A Guide to Developing OPC Applications using CX-Server OPC

1. Introduction

1.1 The Purpose of this document

This document is intended to help engineers using the Omron CX-Server OPC software create good (effective and efficient) OPC applications. It discusses, in varying detail as required, some of the key concepts in using CX-Server OPC, and provides examples of the use of those concepts. It points out a few common mistakes and how to avoid them.

It is *not* intended to provide a full reference to OPC, or any programming languages, or to PLC technology. Those documents are available elsewhere.

Nor is it a full reference to CX-Server OPC. For details of all CX-Server OPC functionality covered in this document, please see the separate CX-Server OPC Manual, which can be read in conjunction with this guide. It contains a much fuller description of functionality, together with a complete list of available programming methods and events.

This document should be instead regarded as an overview, a partial technical introduction, and a collection of hints and tips – pointers to good application design practice.

In brief, the objective of this guide is to:

- Provide a brief introduction to OPC
- Describe how the CX-Server OPC product works, and how it relates to the OPC specifications generally
- Provide some guidelines and hints and tips for developing OPC applications using CX-Server OPC, including advising on how to achieve best performance
- Provide brief examples of how to use Omron OPC functionality in Visual Basic, Excel and C++

1.2 Who should read this document?

This guide should be read by application engineers and others interested in using CX-Server OPC software to interface to Omron hardware or software.

The target audience for this guide is application developers knowledgeable in industrial automation technology and one or more of Visual Basic, Visual Basic for Applications (as used in Excel) and C++. No attempt is made to teach automation technology and terminology (e.g. SCADA) or any programming language.

Full knowledge of OPC is not required in order to read this guide. However, all OPC developers should have, and use, a separate OPC reference (e.g. as provided by the OPC Foundation).

2. An Overview of OPC

2.1 A Brief History of OPC Data Access

In the early 1990s a group of people from several important companies involved in the creation of SCADA (Supervisory Control and Data Acquisition) systems began meeting at Microsoft's headquarters in Redmond. Their interests focused on the use of the Windows operating system within the factory automation environment, in particular for the acquisition of real time data. Microsoft at the time were working on the development of OLE 2.0 (Object Linking and Embedding) and it was apparent that this new technology would replace that of DDE (Dynamic Data Exchange) which up until that point had been used extensively for data exchange within SCADA systems designed for Windows. The new OLE technology was more flexible, robust and efficient than DDE.

OLE provided an opportunity to create a standard interface between the SCADA core and the device drivers responsible for reading and writing data to various automation devices such as PLCs. Such a standard interface would benefit both the SCADA vendors and equipment suppliers as the SCADA vendors would not need to invest costly effort in developing software drivers, while the equipment manufacturers could provide just one driver that would work with all Windows software – in the same way that printer manufacturers already could.

The first draft of the OPC (OLE for Process Control) specification v1.0 was released in December 1995. The following year, the group of companies involved in the definition of the standard decided that an independent body must be set up to manage the OPC specification. This decision resulted in the formation of the OPC Foundation that has continued to develop the philosophy of standardised interfaces for SCADA.

In 1998 the Data Access 2.0 specification for OPC was released. This addressed several deficiencies and ambiguities in the original standard and included specifications for both the Automation interfaces (typically used by VB programmers) and Custom interfaces (typically used by C++ programmers).

In 2000, using these DA 2.0 specifications Omron created the first version of CX-Server OPC, which provides Client and Server software for the CX-Automation Suite software range.

The current version of Omron's CX-Server OPC is compliant with version 2.05 of the Data Access specification.

2.2 Other OPC Specifications

Since the first OPC Data Access specification was produced in 1995 the OPC Foundation have addressed a number of additional areas of control and automation normally associated with SCADA. The original specification for Data Access is now just part of a whole series of specifications that include such areas as Alarms/Events and Batch control, although the DA interface is still far more commonly supported than the others.

The figure below shows some of the current areas covered by OPC Interface Specifications.

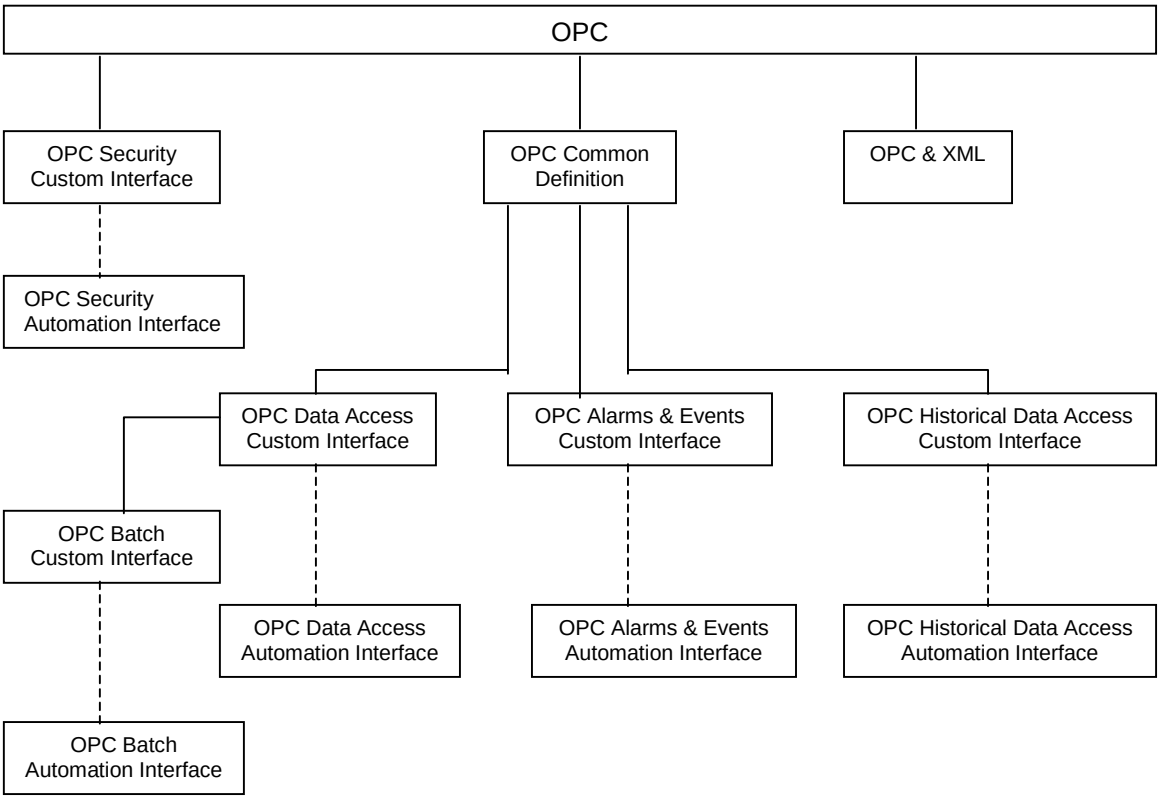


Figure 1: OPC Interface Specifications

2.3 Key Technologies used by OPC

This section provides, for convenience, a brief introduction to some key technologies that are used by, or that form part of, OPC. Some of these are described in more detail in the appendices; plentiful reference books are available elsewhere describing all of them in any required level of detail.

2.3.1 A Brief Introduction to DCOM

Microsoft describes DCOM as

> "The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Previously called "Network OLE," DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP. DCOM is based on the Open Software Foundation's DCE-RPC spec and will work with both Java applets and ActiveX® components through its use of the Component Object Model (COM)."

In other words, DCOM is an object-programming model for the implementation of distributed applications using a client server pattern. A client can use several servers at the same time and a server can provide functionality to multiple clients simultaneously.

Translating that into plainer English, COM basically allows software components to be written in such a way that they can be used by all COM-aware applications (e.g. C++, later versions of Visual Basic) without those applications needing to know anything about the "internals" of the object. DCOM is simply the distributed version of COM – i.e. the objects can be spread across a network. It is a very powerful system, although some machine-level and security configuration may be required to allow it to work correctly and reliably.

Central to the capability of a DCOM object are its interfaces. **All** communication with a DCOM object occurs through its interfaces – an interface is said to provide a "contract" (a full and unchanging description) for the functionality provided by that object. Each interface has a unique ID and describes a group of related methods. The description of the interface defines the syntax and the semantics of the services provided by that interface – the internal implementation of those services doesn't matter to the calling applications.

In the case of OPC these interfaces for each of the DCOM objects are defined within the relevant specifications. This guide only deals with the OPC Data Access specification Custom (used by C++) and Automation (used by script languages and VB) interfaces.

2.3.2 A Brief Introduction to Custom and Automation Interfaces

Custom and Automation interfaces define different methods of connecting client and server objects together, and they are essentially programming language dependent. Custom interfaces are used in programming languages such as C and C++ that support the concept of function pointers, whilst languages that do not, such as Visual Basic and the VBA script language (included in products such as Microsoft Excel) require the ability to call functions by name and therefore use an alternative interface called Automation. Basically, connections between custom (C++) objects are made at compile time, whereas those between Automation (script) objects are made at runtime.

Automation was originally developed as a way for applications (such as Word and Excel) to expose their functionality to other applications, including scripting languages. The intention was to provide a simple way to access properties and call methods that put as little strain on the Automation client as possible, allowing calls to be made at runtime without needing data type information about the object being accessed. Custom interfaces require the compiler to resolve links to objects at compilation time in order to access the methods supported by the object. Basically, Automation interfaces are often easier but less efficient, Custom interfaces may be more complicated to use but may offer greater performance and more flexibility.

The OPC Foundation provides for a specification for both interfaces allowing developers to connect to servers either way, via the Custom interface or via the Automation interface.

In fact, in the case of OPC, the Automation interface is simply a wrapper around the Custom interface – calls are converted from the Automation interface format and passed on to the Custom interface.

This is shown by the following diagram:

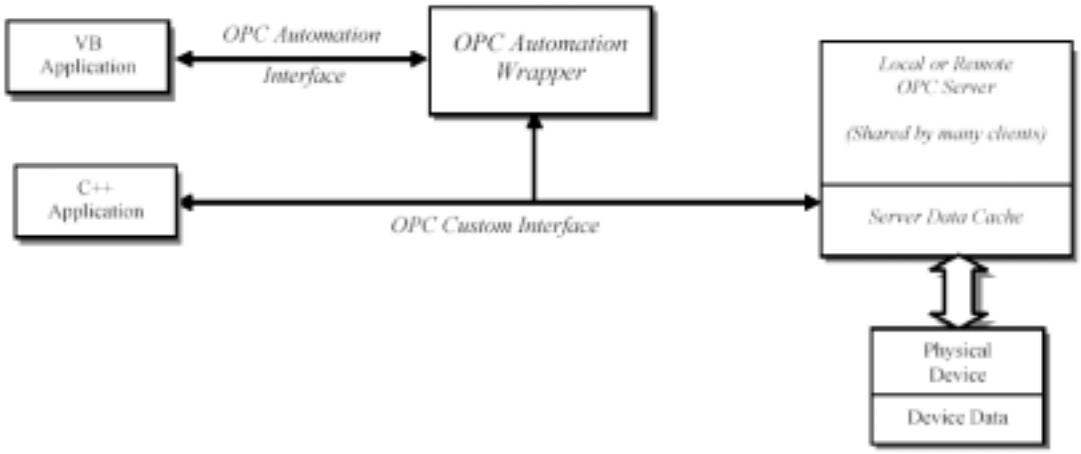


Figure 2: Custom and Automation Interfaces

3. An Overview of CX-Server OPC

3.1 Introduction to CX-Server OPC

CX-Server OPC provides full OPC Data Access client and server support. It therefore provides an easy way of linking Omron hardware with third-party software. It can also be used to link Omron (and indeed any) software that supports ActiveX or programming language interfaces with third-party hardware.

3.2 OPC Standards Supported

CX-Server OPC fully supports the OPC Data Access 2.05 interface. The OPC DA 1.0a interface is also supported by the OPC Server for backwards compatibility reasons, but, where there is a choice, use of the 2.05 interface is recommended. The OPC Client only makes use of the OPC DA 2.05 interface- it neither requires, nor uses, the DA 1.0a interface.

All mandatory and optional Custom and Automation interfaces are supported.

This full support has been verified using version 2.0 of the official OLE Foundation Compliance Test software.

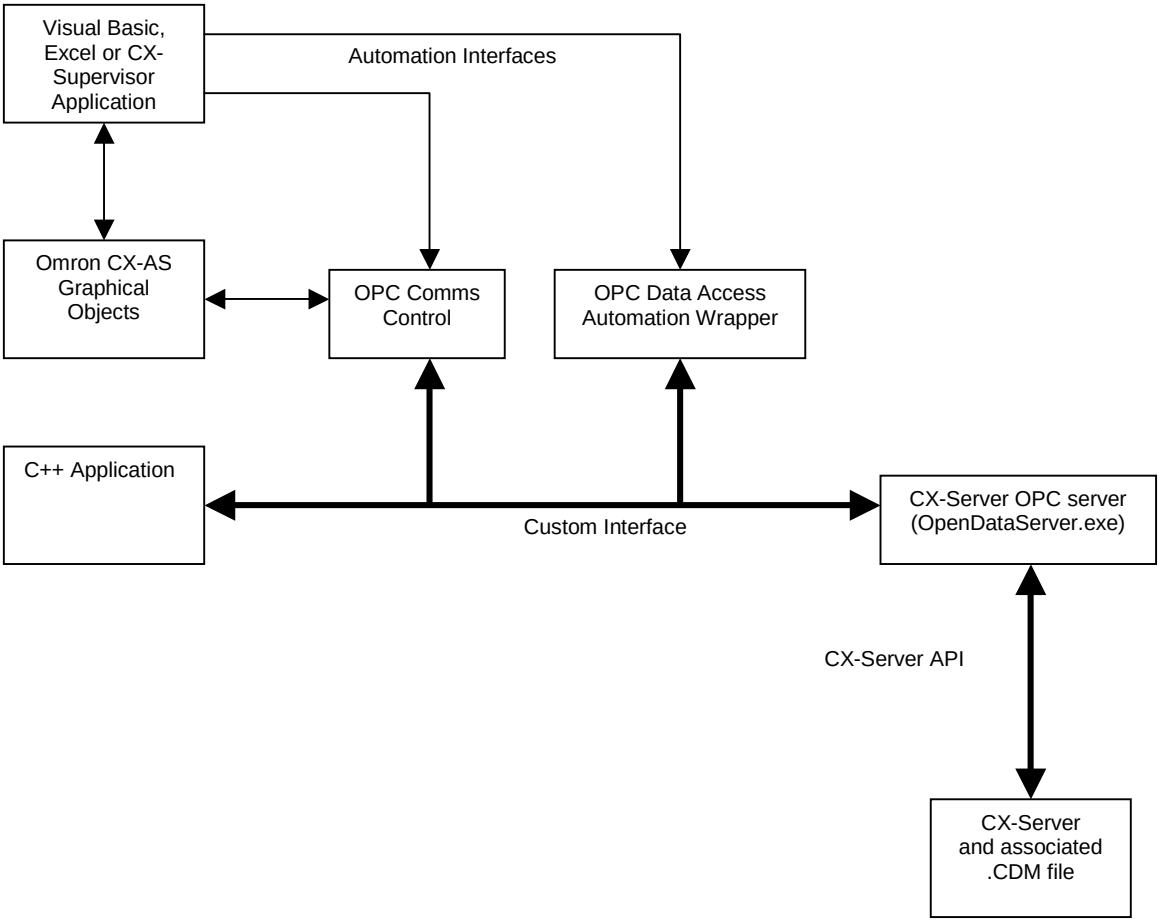
3.3 Relationship with other CX-Automation Suite Products

CX-Server OPC forms an important part of CX-Automation Suite. It provides an OPC interface to the CX-Server device-connection middleware that is used for CX-Automation Suite products.

CX-Server OPC directly uses CX-Server data files (.CDM files), which can be created by many other CX Automation products, including CX-Supervisor, CX-Programmer and CX-Server Lite.
CX-Server also provides an editor and other utilities for creating PLC configurations and symbolic data. These tools are provided as part of CX-Server OPC.

The following diagram illustrates the relationship between CX-Server OPC and other CX Automation Suite products:

Figure 3: Relationship of CX-Automation Suite to OPC



4. The OPC Data Access Server

4.1 Purpose and versions

The OPC Data Access Specification is usually regarded as the “main” OPC specification, to the extent that saying “an application supports OPC” is usually the same as saying “an application supports the OPC Data Access Specification. It is this specification that governs the data connection between factory-floor devices and SCADA software.

The OPC DA specification exists in three main versions, 1.0a, 2.0 and 3.0. Of these three, version 2.0 is now by far the most common, version 1.0a is still supported for backwards-compatibility reasons by many older servers, and version 3.0 is new and not yet widely adopted.

4.2 The structure of the Data Access Server

The following information, taken from the OPC specifications, provides a brief introduction to the structure of the OPC DA v2.0 Server:

At a high level, an OPC Data Access Server is comprised of several objects: the server, the group, and the item. The OPC server object maintains information about the server and serves as a container for OPC group objects. The OPC group object maintains information about itself and provides the mechanism for containing and logically organizing OPC items.

The OPC Groups provide a way for clients to organize data. For example, the group might represent items in a particular operator display or report. Data can be read and written. Exception based connections can also be created between the client and the items in the group and can be enabled and disabled as needed. An OPC client can configure the rate that an OPC server should provide the data changes to the OPC client.

There are two types of groups, public and local (or ‘private’). Public is for sharing across multiple clients, local is local to a client. Refer to the section on public groups for the intent, purpose, and functionality and for further details. There are also specific optional interfaces for the public groups.

Within each Group the client can define one or more OPC Items.

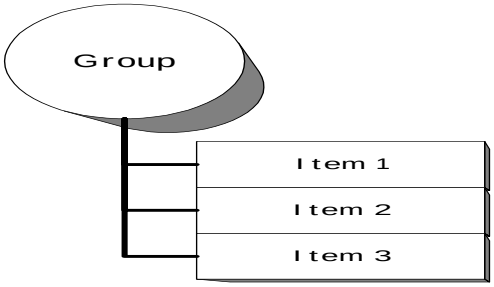


Figure 4: Group/Item Relationship

The OPC Items represent connections to data sources within the server. An OPC Item, from the custom interface perspective, is not accessible as an object by an OPC Client. Therefore, there is no external interface defined for an OPC Item. All access to OPC Items is via an OPC Group object that “contains” the OPC item, or simply where the OPC Item is defined.

Associated with each item is a Value, Quality and Time Stamp. The value is in the form of a VARIANT, and the Quality is similar to that specified by Fieldbus.

Note that the items are not the data sources - they are just connections to them. For example, the tags in a DCS system exist regardless of whether an OPC client is currently

accessing them. The OPC Item should be thought of as simply specifying the address of the data, not as the actual physical source of the data that the address references.

See section 4.3.2 for a diagram showing the general structure of an OPC (Automation) Data Server

4.3 The structure of CX-Server OPC

The CX-Server OPC Data Access Server (called “Omron Open Data Server”) is implemented within the main CX-Server OPC executable file, which is called OpenDataServer.exe. The diagram in section 3.3 shows the relationship between this server and other Omron applications.

The name space (the tag or “point” information) for the server is supplied from a CX-Server data file (.CDM file). This file is created using the CX-Server Point Editor included as part of CX-Server OPC, or by other CX Automation Suite products such as CX-Programmer.

4.3.1 CX-Server OPC Client Communications Control

The CX-Server OPC Client Communications Control is an ActiveX control that can be embedded within any suitable ActiveX container such as Visual Basic, Excel and CX-Supervisor.

This OPC Client Control provides a simplified high-level interface to **any** OPC Server. The interface to this control is Omron specific; the control then automatically drives the OPC Server using the standard OPC interfaces.

It provides quick-to-learn (configuration is mostly menu and dialog driven) and easy access to the server from script languages and also automatically acts as a link to the Omron Graphical controls (toggle button, gauges, etc.) and to CX-Supervisor. In other words, the graphical controls and CX-Supervisor can detect its presence on a VB, Excel or CX-Supervisor form or page, and the user can then configure data connections to it from them via menus and dialogs, with no script required. This control is also used in hidden embedded form within the CX-Supervisor Point Editor to provide OPC connectivity within that product.

4.3.2 CX-Server OPC Server Automation Wrapper

The wrapper supplied with CX-Server OPC provides full and standard OPC DA 2.0 Automation Interface support.

The CX-Server OPC Server Automation Wrapper can be used within Visual Basic and script languages such as VBA (as used in Microsoft Excel).

It is called a “wrapper” because it wraps the custom interface (in other words, all calls to the Automation Wrapper are actually converted into calls to the custom interface before they reach the OPC Server.)

The following information (taken from the OPC Foundation Data Access 2.0 Automation Interface Specification) summarises the OPC Automation Server functionality:

OPC Automation Server Object Model

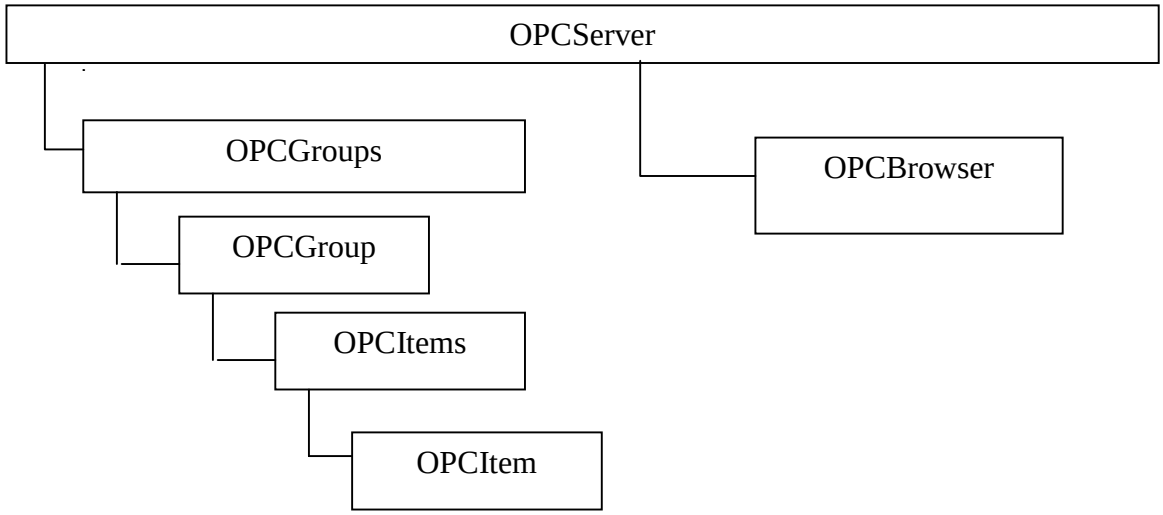


Figure 5. OPC Object Hierarchy

Object	Description
OPCServer	An instance of an OPC Server. You must create an OPCServer object before you can get references to other objects. It contains the OPCGroups Collection and creates OPCBrowser objects.
OPCGroups	An Automation collection containing all of the OPCGroup objects this client has created within the scope of the OPCServer that the Automation Application has connected to via the OPCServer.Connect()
OPCGroup	An instance of an OPCGroup object. The purpose of this object is to maintain state information and provide the mechanism to provide data acquisition services for the OPCItem Collection object that the OPCGroup object references.
OPCItems	An Automation collection containing all of the OPCItem objects this client has created within the scope of the OPCServer, and corresponding OPCGroup object that the Automation Application has created.
OPCItem	An automation object that maintains the item's definition, current value, status information, last update time. Note the Custom Interface does not provide a separate Item Object.
OPCBrowser	An object that browses item names in the server's configuration. There exists only one instance of an OPCBrowser object per instance of an OPC Server object.

4.3.3 CX-Server OPC Server Custom Interface

CX-Server OPC provides full and standard support for the OPC DA 2.0 Custom Interface.

The OPC Custom Interface is accessed via advanced programming languages such as C++.

Through the custom interface there are two COM objects, **OPCServer** and **OPCGroup.** It is their associated interfaces that provide for full control over the server. A description of the interfaces available on each of these objects is available in the OPC Foundation Data Access 2.0 Custom Interface Specification.

The following list taken from that specification provides a *very brief* summary of the available interfaces. *(Note: There is little point in including too much of this information here, as using the OPC custom interface is a complex task that requires full use of that (large) OPC Custom Interface Specification.)*

Available OPC Custom Objects and interfaces are as follows:

OPCServer
- **IOPCServer**
- **IOPCServerPublicGroups (optional)**
- **IOPCBrowseServerAddressSpace (optional)**
- **IOPCItemProperties (new 2.0)**
- **IConnectionPointContainer (new 2.0)**
- **IOPCCommon (new 2.0)**
- **IPersistFile (optional)**

OPCGroup
- **IOPCGroupStateMgt**
- **IOPCPublicGroupStateMgt (optional)**
- **IOPCASyncIO2 (new 2.0)**
- **IOPCAsyncIO (obsolete - V1)**
- **IOPCItemMgt**
- **IConnectionPointContainer (new 2.0)**
- **IOPCSyncIO**
- **IDataObject (obsolete - V1)**

EnumOPCItemAttributes
- **IEnumOPCItemAttributes**

4.4 Server initialisation and address space

When the CX-Server OPC Server is invoked it initialises its address space (list of configured points) from the associated CX-Server (CDM) data file. Clients can then browse this address space and request information about the points available for access within the server.

If the “serial-only” version of CX-Server OPC is being used (i.e. a “serial-only” license number has been entered) then only point information for PLCs connected via a serial connection will be loaded and available for browsing.

The data file is normally selected directly from the main CX-Server OPC menu. Without a CX-Server data file loaded the CX-Server OPC server cannot operate (i.e. no connections to devices are available).

4.5 The Concept of Items and their attributes with relation to Groups

In order to access a point a client must first make a request to the server to create a group and item that refers to the designated point. Each group has a set of attributes that include its name, active state and rate of subscription (see the OPC specification for details). A point within the server address space may be referenced by any number of items in any number of groups and by numerous clients as depicted in the diagram below.

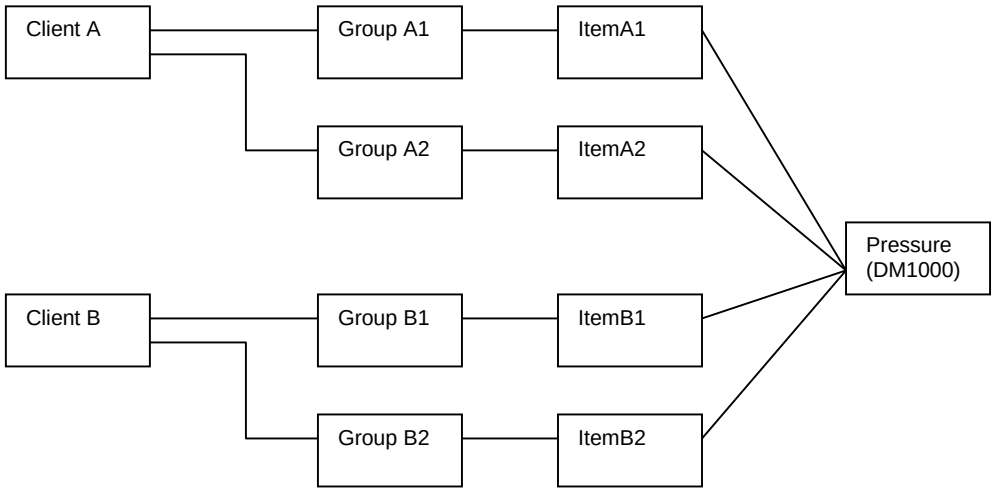


Figure 6: Relationship of Items to Points

The groups and items are client-specific – each client records its own configuration data in its own data file. In the above diagram only the information about the CX-Server point "Pressure" is stored in the OPC server data file (the CX-Server .CDM data file in the case of CX-Server OPC). The client stores all the other information. (The CX-Server OPC client control stores its own information in standard XML files with the suffix .OPC).

The OPC Server supports an interface for public groups. In theory, these allow more than one client to refer to the same group. However, use of this facility is rare and should be discouraged since the OPC Foundation has made this functionality obsolete in the Data Access 3.0 specification.

4.6 Reading and Writing data to the server.

To appreciate the best ways to read or write data from devices, it is important that the meaning of some of the most commonly used terms is understood. These include:

- Synchronous
- Asynchronous
- Cache
- Device

There are three methods of obtaining the current value and quality of an item. These are:

- Subscription
- Synchronous Read
- Asynchronous Read

When creating an application using an OPC Server it's important to consider during operation what is the most efficient type of communication to use. For example there is no point

subscribing (asking for data to be obtained at a specified interval) to a group of items if an application only requires them once on initialisation. In such cases it is better to create the group as inactive and read the values once using a synchronous read call to the server, and then remove the group and associated items.

4.6.1 Synchronous or Asynchronous?

It is important to be aware of the distinction between synchronous and asynchronous communications.

4.6.1.1. Synchronous Communication

Synchronous communication is when the client requests that the server carry out a particular task and the thread of execution of that task passes from the client to the server. In other words, the client waits for the server to service the allotted task and on completion execution returns to the client at the point of return from the function call. In this scenario the client will have to wait as long as the server takes to complete the task. Basically, nothing happens in the client while it is waiting for the server to complete the operation.

Examples of use (In Excel):

```
Value = OPCComms1.Read(Group, Item, ReadFromDevice)
Value = OPCComms1.Read(Group, Item, ReadFromCache)
Value = OPCComms1.Read(Group, Item, ReadFromCacheOrDevice)
OPCComms1.Write Group, Item, Value, WaitUntilComplete
```

4.6.1.2 Asynchronous Communication

With asynchronous communication the client initiates a request of the server, and then continues its execution separately while the server carries out the requested task. On completion of the task the server interrupts the client (an event in VB) and supplies the client with the requested data. The time taken between the request and the answer from the server is undetermined. Basically, the client makes a request and then carries on operation; when the server is ready it calls the client back with the finished result of the operation.

Examples of use (In Excel):

```
      OPCComms1.Read  Group, Item, ReadFromDeviceAsync
OPCComms1.Write  Group, Item, Value, NoWaiting
```

4.6.2 Cache or Device?

The interface on the OPC server supports two types of synchronous reads - either from cache or from device. Synchronous read behaviour differs according to whether CACHE or DEVICE is specified as a parameter to the OPC Server.

4.6.2.1 Reading from Cache

If CACHE is specified then the last value known to the OPC Server will immediately be returned - no attempt will be made to read the current value from the device. The data will be valid only if both the group and item are active, i.e. the item is being subscribed (read at a regular interval from the device).

Because there is no requirement for the OPC Server to actually read the value again from the device, it is many times faster to read values from cache, and this is recommended whenever it is not essential for the value to be fully up to date.

The CX-Server OPC Communications Control contains an important optimisation in that, if requested, it will automatically ask the OPC Server to read the value from the device if it knows that the point is not currently active and therefore will not be available in cache.

Example of use (In Excel):

```
Value = OPCComms1.Read(Group, Item, ReadFromCacheOrDevice)
```

This will cause the OPC Communications control to check to see if the item is active, and if it is it will ask the server to read the item from cache. If the item is not active (and therefore does not exist in cache) then the OPC Server will be asked to read the item from device.

4.6.2.2 Reading from Device

If DEVICE is specified the active state of the group and item are ignored. The actual memory address for the point is always read from the device, and only when the data is obtained is it returned to the client.

Note: Some OPC Servers behave differently, returning data on some occasions from the cache even if DEVICE is specified. This behaviour can make the OPC server appear significantly faster, but means that the data obtained *cannot* be relied on. In some cases the data returned is wrong, either out of date or even data not yet written to the device (cached from a write operation that has not yet taken place). In order to guarantee correct data is returned, if DEVICE is specified, CX-Server OPC *always* reads the data from the device before returning it to the calling client. If cached data is preferred, it is necessary to use the ReadFromCache or ReadFromCacheOrDevice parameter instead.

Example of use (In Excel):

```
Value = OPCComms1.Read(Group, Item, ReadFromDevice)
```

4.6.3 Subscription

This is the normal/default operation of an OPC server. A client requests that the server creates a named group and then adds to that group a list of named items that correspond to points available within the server's address space (e.g. within a CX-Server .CDM data file). The client can request that the group is created with a specific update rate. In general this means that the server will send the value, quality and timestamp of each item in the group back to the client on a regular basis. However, exact timing depends on a number of factors. Most OPC servers including CX-Server OPC support a limited number of update rates. The server will return to the client the actual update rate closest to that requested.

CX-Server OPC supports the following update rates

100, 500, 1000, 2000, 5000, 10000, 60000, 120000 milliseconds

The data associated with an item will only be returned to the client if the value of that item has actually changed during the update rate period. Therefore if a group has an update rate of 1000ms and none of the items change during that time frame the client will not receive any data about the items. If the group is inactive the client will also not be informed of any changes in the values of items within the group. If the group is active then only items that are active and have changed in value during the time frame will be returned to the client.

During normal operation most clients simply request the creation of a group with a specified update rate and respond to changes in values of the items within the group. Subscription places the least stress upon the server and therefore should be the main method employed for receiving data from a server.

The current value, quality and timestamp associated with point that is subscribed to are cached within the server, and can be subsequently obtained using a synchronous read from cache.

The active flag associated with a group can be used effectively to control when data for a particular set of points needs to be retrieved on regular basis. This might occur within a SCADA package when a particular page is on display (i.e. the group is set active when the page is loaded, and inactive when the page is closed).

Examples of use (In Excel):

```
      OPCComms1.GetData Group, Item, Continuous
```

```
OPCComms1.GetData Group, Item, OnChange
```

Note: The Continuous flag indicates that data will be sent back from the control (an event generated) at the configured interval even if it has not changed, whereas the use of OnChange indicates that it will only be sent back if the value has changed. However, in practise this usually makes little difference, as most OPC servers (including CX-Server OPC v1.2) only return a value to the client if it has changed.

4.6.4 Synchronous Read

The client waits while a synchronous read operation occurs. A synchronous read operation can be carried out on a list of items from within a group. The interface on the OPC server supports two types of synchronous reads - either from cache or from device. See Cache or Device On completion of a read from device the cache used by synchronous read from cache is updated.

Examples of use (In Excel):

```
Value = OPCComms1.Read(Group, Item, ReadFromCache)
Value = OPCComms1.Read(Group, Item, ReadFromDevice)
Value = OPCComms1.Read(Group, Item, ReadFromCacheOrDevice)
```

The ReadFromCacheOrDevice option, incidentally, will read from cache if possible, otherwise it will read from device.

4.6.5 Asynchronous Read

Values from asynchronous reads are returned to the client via event callbacks – the client continues operation while the server is obtaining the new value. Values are *always* read from the device and are not affected by the active state of the group or item. When the new value is available, the server calls the client using a callback (sends an event to the client).

Example of use (In Excel):

```
OPCComms1.Read Group, Item, ReadFromDeviceAsync
```

4.6.6 Synchronous Write

The synchronous write call is one of two methods for writing data to the device. The function does not return to the calling application until confirmation is received that the data has been written to the device, at which point the cache used by synchronous read is updated. Synchronous write is not affected by the current state of the group or item.

Example of use (In Excel):

```
OPCComms1.Write Group, Item, Value, WaitUntilComplete
```

4.6.7 Asynchronous Write

This method involves instructing the server to write data to a device, but not waiting until the write is complete. Execution returns to the client immediately, before the data is written. As with the synchronous write the active state of the group and item are ignored. On completion of the write the cache used by synchronous read is updated and the server returns confirmation of the write to the client using a callback (sends an event to the client).

Example of use (In Excel):

```
OPCComms1.Write Group, Item, Value, NoWaiting
```

4.7 The CX-Server Communications Utilities

CX-Server OPC is supplied with CX-Server. This powerful communications middleware includes a set of utilities typically used for configuring or monitoring PLC networks, e.g. as part of a CX-Programmer application. These utilities can also be used with CX-Server OPC. They are accessed from the “Communications Utilities” submenu that is obtained by right

clicking on the CX-Server OPC icon that is present in the Windows task bar whenever CX-Server OPC is running.

Most of these utilities are only available when CX-Server OPC is connected to a PLC (i.e. when a client is connected), in which case a PLC selection dialog is shown before the utility is launched. The utility will then automatically connect to the PLC.

The Network Configuration Tool and Performance Monitor Tool are always available. The Network Configuration Tool is particularly powerful, including options to scan serial ports for PLCs, or to check communications with a PLC, go online and configure it.

For full details of these utilities consult the CX-Server help files.

Note: If access to these powerful utilities is not required, and no other program that requires them (e.g. CX-Programmer, CX-Supervisor) is being used on a machine, then they can be removed by uninstalling CX-Server PLC Tools from the Windows Control Panel.

4.8 Using CX-Server OPC offline

CX-Server OPC can be used offline (not connected to a PLC). To use this mode of operation set the “local values” option on the “Server Info” dialog (accessed by right-clicking with the mouse on the CX-Server OPC icon in the taskbar that is present whenever CX-Server OPC is running.) The OPC Server should be closed and restarted after setting or unsetting this option to ensure correct operation.

4.9 Controlling CX-Server Input Optimisations

The Omron CX-Server middleware contains optimisations that improve communications throughput and which are enabled by default. In certain specific circumstances it may be best to disable the CX-Server input optimisations: if a program is running in the PLC which changes a memory location that is being written by CX-Server very soon after the time that CX-Server writes it, then there is a theoretical risk that the change of value by the PLC program may be missed by a current subscription to the same location. (Note: any new read attempt from device, e.g. a synchronous read from device, would return the correct value, although a read from cache might not.) Disabling the CX-Server input optimisations will ensure that the correct value is always obtained, but will slow down communications, although this is unlikely to be noticeable except in the circumstances where there are subscriptions to large numbers of points that are rapidly changing value.

5. Client Connection to a Server

5.1 Using the OPC Client Communications Control

The easiest method of connection to OPC servers is to use the CX-Server OPC Client Communications Control. This is an OPC client control that provides a simplified high-level interface to **any** OPC Server. The interface to this control is Omron specific; the control then automatically drives the OPC Server using the standard OPC interfaces.

It provides quick-to-learn (configuration is mostly menu and dialog driven) and easy access to the server from script languages and also automatically acts as a link to the Omron Graphical Controls (toggle button, gauges, etc.) and to CX-Supervisor. In other words, the graphical controls and CX-Supervisor can detect its presence on a VB, Excel or CX-Supervisor form or page, and the user can then configure data connections to it from them via menus and dialogs, with no script required. This control is also used in hidden embedded form within the CX-Supervisor Point Editor to provide OPC connectivity within that product.

This method is most appropriate for small to medium applications created in Visual Basic or Excel since it provides fairly limited control over the server, and data throughput is restricted to a single event to the application utilising the control. It can also be used in Visual C++, and an example is provided below. For medium to large applications use of the Automation Wrapper or Custom Interface will typically be more efficient.

5.1.1 Examples of use of OPC Client Communications Control

The following provides some examples of the use of OPC Client Communications Control functionality in Excel, Visual Basic and Visual C++

5.1.1.1 Step by Step OPC Client example using Excel

1. After installation, start Microsoft Excel.

2. *Either* a) If the toolbar showing Omron graphical components is visible then click on the button for the **Add OPC Communications Control** *or* b) ensure the Control Toolbox is displayed, by selecting Toolbars on the View menu and then click the button labelled **More Controls** (the one with the hammer and spanner symbol) then select **OMRON CX OPC Communications Control** from the list of all ActiveX controls. In either case, drag a square on the Worksheet to create the object.

3. Click the Communications Control with the right mouse button, to popup the context menu. Select Properties from the **OMRON CX OPC Communications Control Object** menu to bring up the properties dialog. The Computer Name and Server Name can now be configured. For now, leave the Computer name at the default value (the current machine name), and set the OPC Server to **OMRON.OpenDataServer.1** if it is not already selected.

4. You now need to specify a client project file name - use the **New** or **Open** buttons as appropriate to create or open one. The OPC client connection definitions will be stored in this file (Note: this is basically a list of which points the client is interested in, and is not the same as a *server* project file, which is used to configure the underlying PLC data).

5. Click on the **Groups** tab to add an OPC client group, and specify an update rate in milliseconds (e.g. 1000)

6. Click on the Items tab to add an OPC item. Select a group and then the **Add** button. Now add the OPC item - type in a name and then use the browse button to browse the Server. Select a point from the lower portion of the workspace browser and hit **OK**. The details of the point will now be filled in on the **Item Attributes** dialog, which can be confirmed by the **OK** button. Click the main OK to complete communications configuration

7. Either a) If the toolbar showing Omron graphical components is visible then click on the button for the **Add Linear Gauge Control** or b) Click the button labelled **More Controls** again, and select the **OMRON CX Linear Gauge Control** from the list. In either case, drag a square on the worksheet to create the object

8. Click the graphical control with the right mouse button, to popup the context menu. Select Properties from the **OMRON CX Linear Gauge Control** Object menu. Click the **Data Source** tab and select the required item (OPC Point) from the list, and click **OK**. Setup is now complete!

9. To connect to the OPC Server and show the data, click the button labelled **Exit Design Mode** on the Control Toolbox. The data will be displayed,

5.1.1.2 Step by Step Script example using Excel

1. Start Excel, and, using the steps outlined in the previous Excel example, select an Omron CX Graphical Control (e.g. a Linear Gauge) from the Control Toolbox and drag it onto the worksheet.

2. Add a standard Excel command button (using the button symbol on the Control Toolbox) and double click on it to bring up the Excel VBA Editor.

3. Type the name of the CX Graphical Control (e.g. Gauge1) 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. The line of VBA script now reads something like **Gauge1.Value = 10** (Tip: because Value is the default property for a graphical control it can optionally be omitted - **Gauge1 = 10** will also work and saves some typing)

4. To run the application, click the button labelled **Exit Design Mode** on the Control Toolbox

5. To test your script, press the command button. The gauge will change to display the value that you have set.

5.1.1.3 Step by Step OPC Client example using Visual Basic

1. Start Visual Basic. *(Note: the examples in this section use Visual Basic 5.0, when using other versions the instructions may have to be adapted slightly)*

2. Ensure the **Toolbox** is displayed, by selecting **Toolbox** on the **View** menu. Click the **Toolbox** with the right mouse button and select **Components...** from the popup menu. Select the components required e.g. **OMRON CX OPC Communications Control** and **OMRON CX Linear Gauge Control**. Click **OK** to add components to the toolbox

3. Double click the **OMRON CX Communications Control** in the toolbox. Click the Communications Control with the right mouse button, to popup the context menu and select **Properties**. The Computer Name and Server Name can now be configured. For now, leave the Computer Name at the default value (the current machine name), and set the OPC Server to **OMRON.OpenDataServer.1** if it is not already selected

4. You now need to specify a client project file name - use the **New** or **Open** buttons as appropriate to create or open one. The OPC *client* connection definitions will be stored in this file (Note: this is basically a list of which points the client is interested in, and is not the same as a *server* project file, which is used to configure the underlying PLC data).

5. Click on the Groups tab to add an OPC client group, and specify an update rate in milliseconds (e.g. 1000)

6. Click on the Items tab to add an OPC item. Select a group **Group1** and then the **Add** button. Now add the OPC item - type in a name **Item1** and then use the browse button to browse the Server. Select a point from the lower portion of the workspace browser and hit **OK**. The details of the point will now be filled in on the Item Attributes dialog, which can be confirmed by the **OK** button. Click the main **OK** to complete communications configuration

7. Double click the required Graphical component in the toolbox, for example **OMRON CX Linear Gauge Control**. Click the graphical control with the right mouse button, to popup the context menu and select **Properties**. Click the **Data Source** tab and select the required item(OPC Point) from the list, and click **OK**. Setup is now complete!

8. To connect to the PLC and show the data, click the **Start** button

5.1.1.4 Step by Step Script example using Visual Basic (using script with graphical control)

1. Start Visual Basic, and, using the steps outlined in the previous Visual Basic example, add an Omron OPC Communications Control and a CX Graphical Control to a form, and configure them.

2. Using the same procedure as with the Gauge, add a Knob Control to the worksheet and connect it to the same group and point specified before (**Group1** and **Item1**)

3. Add a standard Visual Basic command button (e.g. double click on the icon in the Visual Basic toolbox), and then double click on it to bring up the Visual Basic Editor.

4. Type the name of the CX Knob Graphical Control (e.g. **Knob1**) 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. The line of VB script now reads something like **Knob1.Value = 10** (Tip: because Value is the default property for a graphical control it can optionally be omitted – **Knob1 = 10** will also work and saves some typing)

5. To run the application, click the Visual Basic **Start** button, or select **Start** on the **Run** menu.

6. To test your script, press the command button. The information will be written to the OPC Server (via the Knob control), and the knob and gauge will change within the update rate you have specified (e.g. within 1000 milliseconds) to display the value that you have set. *(Note: Although simple and useful, this example is rather artificial. It is more normal and efficient to drive the OPC Client Control directly from script, as demonstrated by the next example. You may wish to save the worksheet you have just created for use in that example.)*

5.1.1.5 Step by Step Script example using Visual Basic (using script with comms control)

1. Start Visual Basic, and, repeat the steps outlined in the previous Visual Basic example, or load the previously saved worksheet.

2. Add another standard Visual Basic command button (e.g. double click on the icon in the Visual Basic toolbox), and then double click on it to bring up the Visual Basic Editor.

3. Type the name of the OPC Client Comunications Control (e.g. **OPCComms1**) 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 **Write** command (note: you can also just type **w** and the list box will change selection to a point near 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 the Group name (in quotes as we are passing it as a string parameter), a number, e.g. 25 and a final parameter **NoWaiting** which indicates that the data should be written immediately, before control returns to the Excel script. The line of VB script now reads something like

```
OPCComms1.Write "Group1", 25, NoWaiting
```

4. To run the application, click the Visual Basic **Start** button, or select **Start** on the **Run** menu.

5. To test your script, press the new command button. The information will be immediately written to the OPC Server, and the knob and gauge will change within the update rate you have specified (e.g. within 1000 milliseconds) to display the value that you have set.

5.1.1.6 Step by Step Example using Microsoft Visual C++ and MFC

1. Start Visual C++ and generate a new project
a) Select "**MFC AppWizard (exe)"**
b) Select "**Dialog based"** radio button
Do not select any of the OLE options in App Wizard; they will pull in many large pieces of the MFC that you don't care about, making your app bigger and slower. Instead, simply add the header file **"afxole.h"** to your "stdafx.h" file.
c) You must also call the function **AfxOleInit()**, which calls **OleInitialize**() and sets up the MFC's own internal OLE bookkeeping, in your app's **CWinApp::InitInstance()** method.

2. Add an **IDispatch** wrapper class (i.e. an automation wrapper) for the OPC Client Control, which will be generated by Microsoft Visual C++.
a) Go to **Project -> Add to Project -> Components and Controls**
b) Select Registered ActiveX Control folder
c) Insert into the project the component titled **OMRON CX OPC Communications Control**. Two new files will be added on your workspace (cxopcclientcommunicationsctrl.cpp and cxopcclientcommunicationsctrl.h)

3. Insert an OPC Client Communication Control.
a) Go to resources and select the dialog that has been generated.
b) Click on the right hand button of the mouse and select **"Insert ActiveX Control…"**
c) Insert **OMRON CX OPC Communications Control**
d) Configure the properties of the control, selecting a project file and adding groups and items as required.

4. To add an event method, for data returned by **OnData**.
a) Click on the right hand button of the mouse and select Class Wizard
b) Select Message Maps pane and select **IDC_ CXOPCCLIENTCOMMUNICATIONS** from the Object Ids list box.
c) Select **OnData**, which is located in the Messages List Box and click **Add Function…**
The method will be generated in OPCExampleDlg.cpp file.

5.1.1.7 Ideas for optimising performance

As well as those ideas mentioned earlier (e.g. using cache whenever possible), reducing the number of active groups and items could significantly improve performance. To enable or disable the active state of an OPC group use **EnableGroup**, e.g. in Excel use

```
OPCComms1.EnableGroup GroupName, True
```
or
```
OPCComms1.EnableGroup GroupName, False
```

To enable or disable the active state of an OPC item use **EnableItem**, e.g. in Excel use
```
OPCComms1.EnableItem GroupName, True
```
or
```
OPCComms1.EnableItem GroupName, False
```

Note: For fuller and more general information on optimising applications see Section 6 – Implementing an OPC Solution.

5.2 Using the OPC Automation Wrapper

The CX-Server OPC Server Automation Wrapper is less easy to use than the Client Control but more powerful. It is intended for use with script languages and Visual Basic, and provides for considerably greater programmatic control over the server. It is called a "wrapper" because it wraps the custom interface (in other words, all calls to the Automation Wrapper are actually converted into calls to the custom interface before they reach the OPC Server.)

The wrapper supplied with CX-Server OPC provides a full and standard OPC DA 2.0 Automation Interface.

This method is appropriate for most applications created in Visual Basic. It provides good functionality and a high-level of control without requiring understanding or use of advanced programming languages such as C++

5.2.1 Examples of use of OPC Automation Wrapper

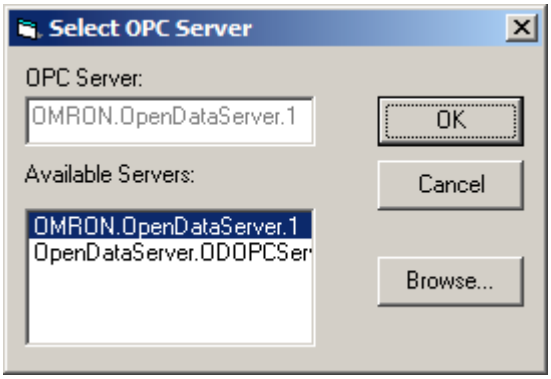
The following provides some simple examples of the use of OPC Automation Wrapper functionality

5.2.1.1 Enabling the OPC Automation Wrapper in Visual Basic 6

1. After creating an application project select Project | References… and check the "**Omron DA OPC Automation 2.0**" option. This allows programmatic use of the OPCServer and its related objects.
2. To see available objects **View | Object Browser** (or press **F2**). Drop down the first (libraries) combo box and select **OPCAutomation**. It is now possible to browse all the objects.

5.2.1.2 Connecting to an OPC Server in Visual Basic 6

Before an OPC server can be used it is necessary to locate one and connect to it. The methods for achieving this are provided by the **OPCServer** object. It is often helpful to display a dialog that allows the user to select and connect to the required OPC server. One possible dialog is shown below:



Server Selection Dialog

The following code can be used in the **Form_Load()** subroutine to initialise the dialog controls:

```
Private Sub Form_Load()
    Dim ServerNames As Variant
    Dim i As Integer
    Set theServer = New OPCServer
    ServerNames = theServer.GetOPCServers
    For i = LBound(ServerNames) To UBound(ServerNames)
        lstServers.AddItem ServerNames(i)
    Next
    lstServers.ListIndex = 0
    txtServer.Text = lstServers.List(0)
End Sub
```

The local PC is searched for available servers by using the **GetOPCServers** method to provide a list of the locally available OPC servers. To list the servers on a remote PC pass the ID of the required PC to the OPCServer object using the same routine.

Having selected the identifier of the server you need to connect to it. The OPCServer object provides a subroutine for doing this called "**Connect**" that takes as its arguments the identifier of the server, and optionally the machine name to attach to a remote computer (the default is to the local PC).

The code to connect when the OK button is selected is very simple:

```
Private Sub cmd_OK_Click()
    theServer.Connect (lstServers.List(lstServers.ListIndex))
    Unload Me
End Sub
```

If the Cancel button is selected it is necessary to destroy the OPCServer object created at the start. This can be done as follows:

```
Private Sub cmdCancel_Click()
    Set theServer = Nothing
    Unload Me
End Sub
```

5.2.2 OPC Automation Wrapper Tutorial

For a full OPC Automation Wrapper Tutorial, see the separate OPC Automation Wrapper Tutorial, which can be accessed from the Examples section of the CX-Server OPC menu, which is accessible from the Windows Start Menu.

5.3 Using the OPC Custom Interface

The most powerful and flexible method for accessing CX-Server OPC involves using the standard OPC DA 2.0 Custom Interface of the CX-Server OPC server directly

Through the custom interface there are two COM objects, OPCServer and OPCGroup. These groups and their associated interfaces provide full control over the server. A detailed description of the interfaces available on each of these objects is available in the OPC DA custom interface specification.

This method is appropriate for medium to large applications created in C++. It provides maximum flexibility and performance, but is also the most complex method, and should only be attempted by proficient programmers.

Programming using the OPC Custom Interface typically makes direct use of source header files provided by the OPC Foundation. For convenience, a copy of these is installed with CX-Server OPC in the **OPC Source Files** subdirectory.

5.3.1 Examples of OPC Custom interface use

The following provides some examples of different aspects of the OPC Custom Interface functionality. All these examples are for Microsoft Visual C++ and assume that the reader is proficient in the use of Visual Studio and MFC.

5.3.1.1 Creating basic project settings:

1. Add the following OPC source files files to the Visual C++, MFC application project: opccomn_i.c, opcda_i.c and opcenum_clsid.c. (Note: for convenience OPC source files are installed by CX-Server OPC into the OPC Source Files subdirectory. The OPC Foundation website may contain more up to date versions.)

2. For each of the files (from step 1 above) make sure that they do *not* use precompiled headers.

3. Add DCOM to the C++ pre-processor definitions settings.

4. In the appropriate headers and includes for: opccomn.h, opcda.h and opcserverdefinitions.h.
   Add the following lines into the appropriate header file:
   ```
   EXTERN_C const CLSID CLSID_OPCServerList;
   EXTERN_C const IID IID_IOPCServerList;
   EXTERN_C const IID IID_IOPCServer;
   ```

5.3.1.2 Initialising the OPC Server:

1. Initialise the COM library with:
   ```
   CoInitialize(NULL);
   ```

2. With the CLSID of the OPC Server create an instance of the OPCServer object using:
   ```
   IOPCServer* m_pOPCServer;
   CoCreateInstance(clsid, NULL, CLSCTX_LOCAL_SERVER, IID_IOPCServer,
   (void**)&m_pOPCServer);
   ```
3. To retrieve a pointer to the **IBrowseAddressSpace** interface:
   ```
   IOPCBrowseServerAddressSpace* m_pBAS;
   m_pOPCServer-
   >QueryInterface(__uuidof(IOPCBrowseServerAddressSpace), (void**)
   &m_pBAS);
   ```

4. To add a group call the **AddGroup** method of the OPCServer object.

5. To add an item:

```
CComQIPtr<IOPCItemMgt> pIIM(m_pUnknown);
pIIM->AddItems(/*Appropriate values in here*/);
```

5.3.1.3 Dealing with data callbacks:

1. Add the following code to the header file:

```
BEGIN_CONNECTION_PART(OPCGroup, OPCDataCallback)
   CONNECTION_IID(IID_IOPCDataCallback)
END_CONNECTION_PART(OPCDataCallback)

 DECLARE_CONNECTION_MAP()

   DECLARE_INTERFACE_MAP()

   BEGIN_INTERFACE_PART(OnDataCallback, IOPCDataCallback)
   STDMETHOD(OnDataChange)(
         /*[in]*/ DWORD dwTransid,
         /*[in]*/ OPCHANDLE hGroup,
         /*[in]*/ HRESULT hrMasterquality,
         /*[in]*/ HRESULT hrMastererror,
         /*[in]*/ DWORD dwCount,
         /*[in, sizeis(dwCount)]*/ OPCHANDLE* phClientItems,
         /*[in, sizeis(dwCount)]*/ VARIANT* pvValues,
         /*[in, sizeis(dwCount)]*/ WORD* pwQualities,
         /*[in, sizeis(dwCount)]*/ FILETIME* pftTimeStamps,
         /*[in, sizeis(dwCount)]*/ HRESULT* pErrors);
   STDMETHOD(OnReadComplete)(
         /*[in]*/ DWORD dwTransid,
         /*[in]*/ OPCHANDLE hGroup,
         /*[in]*/ HRESULT hrMasterquality,
         /*[in]*/ HRESULT hrMastererror,
         /*[in]*/ DWORD dwCount,
         /*[in, sizeis(dwCount)]*/ OPCHANDLE* phClientItems,
         /*[in, sizeis(dwCount)]*/ VARIANT* pvValues,
         /*[in, sizeis(dwCount)]*/ WORD* pwQualities,
         /*[in, sizeis(dwCount)]*/ FILETIME* pftTimeStamps,
         /*[in, sizeis(dwCount)]*/ HRESULT* pErrors);
   STDMETHOD(OnWriteComplete)(
         /*[in]*/ DWORD dwTransid,
         /*[in]*/ OPCHANDLE hGroup,
         /*[in]*/ HRESULT hrMasterError,
         /*[in]*/ DWORD dwCount,
         /*[in, sizeis(dwCount)]*/ OPCHANDLE* phClientItems,
         /*[in, sizeis(dwCount)]*/ HRESULT* pError);
   STDMETHOD(OnCancelComplete)(
         /*[in]*/ DWORD dwTransid,
         /*[in]*/ OPCHANDLE hGroup);
   END_INTERFACE_PART(OnDataCallback)
```

2. In the implementation file add the following code:

```
BEGIN_CONNECTION_MAP(OPCGroup, CCmdTarget)
    CONNECTION_PART(OPCGroup, IID_IOPCDataCallback,
OPCDataCallback)
END_CONNECTION_MAP()

BEGIN_INTERFACE_MAP(OPCGroup, CCmdTarget)
    INTERFACE_PART(OPCGroup, IID_IOPCDataCallback, OnDataCallback)
END_INTERFACE_MAP()
```

3. The compiler will indicate that the methods added must be implemented. This can be done using the standard MFC techniques, for example:

```
HRESULT FAR EXPORT
OPCGroup::XOnDataCallback::OnWriteComplete(DWORD dwTransID,
OPCHANDLE hGroup, HRESULT hrMasterError, DWORD dwCount, OPCHANDLE*
phClientItems, HRESULT* pError)
{
   METHOD_PROLOGUE(OPCGroup, OnDataCallback)

   pThis->m_PerformTime.Finish();
   SetEvent(pThis->m_hAsyncCompleted);

   return S_OK;
}
```

Note: pThis points to the this pointer of the class you used to declare the maps. For details of how to add events into MFC applications, consult the MFC online or internet documentation (e.g. read TN038: MFC/OLE IUnknown Implementation, or look up information on the macros used, e.g. `BEGIN_CONNECTION_MAP` and `BEGIN_INTERFACE_MAP`)

5.3.2 OPC Custom Interface Tutorial

For a full OPC Custom Interface Tutorial, see the separate OPC Custom Interface Tutorial, which can be accessed from the Examples section of the CX-Server OPC menu that is accessible from the Windows Start Menu.

6. Implementing an OPC Solution

6.1 Decide what type of application is required

The first and most important task is to decide exactly what solution is required. This step is essential to achieve a good result.

Important questions include:

- Can the solution be local or is remote access required?
- What are the pre-determined hardware requirements?
- What are the pre-determined software requirements?
- How large is the application?
- What data throughput is required?
- Which programming languages and tools are the application developers familiar with?

The answers to these questions will determine whether it is best to use the OPC Client Communications control, the OPC Automation Wrapper or the OPC Custom interface, or it may be that use of specific OPC client or server software is mandated.

6.2 Design the Application Carefully

When it is clear *what* is required, careful consideration needs to be given to *how* to achieve that solution.

One important consideration is to make sure that the load on the OPC Server is reduced as much as possible. This can be achieve by organising the data carefully, wherever possible arranging similar data items into contiguous blocks that can be read as one operation rather than as individual reads.

6.3 Choose the right client interfaces and technologies to use

For clients, it is important to consider whether the Omron OPC Communications Client should be used, or the Automation Wrapper, or the Custom Interface.

In practise, this decision may depend partly on available programming expertise. As a general rule-of-thumb:

- Use the Communications Client for small to medium applications
- Use the Automation Wrapper for medium to large Visual Basic applications
- Use the Custom interface for medium to large C++ applications
- For large and complex applications, where there is a choice of programming languages available, use the Custom interface and C++.

In this context a “small” application is one where only a few points need to be active at one time, a “medium” application is one where perhaps a hundred points may need to be accessed at a fairly frequent intervals, and a “large” application may involve thousands of active points.

6.4 Choose the correct hardware

Many factors affect the overall performance (and reliability) of a system. Some of these are hardware specification related and obvious but should not be overlooked. Factors include:

- The operating system being used – For maximum reliability and generally good performance **Windows XP Professional** is recommended
- Processor speed – For good performance. processor speed should be **2.0 Ghz or above**
- Memory – The minimum recommended is 256 Megabytes; **512 Megabytes** will sometimes significantly improve important

It is also important to choose the right type of PLC and communications network. Any attempt to run a large application on a low-end PLC using a serial connection is almost certain to lead to serious problems.

6.5 Consider whether to use the Omron Graphical Components

The Omron Graphical Components provide a quick and easy way of displaying data values, particularly in Excel and Visual Basic. However, they should be used with caution, as each component has a single connection through the communications control to an OPC Server, used for transferring one point value at a time. As a result, use of many of these controls on a page or form can be very inefficient.

On occasions it may be more efficient to create a separate link to the OPC server, and then use the graphical controls for visual purposes only, i.e. drive them directly from script rather than connecting them directly to the OPC Communications control.

6.6 Check connections and configuration

The CX-Server PLC Tools provide an excellent means of checking PLC and communications configuration. The CX-Net Network Configuration Tool can be accessed from the main CX-Server OPC Server menu. The option “Scan Serial Ports For PLCs” can be chosen to see if communications with the chosen PLC is possible. A CX-Server project file can then be loaded, or a new one created, and the PLC opened and configured appropriately.

While CX-Server OPC is connected to a PLC (i.e. while a client is connected) all of the PLC tools can be invoked, after first selecting the desired PLC. The PLC Data Monitor Tool can be used to check that the values in the PLC match those expected, or to set them to test values that can be used when debugging the CX-Server OPC application.

Overall, after creating an application, it is important to test it thoroughly, including checking that the correct values are being written to and read from the PLC.

CX-Server OPC also includes the CX Automation Suite Trace Tool, which will display status and error messages. When required, this diagnostics tool should be run before CX-Server OPC is started, and tracing enabled for the CX-Server OPC Server and Client. Note: it is not necessary to run the trace tool during final operation after the application is completed and debugged, as it will slow the system down slightly.

6.7 Optimise performance

In order to make communication speeds as quick as possible, the application should be designed and created with optimisation in mind. An application can be optimised in the following ways:

- Use subscription rather than calling synchronous read at intervals
- Read from cache rather than device whenever possible – it is many times faster
- Points (data locations) being read at the same frequency from a PLC should be in adjacent physical memory locations. This allows CX-Server to perform much better optimisations.
- The CX-Server data file (.CDM file) should be kept as small as possible. Any unused points should be removed – even inactive points can significantly slow down the operation of the system

- Use array points to read blocks of PLC data instead of creating a new point for each PLC address. Hundreds (or even thousands) of PLC addresses (e.g. DM0 to DM1023) can be read with just one point. The Individual PLC addresses can still be referenced using square brackets e.g. PLCOutput[7].

- Bit addresses can also be read in blocks (e.g. if the element size is 64 and the starting address is IR00000 then IR0 bit 0 to IR3 bit 15 would be read).

- If necessary, use group and item activation to control exactly when the I/O points are being updated. For example, all the points on one page could be assigned to the same group; that group could be made active only while the page is on display. For example, if using the Client Communications control then the EnableGroup method could be used, or individual items could be enabled and disable by using the EnableItem method

- Use sensible I/O update rates. Don't use an update rate such as 100 milliseconds unless it is **genuinely** necessary. If you do use it then only use it for small numbers of points. Improper use of this can slow down communications speed enormously.

- Use the CX-Server Performance Tool to provide a rough guide to overall CX-Server Communications loading.

- Follow the normal rules for good PLC performance. Make sure that a suitably powerful PLC is chosen and that the correct communication type is used. As an example, Toolbus should be used in preference to SYSMAC Way, and Ethernet is far faster than any serial communications.

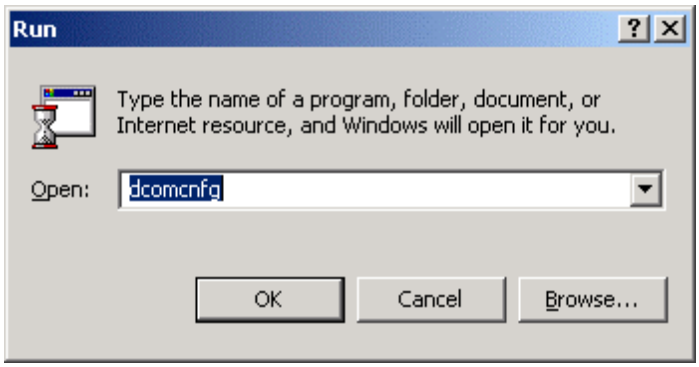
7. How to set up DCOM for Remote Access to Servers

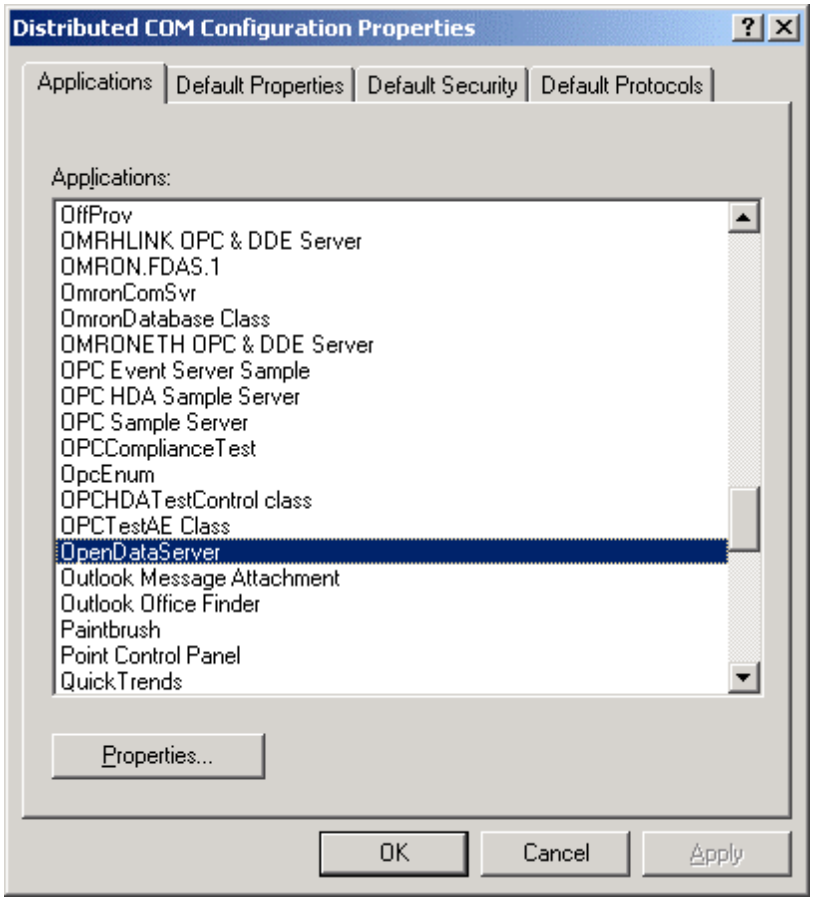
The OPC interface uses a Microsoft technology called DCOM. This allows the OPC client and OPC server to be seamlessly 'distributed' over a PC network. The OPC Server should be running on the PC with direct connection to the PLC or PLC network. However, the OPC Client, or indeed multiple OPC Clients, can be run on different networked PC's and will automatically read and write data over the PC network. To do this, the PC running the OPC Server must be correctly configured. DCOM is very powerful, but also requires careful machine level and application-level configuration. This section provides guidelines on how to set up DCOM for some common OPC Servers (including CX-Server OPC). For fuller details, consult the manuals for the relevant products and Microsoft documentation.

The following is a quick guide:

7.1 Configuring a machine running Windows NT, 2000 and XP

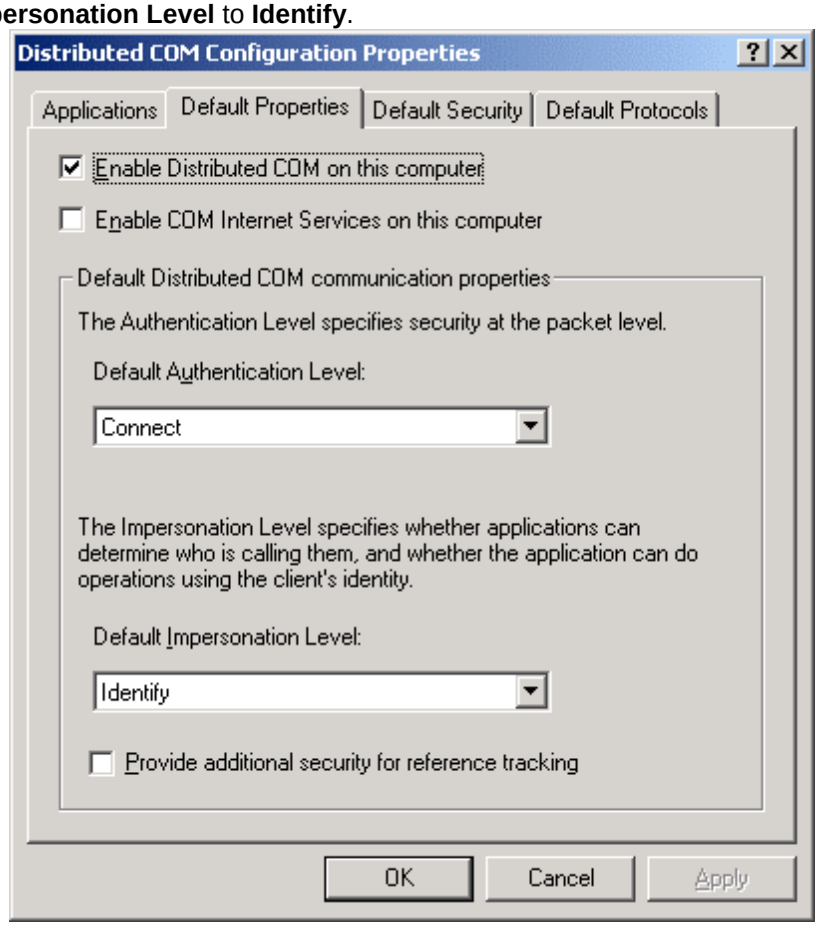
Start DCOMCNFG.EXE e.g. by selecting **RUN** from the Start button. The default location is C:\WINNT\SYSTEM32.





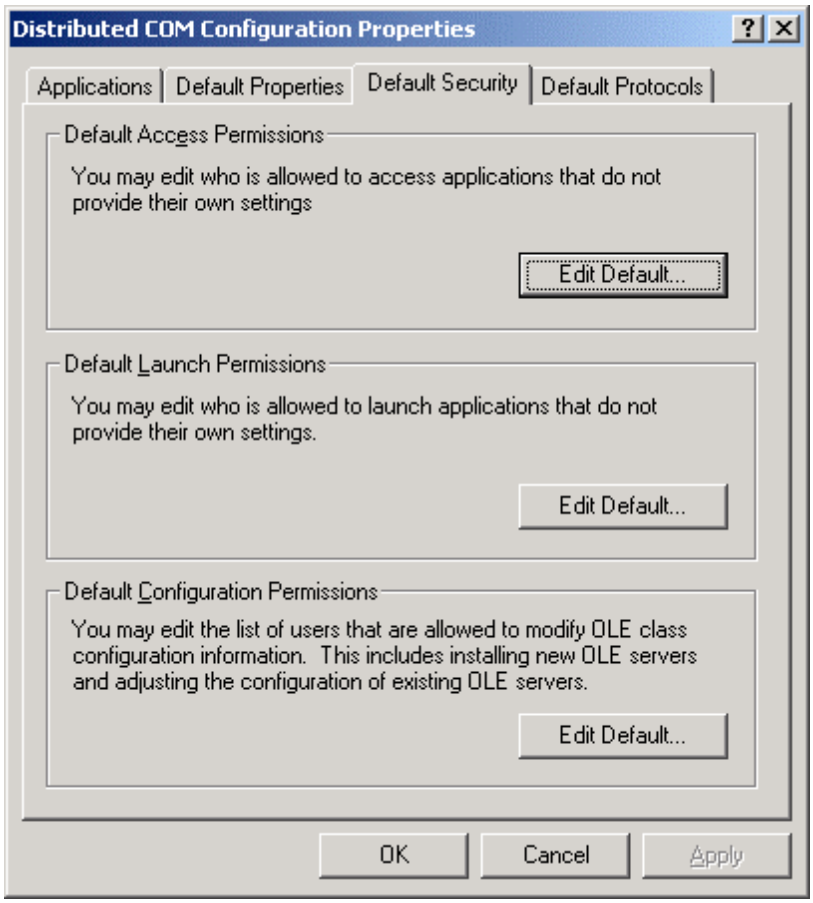
Note: Dialogs shown in this section are NT-specific, and will differ in other operating systems, although the overall functionality is similar. When using XP the Component Services View will appear after DCOMCNFG.EXE is invoked. To access the Default Properties click on 'Component Services' and then on 'Computers' in the treeview, and right click on 'My Computer' and select the 'properties' option on the pop-up menu.

View the **Default Properties** tab. Ensure that the **Enable Distributed COM on this computer** is checked. Configure the **Default Authentication Level** to **Connect** and the **Default Impersonation Level** to **Identify**.



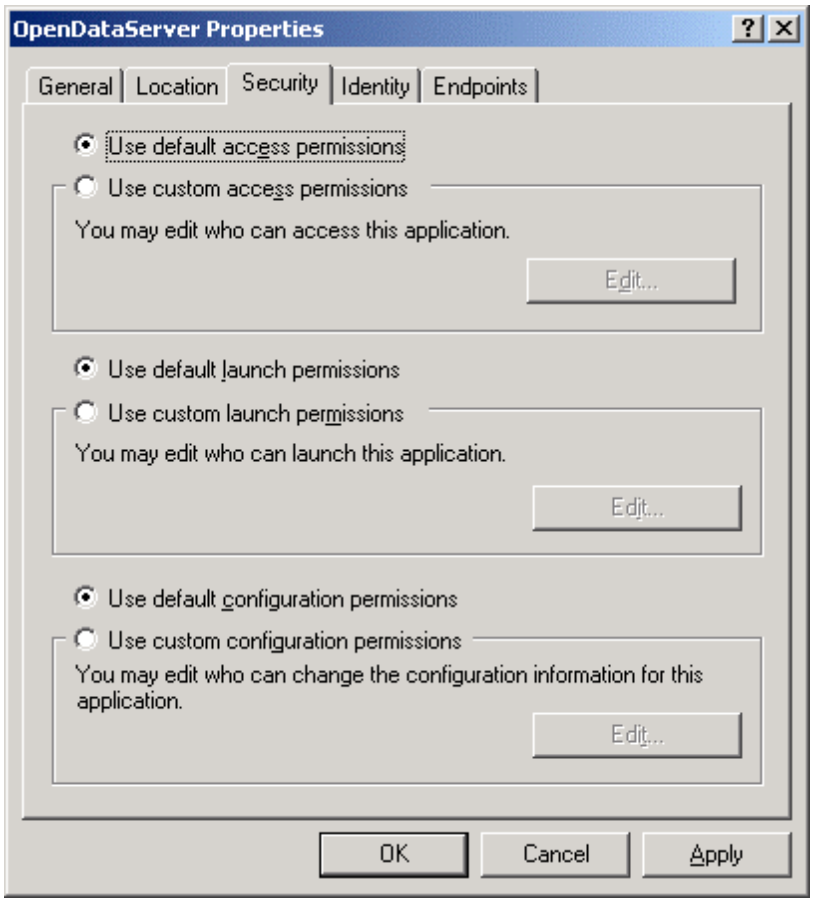
Setup the access permissions by either:

(a) On the **Default Security** tab, adding the user to the Access, Launch and Configuration lists by clicking the **Edit Default…** button in each case. The user added should have Administrator rights on the local PC. If not, it may be necessary to add user groups '**INTERACTIVE**' and '**NETWORK**' as well.

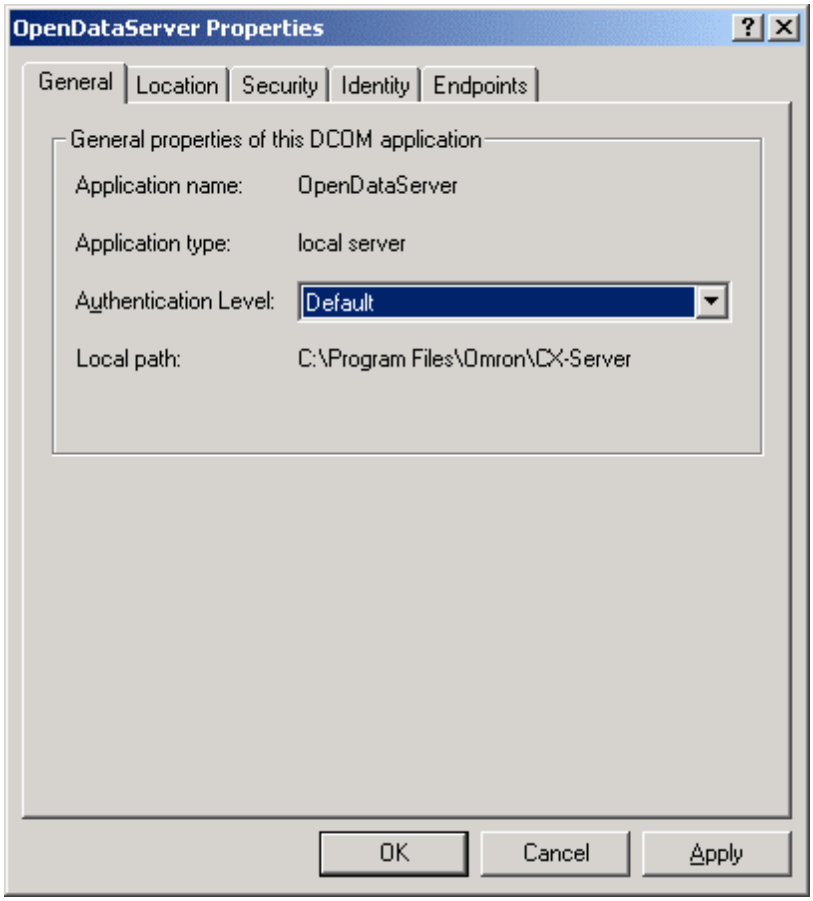


OR

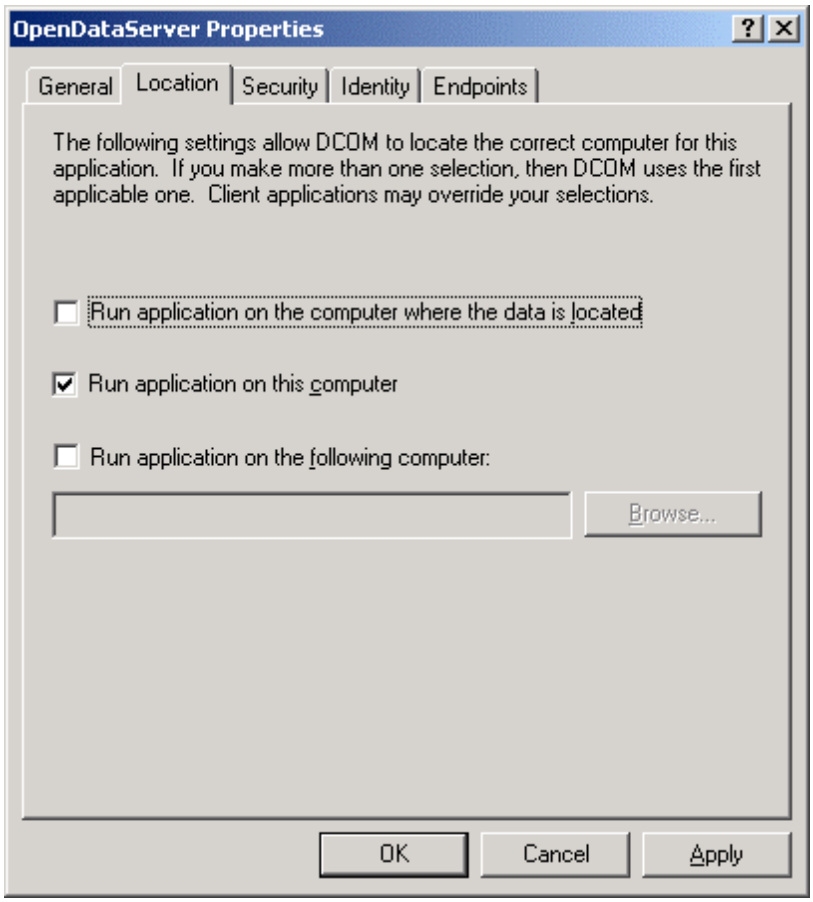
(b) From the NT **Applications** tab, configure the properties for **OpenDataServer** and **OpcEnum**. On the **Security** tab, add the required users to each of the **Custom Permissions.** The users added should have Administrator rights on the local PC. If not, it may be necessary to add user groups '**INTERACTIVE**' and '**NETWORK**' as well. (Note: On XP the application configuration functionality can be accessed from the 'DCOM Config' subtree which appears when 'My Computer' is expanded in the treeview. Locate OpenDataServer in the treeview, right-click on it and select 'properties' from the pop-up menu).



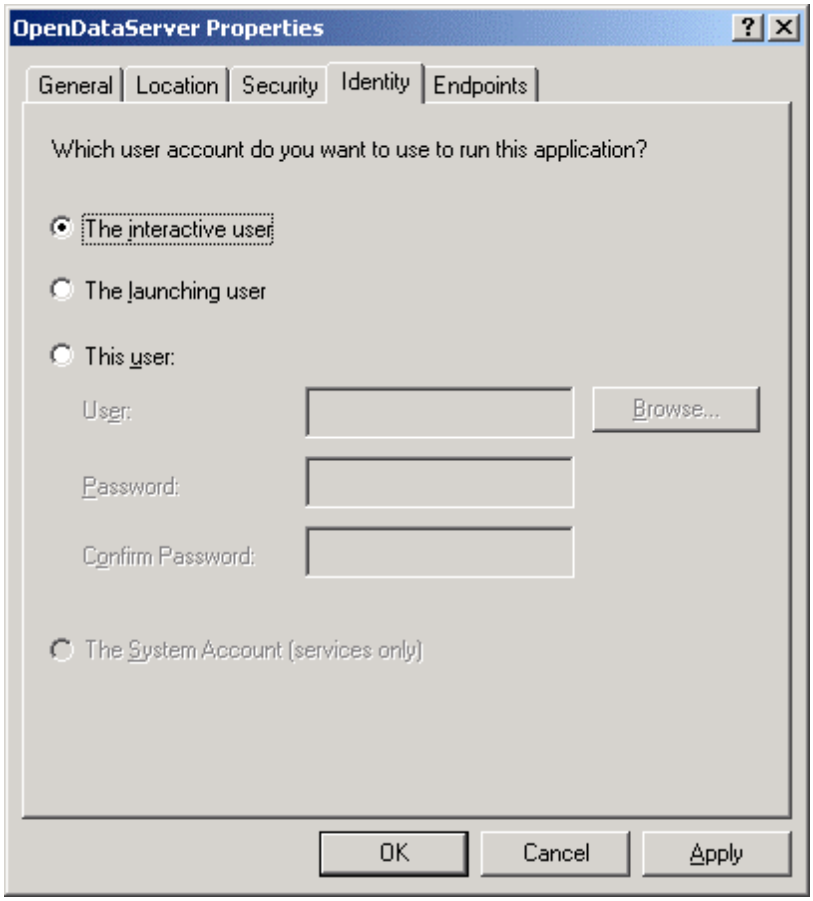
(c) On the **General** tab, set Authentication level to **Default**



(d) On the **Location** tab, set DCOM location to **Run Application on this Computer**



(e) On the **Identity** tab, set the user account to **Interactive User**



7.2 Configuring a machine running Windows 98 or Windows 95

Ensure **File and Printer sharing** is enabled by selecting Network from the Control Panel. If not installed, add a service and click on “File and Print Sharing” and ensure “I want to be able to give others access to my files” is checked.
Start DCOMCNFG.EXE e.g. by selecting **RUN** from the Start button. The default location is C:\WINDOWS\SYSTEM.

View the **Default Properties** tab. Ensure that the **Enable Distributed COM on this computer** is checked.

View the **Default Security** tab and check the **Enable remote connection** check box.

The remainder of the setup is as for Windows NT

Third party servers and clients running on Windows 95 or Windows 98, also require the **Microsoft Remote Registry** network service to be installed with the operating system and correctly configured on both the server and client machine. To check: start the Control Panel and view the Network settings. In the list of network components, look for **Microsoft Remote Registry**. If it does not exist, follow these steps to add it.

1, 2, 3…

1. In the Network settings, ensure **User-level access control** is selected on the **Access Control** tab.
2. From the **Configuration** tab, click **Add** to add a Network component. Choose **Service** from the type list and click **Add**.

3. Click **Have Disk…** and browse your Windows CD. (Note: Remote Registry may not be included on the Windows ME CD, so use a Windows 98 CD.) Select the path **\Tools\ResKit\NetAdmin\RemotReg** (or **\Admin\NetTools\RemoteReg** for Windows95) and select **regsrv.inf**.
4. Follow screen prompts to complete installation and reboot if necessary.
5. On the server machine, select **Passwords** from the Control Panel.
6. Ensure the **Enable remote administration of this server** option is checked.
7. Add all required user ids to the Administrators list by clicking **Add…**.

7.3 DCOM Configuration Settings for Supported OPC Servers

OPC Server	Default Authentication Level	Default Impersonation Level	Access Permissions	Launch Permissions	Configuration Permissions.
SIEMENS WinCC	Not Specified	Not Specified	Administrators Everyone INTERACTIVE	Everyone	Not Specified
Wonderware Intouch	Connect	Identify	Everyone	Everyone	Everyone
Intellution IFix	Not Available at this time	Not Available at this time	Not Available at this time	Not Available at this time	Not Available at this time
GE Cimplicity	(None)	(Anonymous)	Guest (or Domain Guest) INTERACTIVE NETWORK SYSTEM	Guest (or Domain Guest) INTERACTIVE NETWORK SYSTEM ADMINISTRATOR	Guest (or Domain Guest) INTERACTIVE NETWORK SYSTEM ADMINISTRATOR

7.4 Additional DCOM Setup for GE Cimplicity

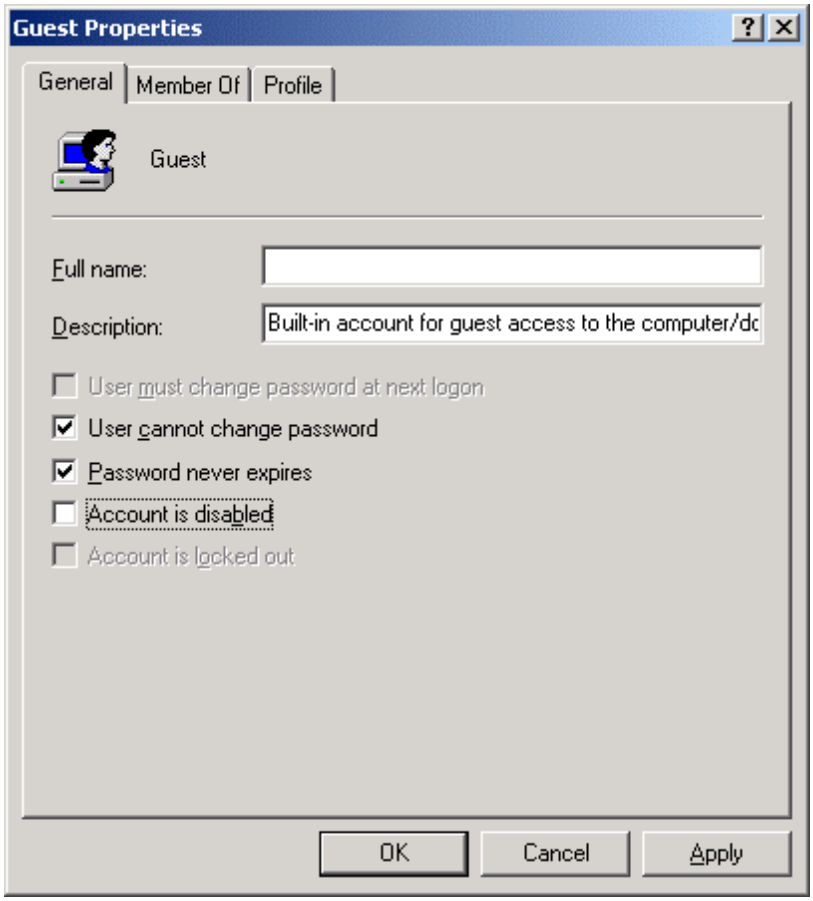
7.4.1 The Windows NT Guest account

When Windows NT is installed, two users are set up by default. They are ADMINISTRATOR and GUEST. The GUEST account, by default, is disabled.

DCOM compliant applications sometimes use this GUEST account to access a computer where a necessary OLE Server component resides. This is because the GUEST account is most likely to exist since it is set up by default during NT installation, and it is safer than using the ADMINISTRATOR account while still providing enough privileges to run the component.

Because the account is disabled by default, the first step is to enable it.
1. From the Start menu, go to **Program**s, **Administrative Tool**s.
2. Run the **User Manager** utility.
Under NT Server, it may be called **User Manager for Domain**s.
The User Managers main window contains two list boxes, one at the top of the window containing User accounts, and one at the bottom of the window containing Groups.

In the top list box, double click on the Username GUEST…



A dialog box similar to the one above should appear. In the lower left area of the dialog box, locate the Check Box labeled: **Account Disable**d. Make sure it is **NOT** checked. Click the OK button to accept the change and close the dialog box. Then exit the User Manager utility.

8. The OPC Compliance Test

8.1 OPC Compliance Test Tool Introduction

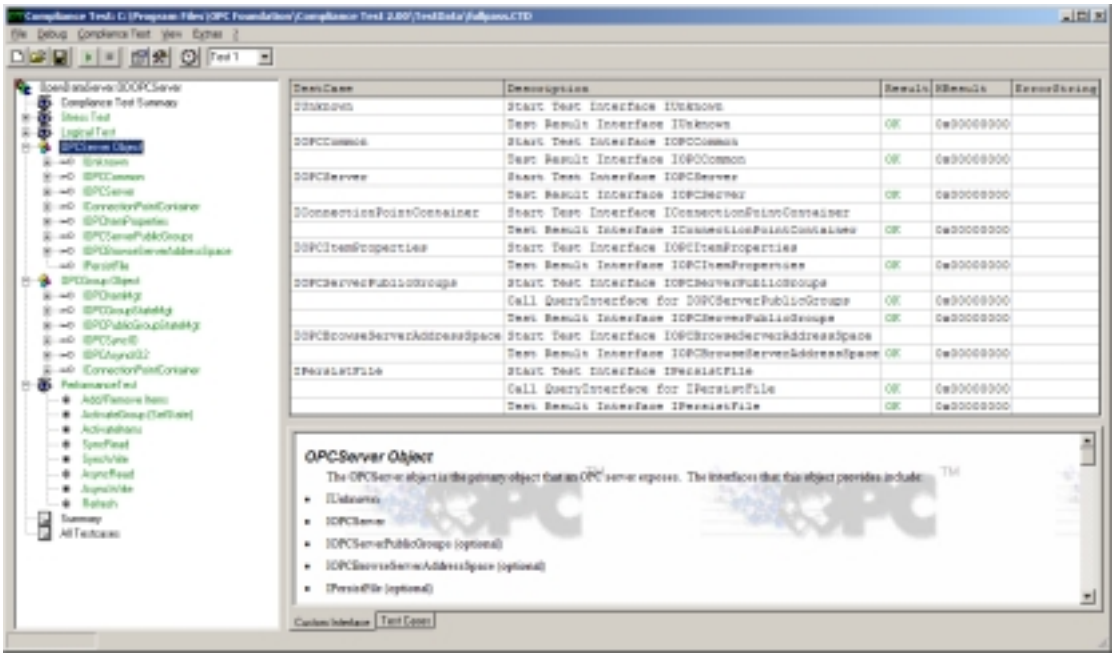
The OPC Foundation provides a free OPC Compliance Test Tool that is used to verify compliance with the OPC Data Access standards.

The tool is continually being upgraded and enhanced, with more and more detailed and thorough tests being added. It is therefore now very rigorous and comprehensive, to the extent where OPC client or server software is probably now considerably more likely to fail a new version of this tool than it is to encounter any problems in real-life operation. (In fact, the tool already tests far more functionality than is used in the vast majority of actual real-world applications.)

8.2 OPC Compliance Test Results

CX-Server OPC has been tested using the OPC Foundation OPC Compliance Test Tool version 2.4 (release 2 build 1110). It is fully compliant with all tests.

A screen capture of the test results (with all tests shown in green – i.e. passed) is shown below:



Appendix A. Some more detailed information

Appendix A.1 DCOM

*Important Note: It is **not** necessary to read and understand this section to make effective use of CX-Server OPC. This information is provided simply as additional background material explaining how communication is achieved between clients and servers using DCOM and ultimately how this is utilised by the OPC components. For even more detail, please consult a book on DCOM or online information.*

Microsoft describes DCOM as

> "The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Previously called "Network OLE," DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP. DCOM is based on the Open Software Foundation's DCE-RPC spec and will work with both Java applets and ActiveX® components through its use of the Component Object Model (COM)."

In other words, DCOM is an object-programming model for the implementation of distributed applications using a client server pattern. A client can use several servers at the same time and a server can provide functionality to multiple clients simultaneously.

Translating that into plainer English, COM basically allows software components to be written in such a way that they can be used by all COM-aware applications (e.g. C++, later versions of Visual Basic) without those applications needing to know anything about the "internals" of the object. DCOM is simply the distributed version of COM – i.e. the objects can be spread across a network. It is a very powerful system, although some machine-level and security configuration may be required to allow it to work correctly and reliably.

Central to the capability of a DCOM object are its interfaces. **All** communication with a DCOM object occurs through its interfaces – an interface is said to provide a "contract" (a full and unchanging description) for the functionality provided by that object. Each interface has a unique ID and describes a group of related methods. The description of the interface defines the syntax and the semantics of the services provided by that interface – the internal implementation of those services doesn't matter to the calling applications.

In the case of OPC these interfaces for each of the DCOM objects are defined within the relevant specifications.

The ID for each interface is described by a unique 128 bit long identifier called GUID (Global Unique Identifier). Once published a DCOM interface cannot be changed since the basic principle of the DCOM is the fixed binary compatibility of an object's interface. This fixed contract between objects guarantees that one object knows exactly how to talk to another object that supports a particular interface. This is called interface programming and provides for robust software development.

DCOM objects reside within DCOM servers. These are components, and there are two kinds of server: In process "InProc" servers are contained in a DLL (Dynamic Link Library), and out of process "OutProc" servers are contained in an .EXE. An InProc server is the situation where the code is executed within the same process. An OutProc server is where the server code is executed in a separate process from the client.

From the perspective of the client application, the Omron OPC Client Communications control provides an InProc server (is linked in as part of the application) while the OPC Server is an OutProc server (a separate .EXE, OpenDataServer.Exe, accessed remotely).

DCOM objects are identified by their Class Identifiers (CLSID) that are another unique GUID. Information about the objects and their identifiers and the interfaces that they support resides within the windows registry. A client uses the CLSID of a DCOM component to launch it by using the DCOM library and the information about where the component exists from within the

registry. This is the reason why it is so important to have the correct installation of a product, especially OPC since if the information is not correctly entered into the registry or if the DCOM object contained in either the DLLs and .EXEs are not compatible the client server relationship via their interfaces cannot be brokered correctly.

OutProc servers require that the interfaces between the client and the server must be marshalled according a proxy-stub paradigm. OutProc servers are written in C++ and the MIDL (Microsoft Interface Definition Language) compiler is used to generate the proxy stub objects.

The proxy object implements the interface that is to be marshalled, and looks to the client exactly like the object. In addition to this, proxy objects also implement the IRpcProxyBuffer interface, which is used by an object called the proxy manager that COM creates in the client. The proxy manager uses this interface when it makes the connection between proxy and stub. Since each proxy object implements this interface, the proxy manager will be able to talk to each proxy in the same way.

COM/DCOM can use one of many types of inter-process communication (IPC) mechanisms, and the IPC code is wrapped up as a channel object which implements the *IRpcChannelBuffer* interface that generalizes the sending and receiving of marshalling packets. This interface abstracts the client and object from the IPC so that, in fact, they have no knowledge of which is used.

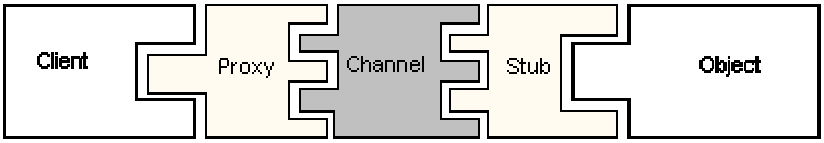


Figure A1: Proxy, stub and channel objects

Figure A2 shows the actual objects used in standard marshalling. The proxy manager aggregates the proxy object, exposing the object interface. It then creates a channel object and passes the channel's *IRpcChannelBuffer* interface to the proxy so that when the client calls an interface method it can make a marshalling packet and transmit it via the channel to the stub.

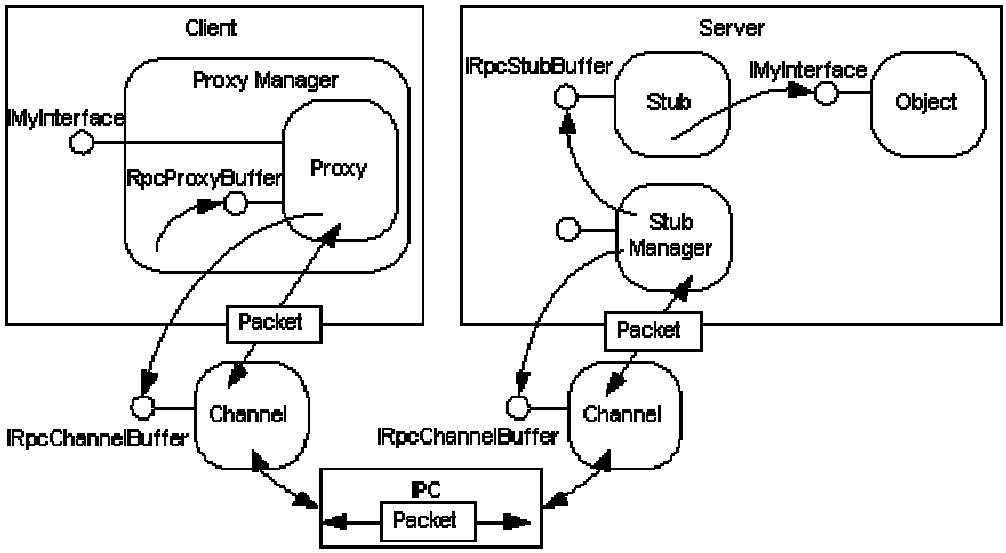


Figure A2: Standard marshalling

The stub manager in the object server manages stub objects. Stubs implement the *IRpcStubBuffer* interface and through this they are passed the interface pointer on the object to make the connection. When the stub manager gets a marshalling packet from the client it can call the stub through its *IRpcStubBuffer* interface to give the stub the chance to un marshal the packet and call the object.

The good thing about this architecture is that it is modular and hence very flexible. Since the proxy and stub talk to the channel object through the standard *IRpcChannelBuffer* interface it means that multiple channels can be implemented using different IPC mechanisms. Thus, the channel used for communication between client and objects on a single machine is LRPC (Local RPC, essentially optimized to use Windows messages); for DCOM on Windows 95 is TCP/IP; and DCOM on NT4 uses any one of UDP/IP, TCP/IP, IPX, SPX or NetBIOS.

It looks very complicated, but it is not necessary to write proxy or stub objects to use OPC clients and servers. It is enough to be aware that the proxy-stub DLLs must be present and registered on each of the PCs that the clients and servers reside upon.

Appendix B. Links and sources of information.

www.opcfoundation.org – Main OPC Foundation website
www.opceurope.org – European OPC Foundation website