDownloadSource As String
Dim bDownloadWait As Boolean
Dim bDownloadReportProgress As Boolean
Dim bDownloadSuccess As Boolean

strDownloadDevice = "PLC1_CJ1G_H"
strDownloadSource = " C:\Common\Test\PLC1\DownloadCJ.bin "
strDownloadDestination = "PRG"
bDownloadWait = True
bDownloadReportProgress = True
```

```
      bDownloadSuccess = CxsLiteCtrl1.DownloadProgram(strDownloadDevice,
strDownloadSource, strDownloadDestination, bDownloadWait,
bDownloadReportProgress)
```

Note: The parameters, in order, are the PLC name, the source file, the destination (see below), whether to wait until complete, 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.

"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 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 without waiting for completion, 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 (in C#):

```
        private void cxsLiteCtrl1_OnDataChange(object sender,
CXSLite.CXSLiteCtrl.DataChangeArgs e)
        {
            bool BadQuality = e.BadQuality;
            string Device = e.Device;
            string Point = e.Point;
            object Value = e.Value;
         }
```

.. and in VB:

```
    Private Sub CxsLiteCtrl1_OnDataChange(ByVal sender As Object, ByVal e
As CXSLite.CXSLiteCtrl.DataChangeArgs) Handles CxsLiteCtrl1.OnDataChange
        Dim Devicename As String
        Dim PointName As String
        Dim BadQuality As Boolean

        Devicename = e.Device
        PointName = e.Point
        BadQuality = e.BadQuality
    End Sub
```

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

Device – 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)

## B.31 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.

C# Example (sets protection for CS series PLC)

```
       bool Success = cxsLiteCtrl1.Protect("device1", true, "12345678",
2468);
```

VB.Net example:

```
        Dim bProtectOnSuccess As Boolean
        Dim strProtectOnDevice As String
        Dim strProtectOn_Password_String As String
        Dim strProtectOn_Password_Number As Long

        strProtectOnDevice = comboListDevices.Text
        strProtectOn_Password_String = txtPassword_String.Text
        strProtectOn_Password_Number = txtPassword_Number.Text

        bProtectOnSuccess = CxsLiteCtrl1.Protect(strProtectOnDevice, True,
strProtectOn_Password_String, strProtectOn_Password_Number)
```

Note: The parameters of this command are, in order:

Device – Name of device

EnableProtection – true to set password protection, false to unset it

PasswordString – Password as a string. For CS series PLCs this should be a string of up to 8 characters. For CV PLCs this should be a string of up to 8 characters containing a hexadecimal number, e.g. “12345678”. For C series PLCs this should be a string of up to 4 characters containing a hexadecimal number, e.g. “1234”.

PasswordNumber – currently this is only used for C and CV series PLCs, and only when the password string is empty. In those circumstances it is simply a number representing the value of the 4 or 8 digit password. Please note that the password is entered in CX-Programmer as a hexadecimal string (as with the PasswordString parameter above), and that, for example, the value 1234 in decimal is the equivalent to “04d2” as a hexadecimal password string.

Additional C Series PLC notes: For C series the PLC program needs code (the first line of the application) in the PLC to enable password setting/release, and this fixes the password value.

e.g. LD AR10.01
FUN49 0 0 #1234 (#1234 – password value in Hex)

When **setting** the password this value is used rather than the value passed – i.e. the password string or number is ignored. The correct password must be provided, however, when disabling the password protection.

## B.32 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:

```
cxsLiteCtrl1.InitCXServer(true); // Initialises CX-Server within a thread
cxsLiteCtrl1.InitCXServer(false); // Uninitialises CX-Server within a thread
```

VB examples:

```
CxsLiteCtrl1.InitCXServer(True)
CxsLiteCtrl1.InitCXServer(False)
```

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

## B.33 GetNumErrors

This property is a count of the number of errors that have occurred since the control was first run.

C# Example:

```
int n = cxsLiteCtrl1.GetNumErrors();
```

```
VB.Net Example
Dim intNumberOfErrors As Integer
intNumberOfErrors = CxsLiteCtrl1.GetNumErrors()
```

## B.34 IsActive

This property returns the connection status of a specified PLC.

C# example

```
bool IsConnected = Comms1.IsActive("MyPLC");
```

VB example

```
Dim IsConnected As Boolean
IsConnected = Comms1.IsActive("MyPLC")
```

In this example, the connected status would be read from 'MyPLC' and stored in 'IsConnected'. If 'MyPLC' is connected (the PLC is open or has any active data connection) 'IsConnected' would be set to true.

# Appendix C Visual Studio .Net Limitations and Workarounds

## C.1 Issues To Be Aware Of When Upgrading

Although Visual Studio is a powerful development environment, it is not perfect and does contain oddities and outright bugs. If you encounter an unexpected problem while using Visual Studio, or with a program produced by it, it is very important to check on the Internet to see if it is a known issue. It is also very often worthwhile installing the latest Microsoft patch.

Examples of common known potential issues include:

- Control stops working after a new version of software is installed
- Data events are not received (typically after a new version of software is installed)

These two issues are discussed below:

### C.1.1. Control stops working after a new version of software is installed

Microsoft Visual Studio keeps track of which version of controls an application was created with, and may refuse to work with other versions. This is deliberate functionality introduced by Microsoft to reduce the risk of errors (preventing an application from running with any version of the control other than the one it was created with). This problem can affect applications whenever a new version of CX-Server Lite/OPC is installed. In these cases it is necessary to delete the control, re-add it, and recompile the application.

The recommended procedure to replace the control is as follows:

1. Ensure that you have a complete backup of your application

2. Delete the control from the toolbox

3. Remove the reference to the control (CXSLite) from the references section of your application (in the Solution Explorer window). It may also be safer to delete the references to CXLicence and CXProvider.

4. Use the toolbox “Choose Items” method to add the new version of the control to your toolbox

5. Drag and drop the control onto your form

6. It may also be necessary to replace the OnData and other event connections (see below)

### C.1.2. Data events are not received (typically after a new version of software is installed)

On occasions Visual Studio seems to lose the mapping between events and the event handler routines. As a result data values are not returned by asynchronous read or GetData events. This happens in both C# and VB.NET, typically after a new version of a control is installed, but may happen at other times. In C# the event properties for the handler become blank and need to be reselected. In VB.Net it may be necessary to delete the event handler and recreate it. After this is done the application should resume working.

## C.2 Active X Compatibility

Although Visual Studio .Net ActiveX compatibility is very good, it is not perfect. This section will explain how to make workarounds to solve compatibility problems which arise when using the CX-Server Lite ActiveX control in C# or VB.Net. This section assumes that the reader has a high level of technical expertise, and is familiar with terms such as “Dispatch Interface”.

Dispatch interfaces in unmanaged COM that have a default property with more than one parameter are not always currently referenced correctly by Visual Studio .Net when a wrapper is automatically created for ActiveX components. The effect of this is that the ‘Value’ property will not be exposed when inserting or referencing the CX-Server Lite ActiveX Communications control in your WinApp C# or VB.Net application.

A solution to this problem is the use of *late binding* to invoke the ‘Value’ property. This way the application does not need to make direct reference to the assembly and there is no need to insert the ActiveX control into the application – it can be done via program code.

To insert a control dynamically, it is necessary to first get the class type associated with the specified program identifier and then to create an instance of the class.

The program identifier for the CX-Server Lite ActiveX Communications control is:

*"CXServerCommunicationControl.CXServerCommunicationControl.1"*

An instance of the control can be created by the following code:

```
public static Type comms =
Type.GetTypeFromProgID("CXServerCommunicationControl.CXServerCommunicationControl.1");

public static object target = Activator.CreateInstance(comms);
```

When an instance of the Lite control has been created, it is necessary before calling the ‘Value’ property to initialise the instance of the control. This is done by calling ProjectName and OpenProject methods using the function InvokeMember.

The InvokeMember method, takes an array of COM VARIANT types that represent the parameters passed to the function. In the case of the ProjectName method, this array contains one string parameter - the path where the CX-Server CDM file is located.

```
object [] path = {"C:\\test.cdm"};
comms.InvokeMember("ProjectName",BindingFlags.SetProperty,null,target,path);
```

comms.InvokeMember("OpenProject",BindingFlags.InvokeMethod,null,target,null);

Finally, using again the function InvokeMember, the 'Value' property can be called.

To read a value from the PLC passing the point name:

*object [] point = {"MyPoint"};*
*comms.InvokeMember("Value", BindingFlags.GetProperty, null, target, point);*

To write a value to the PLC passing the point name and the new value:

*object [] args = {"MyPoint", newValue};*
*comms.InvokeMember("Value", BindingFlags.SetProperty, null, target, args);*

**Note1: The System.Reflection namespace must be added to the application in order to use the methods defined in BindingFlags class.**

**Note2: VB.NET will prefix ActiveX property names with either get_ or set_. As an example, ListPoints will become get_ListPoints**

**Note3: Developers using .Net should be aware of garbage collection issues – memory is under the control of the .Net framework and may not be reclaimed for some time after an application has finished running. Where deleted or out-of-scope objects contain references to COM components (e.g. parts of CX-Server) it is possible that these applications will also remain open for some time until garbage collection occurs. Garbage collection can be forced by using the GC.Collect() method – for further details of recommended garbage collection techniques consult the Microsoft .Net documentation.**

# 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. |
| **Microsoft Excel** | A spreadsheet **application**. |
| **Microsoft Windows** | The most common operating system used by Personal Computers.<br>**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. |
| **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. |
| **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