CX-Server OPC
User Manual

Guide to using CX-Server OPC 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 OPC **client** application and its ability to interface with OPC servers. It does **not** provide detailed information concerning OPC itself, or the CX-Server OPC **server**. The main CX-Server OPC user manual and the "Guide to Developing OPC Applications" should be consulted for that information.

This manual contains the following:

- ***Using CX-Server OPC in Microsoft .Net:*** This describes the use of the CX-Server OPC software within the .Net environment in general terms
- ***Tutorial:*** This is a quick tutorial for use in Visual Studio .Net 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 OPC in Microsoft .Net

This book introduces the use of CX-Server OPC **client** components within a Microsoft .Net environment to a new user. It is assumed that the reader is already familiar with OPC and CX-Server OPC 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 OPC, for details of the CX-Server OPC server, for important system requirements and installation information, and for details of use in other, non .Net, applications. See the "Guide to Developing OPC applications" for information on the use of OPC.

1.1 Welcome to CX-Server OPC

CX-Server OPC allows PLC data collected by the OMRON CX-Server communications software to be accessed from OPC DA 1.0a and 2.0 clients. It also provides client facilities to allow data obtained from any OPC DA v2.0 server to be used. The use of those client facilities within a Microsoft Visual Studio .Net environment is described in this manual.

1.2 About this Manual

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

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

1.3 The Help system, and How to Access it

CX-Server OPC 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 OPC. 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 OPC.

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 OPC 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 OPC 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 OPC

The CX-Server OPC 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 OPC and the CX-Automation Suite can be accessed from the main help contents dialog.

1.5 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 OPC, 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 OPC in Microsoft .Net Overview

2.1 CX-Server OPC Communications Control (ActiveX version)

CX-Server OPC includes a standard ActiveX communications control that acts as a client interface to OPC DA 2.0 compliant 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 OPC user manual.

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

2.2 CX-Server OPC ActiveX Graphical Objects Overview)

CX-Server OPC 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 OPC 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 OPC Communications Control (.Net version)

This control provides a seamless interface between the CX-Server OPC host application (Visual Studio .Net) and any OPC DA v2.0 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 OPC Active X controls in Microsoft .Net Applications

Visual Studio .Net 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 OPC Active X controls in preference to the CX-Server OPC .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 OPC ActiveX Communications Control.

To add one of the Omron ActiveX controls to the Visual Studio .Net 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 the graphical controls 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 group, and an item. 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 OPC Communications Control (.Net version)

The .Net version of the CX-Server OPC 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 it is **not** intended for use on web pages.) 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).

It makes sense to use the CX-Server OPC .Net Communications Control in 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.

The CX-Server OPC .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 2003 (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 OPC .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 “CXSOpcCtrl”. *(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 CXSOPC.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 OPC .Net Communications Control

1. Drag the CX-Server OPC Control (“CXSOpcCtrl”) 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 **ProjectFile** 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 an event in C# use the Visual Studio event editor (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 OPC 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 OPC 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 OPC .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 OPC 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 OPC 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 OPC 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 OPC can communicate with an OPC server 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 OPC 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 an OPC Server

The following procedure takes you through the steps required connect to an OPC Server.

1, 2, 3…

1. Right click on the CX-Server OPC communications control. In the popup menu, select the **ActiveX Properties** option.
2. In the Communication Control Properties dialog select the Project file:
   i. • **To open an existing project (.OPC) 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.

   ii. • **To create a new project (.OPC) 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. By default, the Computer Name field shows the name of the local computer. If the OPC Server is on a remote machine, click **Show All**. This may take a few moments, depending on your network and operating system. When complete the Computer Name list now shows all computer names. Select the required computer, and wait while connection is tested.
4. The Server name list shows all the OPC version 2 compliant servers registered on the computer listed in Computer Name field. For the server included with CX-Server OPC select **Omron.OpenDataServer.1** from the list. The connection can be tested by clicking the **Info…** button that will display standard OPC status information collected from the OPC server.
5. At least one OPC client Groups must now be defined using the **Groups** tab. Choose any meaningful name.
6. Add an Item for the data to display using the **Items** tab. Choose any meaningful name e.g. OPCBoilerTemp. Type the Item ID of the item as defined by the OPC Server or use the **Browse…** button if the server

supports the optional 'Browse items' interface. For the server included with CX-Server OPC the Item ID is same as the CX-Server Point Name e.g. "BoilerTemp".

7. Click the **OK** button to complete the configuration.

The communications control is now ready to connect to the OPC Server, and retrieve data. 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 OPC Controls with C# and VB.NET

5.2.1 Step by Step example in C#

- Start a Visual Studio .Net 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 OPC Communications Control (.Net version)

The process for using the .Net version of the CX-Server OPC 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 OPC controls in the .Net Framework Components Tab, and select the CX Server OPC Control ("CXSOpcCtrl") 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 OPC 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 an OPC Server

The following procedure takes you through the steps required to connect to an OPC Server.

1, 2, 3… 

1. Right click on the CX-Server OPC communications control. In the popup menu, select the Properties option (this is part of the Configurations sub-category). This will ensure that the Visual Studio Properties Window is displayed.

2. In Properties Windows select (click on) the ProjectFile property:

i. • To open an existing project (.OPC) file:

- ♦ Click the ellipsis (…) button and in the Open 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.

ii. • To create a new project (.OPC) file:

- ♦ Click the ellipsis (…) button and in the Open 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 Open button, the full path name of the new file will be entered into the Project field.

8. By default, the Server Computer Name field shows (Local) or the name of the local computer. If the OPC Server is on a remote machine, select ServerComputerName and click on the drop down list button (down arrow).
9. The Server name list shows all the OPC version 2 compliant servers registered on the computer listed in Computer Name field. To refresh the list select ServerName and then click on the drop down list button. For the server included with CX-Server OPC select Omron.OpenDataServer.1 from the list.
10. At least one OPC client Group must now be defined. To do this select the Groups property and click on the ellipsis (…) button. A Group dialog will appear. To add a group right-click in the groups column and select “Add Group” from the pop-up menu. Fill in the group dialog, choosing any meaningful name.
11. Add an Item for the data to display using the Items column (the right portion of the dialog). Right-click in this area and choose “Add Item(s)” from the pop-up dialog. The selected OPC Server will be started and items can now be browsed. Highlight the items that you wish to add, and then use the “Add Items” button. Use the OK button to close the dialog.
12. Click the OK button to complete the configuration.

13. Projects can be edited by clicking the **Items…** button and then making the required changes from the Project Edit Dialog.
14. If only editing of device configuration is required then the **Devices…** button can be used rather than the **Items…** button
15. 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 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 code (changing the name of the control, device and point name if required):

```
Object Value = null;
Bool BadQualityFlag = false;
bool ReadOK = CxsOpcCtrl1.Read(“MyGroup”, ”MyItem”, Value, 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 OPC User Manual.*

The following CX-Server OPC 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
Groups	-	This is a string array that contains the list of groups (typically devices, e.g. PLCs) in the OPC Client Project File. To edit the list of groups click on the ellipsis button (…) to invoke the Device Editor.
Items	-	This is a string array that contains the list of items in the OPC Server Project File. To edit the list of items click on the ellipsis button (…) to invoke the CX-Server Project Editor.
ProjectFile	"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)	"CxsOpcCtrl1"	This is the name used for the OPC control.
Enabled	true	Enables or disables control in runtime
Size	64,64	Gets and sets the height and width of the control
Visible	true false	Switches the object between visible and invisible in run mode. True = Visible, False = Invisible.

Appendix B
Run Mode Interface

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

The CX-Server OPC 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 OPC 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 follow OPC naming conventions

c) 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).

d) To standardise on a name used by the Lite .Net Communications Control

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
GetData	Function for starting OnDataChange Events.
StopData	Function for stopping OnDataChange Events.
OnDataChange	Event for receiving notification of a change in data.
IsBadQuality	Checks whether an item is currently indicating “bad quality”.
ListGroups	Returns a list of the groups (devices, e.g. PLCs) in a project.
ListItems	Returns the list of items (e.g. points) in a project (or in a group).
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

OnWriteComplete	Event that confirms that WriteAsync data has been written.
Activate	Enables or disables communication for an item
IsGroup	Indicates whether a group name exists
IsItem	Indicates whether an item name exists
GetLastError	Returns string containing description of last error that occurred

B.1. Connect

Connects to an OPC Server.

Examples (in C#):

bool ConnectedOK = cxsOpcCtrl1.Connect(); // uses current project name

bool ConnectedOK = cxsOpcCtrl1.Connect("Project1.opx"); // uses specified project

Examples (in VB.Net)

Dim ConnectedOK as Boolean

ConnectedOK = CxsOpcCtrl1.Connect() ' uses current project name

Or

Dim ConnectedOK as Boolean

ConnectedOK = CxsOpcCtrl1.Connect("Project1.opx") 'uses specified project name

B.2 Disconnect

Closes a connection to an OPC Server

Example (in C#):

bool DisconnectedOK = cxsOpcCtrl1.Disconnect();

Example (in VB.Net)

Dim DisconnectedOK as Boolean

DisconnectedOK = CxsOpcCtrl1.Disconnect()

B.3. GetData

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

Example 1 (in C#):

cxsOpcCtrl1.GetData("MyGroup", "MyItem");

Example 1 (in VB.Net):

CxsOpcCtrl1.GetData("MyDevice", "MyItem")

In this example, MyItem in MyGroup would be read at the update rate associated with the group. Data is then sent to the OnDataChange event handler.

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

Example 2 (in C#)

cxsOpcCtrl1.GetData("MyGroup", "MyItem[2]");

Example 2 (in VB.Net):

CxsOpcCtrl1.GetData("MyDevice", "MyItem[2]")

In this example, the third element of the zero based array MyItem can now be accessed as MyItem[2]. By default, if the array element is not specified, then the whole point (whole array of elements) is manipulated.

B.4. StopData

Stops asynchronous data reading of the specified item.

Example (in C#)

cxsOpcCtrl1.StopData("MyGroup", "MyItem");

Example (in VB.Net)

CxsOpcCtrl1.StopData("MyDevice", "MyItem")

In this example, the reading of MyItem in MyGroup would be stopped.

B.5. OnDataChange (Event)

This event is sent back to the application when GetData has been called and new data is available. It returns a quality indicator that follows the OPC Standard, where common values (in hexadecimal) include:

OPC_QUALITY_BAD	0x00
OPC_QUALITY_UNCERTAIN	0x40
OPC_QUALITY_GOOD	0xC0

Masking the returned value with 0xC0 will return a simple good quality / bad quality indicator, where the quality is good if (and only if) the result is 0xC0

Example (in C#)

```
Private void cxsOpcCtrl1_OnDataChange(object sender, CXSOpc.CXSOpcCtrl.DataChangeArgs e)
{
    string GroupName = e.Group;
    string ItemName = e.Item;;
    int Quality = e.Quality; // IMPORTANT: This Quality integer follows OPC standard
    txtBox1.Text = e.ValueToString(); // txtBox1 is a text box
}
```

Example (in VB.Net)

```
Private Sub CxsOpcCtrl1_OnDataChange(ByVal sender As Object, ByVal e As
CXSOpc.CXSOpcCtrl.DataChangeArgs) Handles CxsOpcCtrl1.OnDataChange
    Dim Group As String
    Dim Item As String
    Dim Quality As Integer
    Group = e.Group
    Item = e.Item
    Quality = e.Quality 'IMPORTANT: This Quality integer follows OPC standard
    TextBox1.Text = Convert.ToString(e.Value) ' TextBox1 is a text box
End Sub
```

B.6. IsBadQuality

Checks whether an item is currently indicating “Bad Quality”.

Example (in C#)

Bad = cxsOpcCtrl1.IsBadQuality(“MyGroup”, “MyItem”);

Example (in VB.Net)

Dim Bad as Boolean

Bad = CxsOpcCtrl1.IsBadQuality(“MyGroup”, “MyItem”)

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

B.7. ListGroups

Returns an array of groups in a project

Example (in C#)

string[] Names = null;

cxsOpcCtrl1.ListGroups(out Names);

Example (in VB.Net)

Dim Names as Array
CxsOpcCtrl1 .ListGroups(Names)

B.8. ListItems

Returns an array of items in a group.

Example 1 (in C#)

string[] Names = null;

cxsOpcCtrl1.ListItems(“MyGroup”, out Names);

Example 1 (in VB.Net)

Dim Names as Array
CxsLiteCtrl1 .ListItems(“MyGroup”, Names)

B.9. Read

Function that reads a value from a group

Example (in C#):

```
Object Value = null;
bool ReadOK = cxsOpcCtrl1.Read("MyGroup", "MyItem", out Value);
if (ReadOK)
        txtBox1.Text = Value.ToString(); // displaying the value read in text box
```

Example (in VB.Net)

```
Dim OK as Boolean
Dim Value as Object
Value = Nothing
OK = CxsOpcCtrl1.Read("MyDevice", "MyPoint", Value)
If (OK = True) Then
        TextBox1.Text = Value.ToString()
End If
```

B.10. Write

Function that writes a value

Example (in C#):

```
bool WrittenOK = cxsOpcCtrl1.Write("MyGroup", "MyItem", Value);
```

Example (in VB.Net):

```
Dim Value as Object
Dim WrittenOK as Boolean
Value = 10 ' set Value to some appropriate example value
WrittenOK = CxsOpcCtrl1.Write("MyGroup", "MyItem", Value)
```

B.11. ReadAsync

Function that reads a value asynchronously. The value will be returned in the OnReadComplete event.

Example (in C#):

bool ReadingOK = cxsOpcCtrl1.ReadAsync("MyGroup","MyItem");

Example (in VB.Net):

Dim ReadingOK As Boolean

ReadingOK = CxsOpcCtrl1.ReadAsync("MyGroup","MyItem")

B.12. WriteAsync

Function that writes a value asynchronously. Confirmation that the value has been written will be returned in the OnWriteComplete event.

Example (in C#):

bool WritingOK = cxsOpcCtrl1.WriteAsync("MyGroup","MyItem", Value);

Example (in VB.Net):

Dim Value as Object

Dim WritingOK as Boolean

Value = 10 ' set Value to some appropriate example value

WritingOK = CxsOpcCtrl1.WriteAsync("MyGroup", "MyItem", Value)

B.13. 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 cxsOpcCtrl1_OnReadComplete(object sender, CXSOPC.CXSOPCCtrl.DataChangeArgs e)
{
        TxtVal.Text = e.ValueToString();
        string GroupName = e.Group;
        string ItemName = e.Item;;
        int Quality = e.Quality; // IMPORTANT: This Quality integer follows OPC standard
}
```

Example (in VB.Net)

```
 Private Sub CxsOpcCtrl1_OnReadComplete(ByVal sender As Object, ByVal e As CXSOpc.CXSOpcCtrl.ReadCompleteArgs) Handles CxsOpcCtrl1.OnReadComplete
        Dim Group As String
        Dim Item As String
        Dim Quality As Integer
        Group = e.Group
        Item = e.Item
        Quality = e.Quality 'IMPORTANT: This Quality integer follows OPC standard
        TextBox1.Text = Convert.ToString(e.Value) ' TextBox1 is a text box
   End Sub
```

Note: See the OnDataChange event for a description of the “quality” status integer

B.14. OnWriteComplete (Event)

This event is sent back to the application when the data associated with WriteDataAsync has been written.

Example (in C#)

```
Private void cxsOpcCtrl1_OnWriteComplete(object sender, CXSOpc.CXSOpcCtrl.WriteCompleteArgs e)
{
        string Group = e.Group;
        string Item = e.Item;
        bool Success = e.Success;
}
```

Example (in VB.Net):

```
Private Sub CxsOpcCtrl1_OnWriteComplete(ByVal sender As Object, ByVal e As CXSOpc.CXSOpcCtrl.WriteCompleteArgs) Handles CxsOpcCtrl1.OnWriteComplete
        Dim Group As String
        Dim Item As String
        Dim Success As Boolean
        Group = e.Group
        Item = e.Item
        Success = e.Success
End Sub
```

B.15. Activate

Function that enables or disables communication for a group or an item.

Examples (in C#):

```
bool Success  = cxsOpcCtrl1.Activate("MyGroup",  true); // enables comms for group
Success  = cxsOpcCtrl1.Activate("MyGroup", false); // disables comms for group
Success  = cxsOpcCtrl1.Activate("MyGroup", "MyItem", true); // enables item
Success  = cxsOpcCtrl1.Activate("MyGroup", "MyItem", false); // disables item
```

Examples (in VB.Net):

```
Dim Success as Boolean
Success = CxsOpcCtrl1.Activate("Group",  True) ' enables comms for group
Success = CxsOpcCtrl1.Activate("Group",  False) ' disables comms for group
Success = CxsOpcCtrl1.Activate("Group", "Item", True) ' enables item
Success = CxsOpcCtrl1.Activate("Group", "Item", False) ' disables item
```

B.16. IsGroup

Function that returns whether a group exists.

Example (in C#):

```
bool Exists  = cxsOpcCtrl1.IsGroup( "MyGroup");
```

Example (in VB.Net):

```
Dim Exists As Boolean
Exists  = cxsOpcCtrl1.IsGroup( "MyGroup")
```

B.17. IsItem

Function that returns whether an item exists.

Example (in C#):

bool Exists = cxsOpcCtrl1.IsItem("MyGroup","MyItem");

Example (in VB.Net):

Dim Exists As Boolean

Exists = cxsOpcCtrl1.IsItem("MyGroup", "MyItem")

B.18. 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 = cxsOpcCtrl1.GetLastError();

Example (in VB.Net)

Dim LastError as String

LastError = cxsOpcCtrl1.GetLastError()

Note: Simple communications connection/disconnection errors that are indicated by the use of the "bad quality" status flag do not necessarily cause the last error string to be replaced, although there may be an associated error that does.

B.19. IsActive

Function that returns the active state of a group or item.

Examples (in C#):

bool Active = cxsOpcCtrl1.IsActive("MyGroup"); // returns active state for group

Active = cxsOpcCtrl1.IsActive("MyGroup", "MyItem); // returns active state for item

Examples (in VB.Net):

Dim Active As Boolean

Active = CxsOpcCtrl1.IsActive("MyGroup") ' returns active state for group

Active = CxsOpcCtrl1.IsActive("MyGroup", "MyItem") 'returns active state for item

Appendix C Visual Studio .Net Limitations and Workarounds

C.1 Potential Problem Areas

Although Visual Studio .Net 2003 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 .Net, 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 .Net 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.

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

On occasions Visual Studio .Net 2003 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 some of the compatibility problems which arise when using the CX-Server OPC 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 OPC 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 OPC ActiveX Communications control is:

" CXOPCClientCommunications.CXOPCClientCommunicationsCtrl.1"

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

```
public static Type OPCcomms =
Type.GetTypeFromProgID("CXOPCClientCommunications.CXOPCClientCommunicationsCtrl.1");

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

When an instance of the OPC control has been created, it is necessary before calling the ‘Value’ property to initialise the instance of the control. This is done by calling ProjectFile and ServerName properties 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 ProjectFile property, this array contains one string parameter - the path where the OPC file is located.

```
object [] path = {"C:\\test.opc"};
OPCcomms.InvokeMember("ProjectFile",BindingFlags.SetProperty,null,target,path);
object [] servername = {"OMRON.OpenDataServer.1"};

OPCcomms.InvokeMember("ServerName",BindingFlags.SetProperty,null,target,servername);
```

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

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

object [] args = {"MyGroup", "MyItem"};
OPCcomms.InvokeMember("Value", BindingFlags.GetProperty, null, target, args);

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

object [] args = {"MyGroup", "MyItem", newValue};
OPCcomms.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 an item that defines the origin and destination of the data for that **point**. The data for an item 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. **CX-Server OPC** 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	An item is used to hold a value of a predefined type - Boolean, Integer, Text, etc. The contents of an item may be controlled by an object or I/O mechanism such as **DDE**. The contents of an item 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