

iSNMP OPC Server Help

© 2008 Kepware Technologies

Table of Contents

1	Getting Started.....	4
	iSNMP OPC Server.....	4
	Introduction	4
	System Requirements.....	8
2	Server Features.....	8
	Server Features.....	8
	Statistics Tags.....	11
	System Tags	13
	Property Tags.....	20
	CSV Import and Export.....	21
	Project Properties.....	22
	NetDDE	22
	DDE	24
	Processing Array Data in a DDE Client.....	24
	OPC Data Exchange Plug-in.....	25
	Modem Support.....	25
	Modem Support.....	25
	Dial Tag	28
	DialNumber Tag.....	29
	Hangup Tag.....	29
	LastEvent Tag.....	29
	Mode Tag.....	29
	PhoneNumber Tag.....	30
	Status Tag.....	30
	StringLastEvent Tag.....	31
	StringStatus Tag.....	31
	Phonebook.....	31
	Phone Number.....	32
	USRobotics Quick Reference.....	33
	The User Manager.....	41
	User Manager.....	41
	User Properties.....	41
	Event Log	42
	Event Log Print - Page Setup.....	42
3	Basic Server Components.....	43
	Basic Server Components.....	43
	What is a Channel?.....	44
	Channel Functions.....	44
	Channel Properties.....	45
	New Channel - Name Page.....	45
	Channel Properties - Identification.....	45
	New Channel - Driver Page.....	45
	Channel Properties - Comm. Parameters.....	45
	Channel Properties - Network Interface.....	46
	Flow Control RTS & DTS.....	47
	Channel Properties - Manual RTS Flow Control.....	48
	Channel Properties - Modem.....	48
	Channel Properties - Write Optimizations.....	49

Channel Properties - Ethernet Encapsulation.....	51
New Channel - Summary.....	54
What is a Device?.....	54
Device Functions.....	54
Device Properties.....	55
New Device - Name.....	55
New Device - Model.....	55
New Device - ID.....	56
Device Properties - General.....	58
Device Properties - Ethernet Encapsulation.....	60
Device Properties - Timing.....	63
Device Properties - Auto-Demotion.....	65
Automated OPC Tag Database Generation.....	66
New Device - Summary.....	69
What is a Tag?.....	69
Tag Functions.....	69
Tag Properties.....	69
Tag Properties.....	69
Dynamic Tags.....	71
Static Tags (User Defined).....	72
Scaling.....	72
Tag Group Properties.....	74
Tag Selection.....	75
What is the Alias Map?.....	76
Alias Map.....	76
Alias Properties.....	77
Creating an Alias.....	78
What is Diagnostics?.....	80
Diagnostics Overview.....	80
OPC Diagnostics.....	80
Channel Diagnostics.....	82
4 Designing a Project.....	86
Designing a Project.....	86
Designing a Project: Running the Server.....	87
Designing a Project: Starting a New Project.....	87
Designing a Project: Adding and Configuring a Channel.....	87
Designing a Project: Adding and Configuring a Device.....	91
Designing a Project: Adding Tags to the Project.....	94
Designing a Project: Adding Tag Scaling.....	98
Designing a Project: Saving and Testing the Project.....	99
5 Server Options.....	106
General Options.....	106
View Options.....	108
Service Options.....	109
Event Log Options.....	113
OPC Options.....	114
OPC Compliancy Options.....	116
OPC DX Options.....	117
FastDDE & SuiteLink Options.....	118
DDE Options.....	121
iFIX PDB Options.....	123
iFix Signal Conditioning Options.....	127
6 License Transfer.....	133

License Transfer: Instructions (Step1).....	133
License Transfer: Agreement (Step2).....	134
License Transfer: Selection (Step3).....	134
License Transfer: Preparing Removable Disk (Step4).....	136
License Transfer: Moving the License (Step5).....	137
License Transfer: License Installation (Step6).....	139
License Transfer: Transfer Completion (Step7).....	140
7 License and Unlock a Driver or Plug-in.....	141
Licensing Overview.....	141
License a Driver or Plug-in.....	141
Unlock a Licensed Driver or Plug-in.....	142
8 How Do I...?.....	143
How Do I... ..	143
Index	144



CONTENTS

[Introduction](#)

[Server Features](#)

[Basic Server Components](#)

[Options](#)

[Designing a Project](#)

[Purchasing and Licensing a Driver](#)

[Transfer a Driver License](#)

[How do I . . . ?](#)

Help version 1.077

Introduction to iSNMP OPC Server

iSNMP OPC Server is a 32-bit windows application that provides a means of bringing data and information from a wide range of industrial devices and systems into client applications on your windows PC. iSNMP OPC Server falls under the category of a "Server" application. It is very common to hear the term "client/server application" in use across many software disciplines and business segments. In the industrial market, it has come to mean the sharing of manufacturing or production data between a variety of applications ranging from human machine interface software and data historians, to large MES and ERP applications.

Regardless of the business segment served, client/server applications have one thing in common: a standardized method of sharing data. In the industrial segment many client/server technologies have been developed over the last ten years. Initially some of these technologies were proprietary. In many cases these proprietary client/server architectures were in wide use but remained unavailable to third party applications. Early in the development of windows, Microsoft provided a generic client server technology called DDE or Dynamic Data Exchange. DDE provided a basic architecture that would allow many windows applications from a wide range of vendors to share data, but it was not designed for the industrial market. It lacked much of the speed and robustness desired in an industrial setting. However, this did not stop DDE from becoming dominant client/server architecture, largely due to its availability in most windows applications. In time, variations on Microsoft's DDE were developed by some of the leading vendors in the market. These variations addressed some of the speed and reliability issues of DDE but many people in the industrial segment agreed that a better system needed to be developed.

OPC (OLE for Process and Control)

In 1994 a group of vendors representing a broad spectrum of disciplines in industrial segment formed what is now known as the OPC Foundation. The OPC Foundation put forth the goal of developing a single client/server specification that would allow any vendor to develop software and applications that could share data in a fast, robust fashion, and do it in a way that would eliminate the proprietary schemes that forced these same vendors to duplicate development efforts. The OPC Foundation developed the first specification called Data Access Specification 1.0a that was released in early 1996. Using this specification, vendors were able to quickly develop client server software.

A major goal of the OPC Foundation and the Data Access specification was to eliminate the need of client application vendor's to develop their own proprietary set of communications drivers. For many vendors, the effort required to develop numerous communications drivers outweighed the development effort involved in the client application itself. With the adoption of OPC technology a vendor could now focus their efforts almost exclusively on the development of the client application. The Data Access specification defines how both the client and the server application interface must be constructed. If the specification is followed properly, a client vendor knows that any OPC server that exists for an industrial device can provide the connectivity needed for data access. Issues like time to market or reliability no longer restrict applications to which any OPC compatible application can address. OPC has given the end user the additional benefit of being able to select the best of breed software to solve application problems. Historically, if the application software did not have the desired communication driver or if the available driver didn't perform adequately, the only solution was to try to persuade the application vendor to either develop the desired driver or repair an existing driver. The time required in either of these cases was usually never short. With OPC, the end user is no longer tied to the resource limitations of the client application vendor. The user can now choose from a variety of OPC server vendors

to address a new driver requirement or remedy a performance issue. Equally, the client application vendor can now focus on the continued improvement of their core product without the disruptive effort required to address communication issues and needs. Our goal within the OPC environment is to be a leading provider of the server component of the OPC equation and to do so by providing a product that is reliable and easy to use. This server is built upon years of development efforts in communications driver development and OPC technology.

A lot has been said here about OPC but how does the OPC specification work? The OPC Foundation has provided a good overview of OPC technology. With permission of the OPC Foundation, we have provided the following section of the Data Access specification.

OPC Data Access Fundamentals

This section introduces OPC Data Access and covers topics that are specific to OPC Data Access.

OPC Overview

This specification describes the OPC COM Objects and their interfaces implemented by OPC Servers. An OPC Client can connect to OPC Servers provided by one or more vendors.

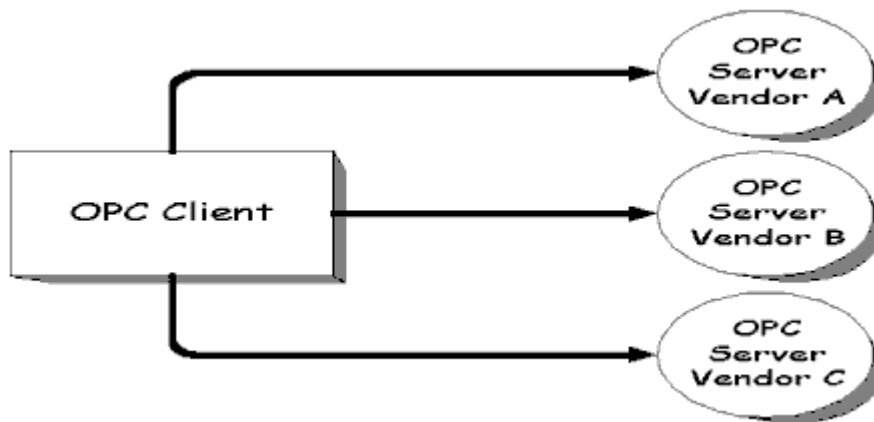


Figure: OPC Client

Different vendors may provide OPC Servers. Vendor supplied code determines the devices and data to which each server has access, the data names, and the details about how the server physically accesses that data. Specifics on naming conventions are supplied in a subsequent section.

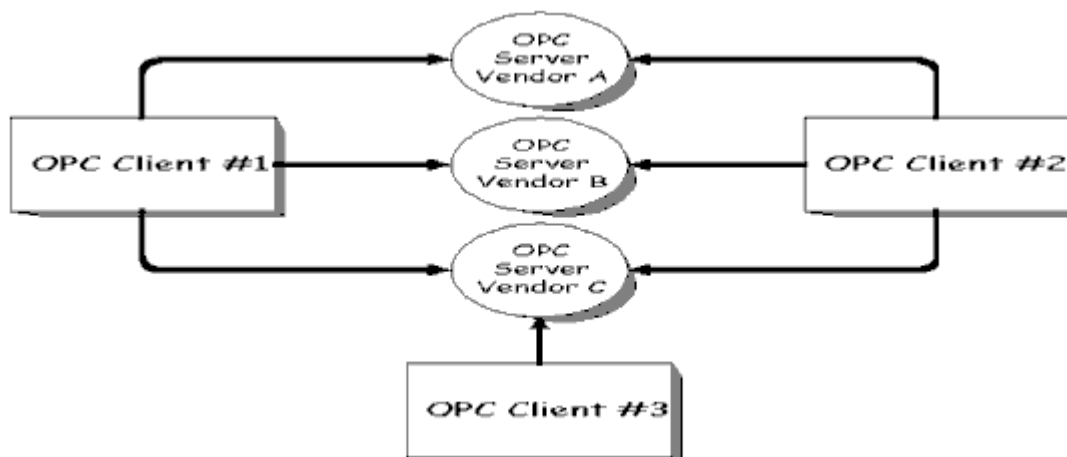


Figure: OPC Client/Server Relationship

At a high level, an OPC 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.

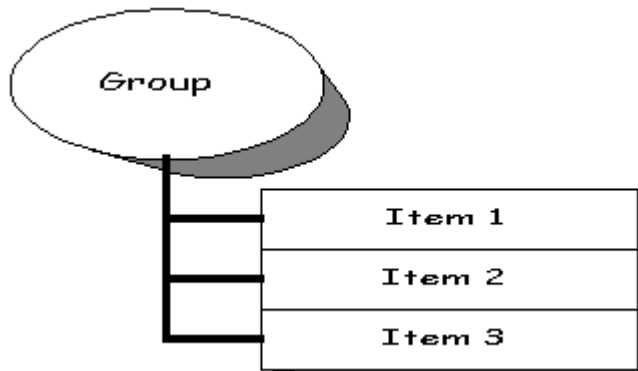


Figure: Group/Item Relationship

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: 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.

Where OPC Fits

Although OPC is primarily designed for accessing data from a networked server, OPC interfaces can be used in many places within an application. At the lowest level they can get raw data from the physical devices into a SCADA or DCS, or from the SCADA or DCS system into the application. The architecture and design makes it possible to construct an OPC Server which allows a client application to access data from many OPC Servers provided by many different OPC vendors running on different nodes via a single object.

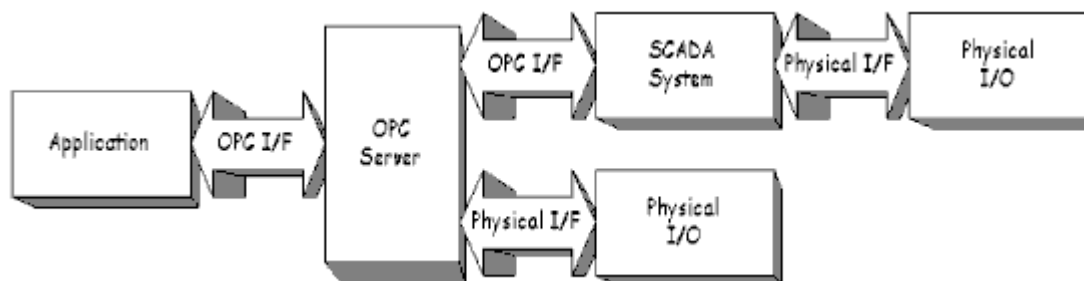


Figure: OPC Client/Server Relationship

General OPC Architecture and Components

OPC is a specification for two sets of interfaces; the OPC Custom Interfaces and the OPC Automation interfaces. A

revised automation interface will be provided with release 2.0 of the OPC specification. This is shown below.

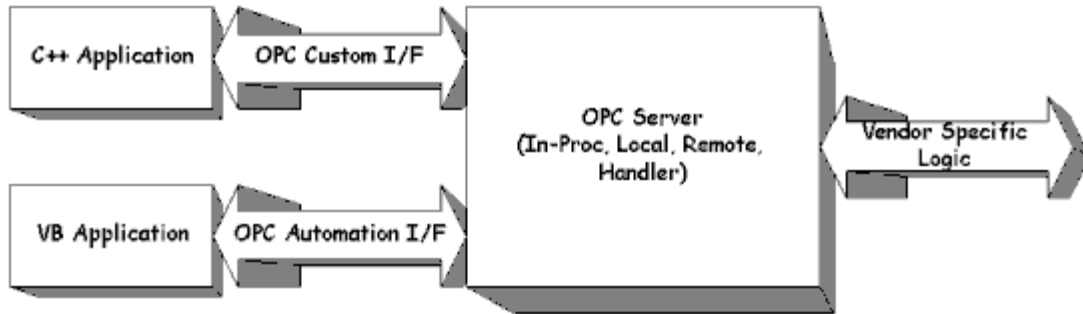


Figure: OPC Interfaces

The OPC Specification specifies COM interfaces (what the interfaces are), not the implementation (not the how of the implementation) of those interfaces. It specifies the behavior that the interfaces are expected to provide to the client applications that use them.

Included are descriptions of architectures and interfaces that seemed most appropriate for those architectures. Like all COM implementations, the architecture of OPC is a client-server model where the OPC Server component provides an interface to the OPC objects and manages them.

There are several unique considerations in implementing an OPC Server. The main issue is the frequency of data transfer over non-sharable communications paths to physical devices. Thus, we expect that the OPC Server will either be a local or remote EXE, which includes code that is responsible for efficient data collection from a physical device.

An OPC client application communicates to an OPC server through the specified OPC custom and automation interfaces. OPC servers must implement the custom interface, and optionally may implement the automation interface.

An "InProc" (OPC handler) may be used to marshal the interface and provide the additional Item level functionality of the OPC Automation Interface. Refer to the figure below: Typical OPC Architecture.

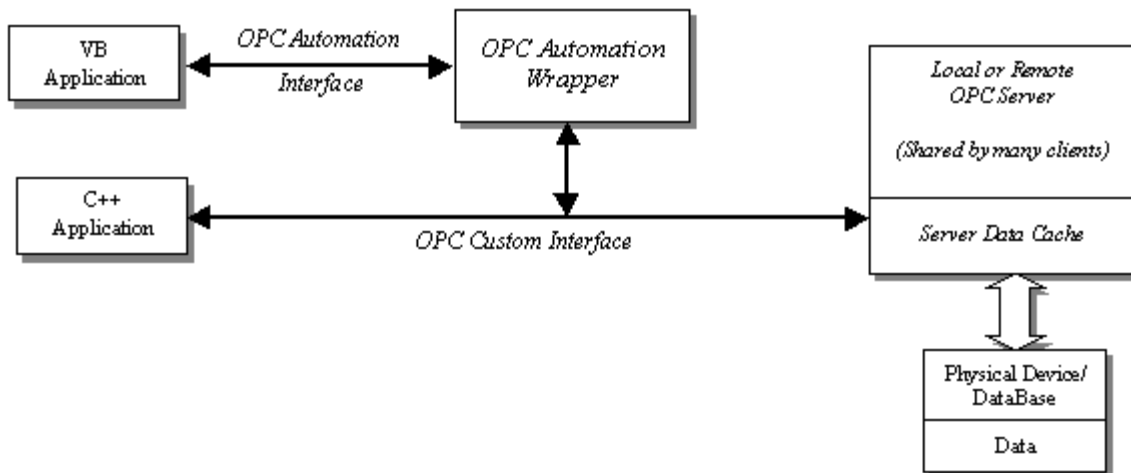


Figure: Typical OPC Architecture

It is also expected that the server will consolidate and optimize data accesses requested by the various clients to promote efficient communications with the physical device. For inputs (Reads), data returned by the device is buffered either by asynchronous distribution or synchronous collection by various OPC clients. For outputs (writes), the OPC Server updates the physical device data on behalf of OPC Clients.

DDE Fundamentals

While the Server is first and foremost an OPC server, it was recognized that a number of legacy applications still depend upon DDE for their underlying client server technology. To address these applications the server has been designed to provide the same access to device data via DDE as can be achieved using OPC.

The server supports these formats of DDE:

1. CF_Text
2. XL_Table
3. AdvancedDDE
4. NetDDE
5. FastDDE/SuiteLink

System Requirements

The OPC server has both software and hardware minimum system requirements. These requirements must be met in order for the application to operate as designed.

This application supports the following Microsoft Windows operating systems:

- Windows 2003 Server*
- Windows XP*
- Windows 2000 Server
- Windows 2000 Service Pack 2 or higher
- Windows NT 4.0 Service Pack 6
- Windows Vista Business/Ultimate (32 bit and 64 bit)
- Windows Server 2008 (32 bit and 64 bit)

*Since Windows Server 2003 and Windows XP have continuous updates, the Windows update feature should be run to get the latest software.

The OPC server requires, at a minimum, the following hardware:

- Intel Pentium III 400 MHz or equivalent processor that supports Microsoft's Windows operating system
- 512 MB installed RAM (256 MB free)
- 40 MB available disk space
- Available Serial Port or Ethernet Card

Server Features

The latest generation of our OPC server technology has incorporated many of the features requested by customers. In addition to customer driven enhancements, many technological changes have occurred. These features and enhancements have all been made with the goal of providing an OPC server that demonstrates unparalleled compatibility and performance.

A few of the enhancements are transparent to the user, but there are a number of new features that are readily apparent and directly available to the user. The following sections will describe the primary features of server.

Connectivity

The server has been enhanced to provide the widest range of connectivity of any server product. It supports the following client server technologies:

OPC Data Access Version 1.0a
OPC Data Access Version 2.05a
OPC Data Access Version 3.0
DDE Format CF_Text
DDE Format AdvancedDDE

OPC Data Access 1.0a was the original specification the OPC Foundation developed back in 1996. Many of the OPC client applications in use today support this original specification. OPC Data Access 2.0 enhanced OPC to make better use of the underlying Microsoft COM technology. Most OPC client applications support version 2.0 of the OPC specification. OPC Data Access 3.0 is the latest version of the OPC interface. The 3.0 specification has added several enhancements to the OPC interface, such as Data Sampling, Item Level Deadband, and the ability for servers to provide keep-alive callbacks when there are no data changes to report for a specified interval of time. The DDE format CF_Text is the standard DDE format as defined by Microsoft. All DDE aware applications support the CF_Text format. AdvancedDDE is a variation on the normal CF_Text format. Advanced DDE allows larger amounts of data to transfer

between applications at higher rates of speed, and with better error handling than a normal CF_Text DDE link. In keeping with our goal of providing the broadest range of connectivity, our OPC server simultaneously supports the entire client server technologies listed above. Client applications using any of these technologies can access data from the server at the same time.

Based on Microsoft's COM technology, OPC servers can share data with remote client applications using DCOM (Distributed COM). DCOM is used to use a single OPC server to provide data to client applications running both locally and on remote machines. DDE is not without its own means of allowing remote access. All of the DDE formats supported by server can also be accessed remotely using what is known as [NETDDE](#). NETDDE allows a remote DDE client application to use the machine name of a remote DDE server when specifying a DDE link. In terms of OPC connections, our server will properly configure your DCOM settings to allow remote OPC clients to access and browse the server for tags. For DDE clients, the server will automatically start NETDDE services and register all of the required DDE shares, to allow remote DDE clients to access device data. Establishing DDE share names can be a time consuming process so as the default, NETDDE services are not enabled in the server.

Runs as NT Service

The server supports running as a service under Windows NT/2000/XP/Server2003. Service operation is completely user configurable from the [Tools Options](#) menu and can be changed at any time allowing you to move from normal stand-alone program operation to NT service mode. Running as an NT service is crucial for many applications where the server provides data to OPC clients via DCOM. For these applications a loss of DCOM connection cannot be tolerated. Normally an OPC server that only supports stand alone program operation is forced to shut down when its host machine experiences a user log in or log out. While running as a service, the server can continue to supply OPC data across user log in sessions and can be configured to interact with the desktop allowing you to make changes to your server project. It can also be configured to have a visible presence while running.

Data Scaling

Direct scaling of device data is now supported by our OPC server. [Scaling](#) allows raw device data to be converted to engineering units for OPC client applications. We provide a number of unique scaling features that make it easy to implement scaling in the application.

The screenshot shows the 'Tag Properties' dialog box with the 'Scaling' tab selected. The 'None' radio button is unselected, 'Linear' is selected, and 'Square root' is unselected. The 'Raw Value Range' section has 'Data type: Word', 'High: 1000', and 'Low: 0'. The 'Scaled Value Range' section has 'Data type: Double', 'High: 100000' with a checked 'Clamp' box, 'Low: 10' with a checked 'Clamp' box, 'Units: Gals/Min', and an unchecked 'Negate scaled value' checkbox. At the bottom are 'OK', 'Cancel', 'Apply', and 'Help' buttons.

The Scaling in the server supports Linear and Square Root formulas. You can specify the range of the raw data from the device and the engineering range of the scaled value. In some cases the raw data received from a device may exceed the range set for the raw data. If this occurs the engineering value can be forced outside of the range you desire. To prevent this, we allow you to specify that the scaled value be clamped to the engineering ranges. In most cases, it is

assumed that a scaled value results in a floating-point number, but our server doesn't make this assumption and is used to select the scaled engineering value to be any valid OPC data type. This means you can scale a 16-bit integer value to a 32-bit integer value. Double is the default data type for all scaled values. To make scaling complete, the server is used to specify the units for the scaled tag. A string of up to 32 characters can be entered and attached to the tag. If the OPC client application in use supports access to OPC Tag Properties, the data ranges and the units can be used in the OPC client to automatically configure objects like user input or data displays.

Full Time online

To acquire data from a PLC or device, a channel and device must be configured in the server, and a client application must be requesting data. The full time online mode of operation is used r OPC server project to be modified while the server continues to supply data to client applications. Almost every parameter can be changed while the server is operating (although some parameters including tag attributes may not take effect until you reconnect the OPC client). Parameters like communication port or baud rate can be changed while a client application is active, if needed. More importantly, [user defined tags](#) can be added to the server without shutting down client applications. When new tags are added to the server, they are immediately added to the OPC browse space, and will be available to OPC clients.

User Management

With a powerful feature like online full time operation, managing what your users can do in the OPC application becomes a necessity. The server includes a built in [User Manager](#) that allows complete control over what types of functionality each individual user can access. The default administrator account is used to add multiple users, each with their own set of rights for server access. Any user action that can influence or disrupt server operation is logged to server's event logging system. By default, all server operations are available at all times; the User Manager functions of server are available only if you need them.

Tag Management

The server's new user defined [tag](#) management features allow you to create a tag database structure that fits the nature of the application. Multiple [tag groups](#) can be defined to segregate your tag data on a device-by-device basis. Drag and drop editing makes adding large numbers of tags easy. Additionally, CSV import and export allows tag editing to be done in any application you desire. Like all other features in our server, new tags can be added to the application at any time.

Automatic Tag Database Generation

The OPC server supports the automatic generation of tags for select communication drivers. The [Automatic Tag Database Generation](#) feature brings OPC technology one-step closer to Plug and Play operation. Drivers that support this feature can either read tag information directly from a device or generate tags from stored tag data. In either case the user no longer needs to enter OPC tags into the server.

Auto-Demotion

The [Device Auto-Demotion](#) parameters allow a driver to temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device off-line, the driver can continue to optimize its communications with other devices on the same channel by stopping communications with the non-responsive device for a specific time period. After the specific time period has been reached, the driver will re-attempt to communicate with the non-responsive device. If the device is responsive, the device will be placed on-scan, otherwise it will restart its off-scan time period.

Network Interface Selection

A NIC card can be specifically selected for use with any Ethernet driver or serial driver running in [Ethernet Encapsulation](#) mode. The [Network Interface](#) feature is used to select a specific NIC card based on either the NIC name or it currently assigned IP address. This list of available NICs will include both unique NIC cards and NICs that have multiple IPs assigned to them. Additionally the selection will also display any WAN connections you may have active such as a dialup connection.

Ethernet Encapsulation

The Ethernet Encapsulation mode has been designed to provide communications with serial devices connected to terminal servers on your Ethernet network. A terminal server is essentially a virtual serial port. The terminal server converts TCP/IP messages on your Ethernet network to serial data. Once the message has been converted to a serial form you can connect standard devices that support serial communications to the terminal server. Using a terminal server device is used to place RS-232 and RS-485 devices throughout your plant operations while still allowing a single localized PC to access the remotely mounted devices. The Ethernet Encapsulation mode allows an individual Network IP address to be assigned to each of the devices as needed. Using multiple terminal servers, users can access hundreds of serial devices from a single PC via the Ethernet network. **See Also:** [Device Properties - Ethernet Encapsulation](#).

OPC Diagnostics

The server's new OPC diagnostics functionality displays all interaction between the server and any Data Access 1.0, 2.0 and 3.0 clients simultaneously. You can filter out events to focus on the methods that are causing problems. **See Also:** [OPC Diagnostics](#) for more details.

Channel Diagnostics

The server's new diagnostic features provide real-time data on the performance of your communication driver. All read and write operations can be viewed in the [diagnostic display window](#) or can be tracked directly in the OPC client application, by using its built-in [diagnostic tags](#). These diagnostics make it easy to debug really tough communication issues. The diagnostic display window also provides a real-time protocol view. Given that the server is online full time, you can view the real-time protocol window while you make changes to key communications parameters like baud rate, parity, or Device IDs. As you make changes to your communications parameters you'll see the effect on communications in real-time, and once you set the correct communication and device settings, you'll immediately see the exchange of data with the device.

Modem Support

The server supports the use of [modems](#) to connect to remote devices. It does this through the use of special modem tags that become available at the channel level when a dial-up network connection has been created. These tags can be used to dial a remote device, monitor the modem status while connected, and terminate the call when finished.

System Tags

The system tags are used to provide general error feedback to client applications, allow operation control over when a device is actively collecting data, and allow the standard parameters of a either a channel or device to be changed from an OPC client application on the fly. The number of system tags available at either the channel level or device level will vary depending on the nature of the driver you are using. The system tag can also be grouped according to their purpose as both status and control or parameter manipulation. **See Also:** [System Tags](#).

Property Tags

Tag Properties are now available as additional tags that can be accessed by any Data Access client by appending the property name to any fully qualified tag address. If you are using an OPC client that supports item browsing, you browse Tag Properties by turning on "Include Tag Properties when a client browses the server" under [OPC Settings](#). **See Also:** [Property Tags](#).

System Tray

We support additional viewing options that provide extended flexibility in deploying the OPC application. The new System tray options allow you to configure when and how the server will display the main window. Using the [View Options](#) settings you can configure the server to only appear as Icon in the system tray when running. Once the server Icon is placed in the system tray, the server can be brought to the foreground if required by simply clicking its system tray Icon.

Statistics Tags

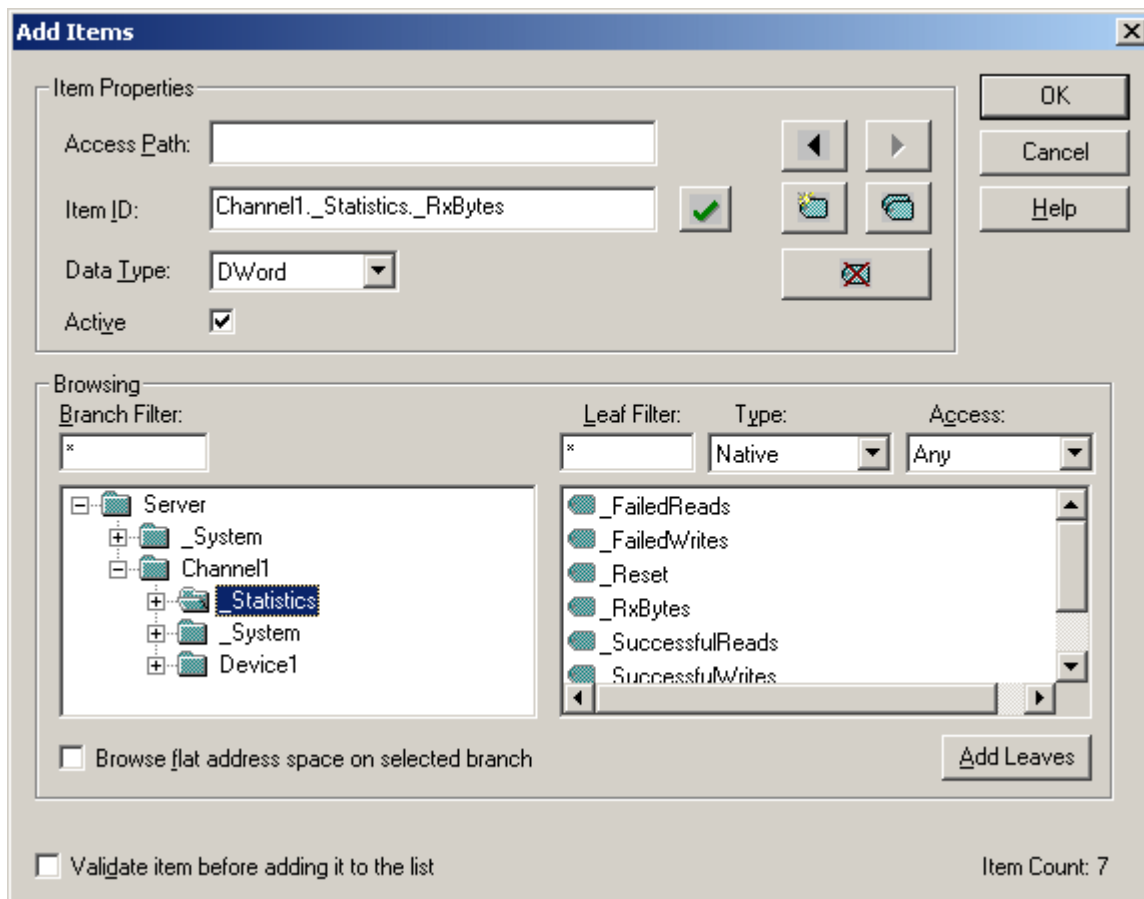
Statistics tags are used to provide feedback to client applications regarding the operation of the channel communications in the server. Currently there are seven built-in statistics tags available when diagnostics are enabled. **See Also:** [OPC Diagnostics](#).

Syntax Example: <Channel Name>._Statistics._FailedReads

_SuccessfulReads	The <code>_SuccessfulReads</code> tag contains a count of the number of reads this channel has completed successfully since the start of the application or since the last time the <code>_Reset</code> tag was invoked. This tag is formatted as unsigned 32 bit integer and will eventually rollover. This tag is Read Only.
_SuccessfulWrites	The <code>_SuccessfulWrites</code> tag contains a count of the number of writes this channel has completed successfully since the start of the application or since the last time the <code>_Reset</code> tag was invoked. This tag is formatted as an unsigned 32 bit integer and will eventually rollover. This tag is Read Only.
_FailedReads	The <code>_FailedReads</code> tag contains a count of the number of reads this channel has failed to complete since the start of the application or since the last time the <code>_Reset</code> tag was invoked. This count is only incremented after the channel has failed the request based on the configured timeout and retry count for the device. This tag is formatted as an unsigned 32 bit integer and will eventually rollover. This tag is

	Read Only.
_FailedWrites	The _FailedWrites tag contains a count of the number of writes this channel has failed to complete since the start of the application or since the last time the _Reset tag was invoked. This count is only incremented after the channel has failed the request based on the configured timeout and retry count for the device. This tag is formatted as unsigned 32 bit integer and will eventually rollover. This tag is Read Only.
_RxBytes	The _RxBytes tag contains a count of the number of bytes the channel has received from connected devices since the start of the application or since the last time the _Reset tag was invoked. This tag is formatted as unsigned 32 bit integer and will eventually rollover. This tag is Read Only.
_TxBytes	The _TxBytes tag contains a count of the number of bytes the channel has sent to connected devices since the start of the application or since the last time the _Reset tag was invoked. This tag is formatted as unsigned 32 bit integer and will eventually rollover. This tag is Read Only.
_Reset	The _Reset tag can be used to reset all diagnostic counters. The _Reset tag is formatted as a Boolean tag. Writing a non-zero value to the _Reset tag will cause the diagnostic counters to be reset. This tag is Read/Write.

The statistics tags are only available when you have enabled diagnostics on the [Channel Properties](#) page. To access from an OPC client, the diagnostic tags can be browsed from the **_Statistics** branch of the server browse space for a given channel. The following diagram, taken from the supplied OPC Quick Client, shows how the diagnostic tags appear to an OPC client.



The `_Statistics` branch found under the Channel branch will only appear when diagnostics are enabled for the channel. If you needed to reference a diagnostic tag from a DDE application given the above example and the DDE defaults, the link would appear as follows:

```
= <DDE service name>|_ddedata!Channel1._Statistics._SuccessfulReads
```

The values of the diagnostic tags can also be viewed directly in the server by using the [Channel Diagnostics](#) window. If "Enable Diagnostics" has been selected under Channel Properties, right-click on that channel and then select **Diagnostics**.

See Also: [System Tags](#) and [Property Tags](#).

Note: Modem Tags are described in the topics under [Modem Support](#)

System Tags

System tags are used to provide general error feedback to client applications, to allow operational control when a device is actively collecting data, and to allow the standard parameters of a either a channel or device to be changed by an OPC client application on the fly.

System Tags

The number of system tags available at either the channel level or device level will vary depending on the nature of the driver you are using. In addition to channel-level and device-level system tags, there are now application-level system tags which allow client applications to monitor the status of the server. System tags can also be grouped according to their purpose as both status and control, or parameter manipulation.

Parameter Control Tags

While the standard system tags provide needed feedback on server operation, the parameter control tags provide the most powerful feature. Parameter control tags can be used to modify the operational characteristic of the server application. This provides a great deal of flexibility in your OPC applications. Using the parameter control tags you can implement redundancy by switching communications links or changing the Device ID of a target device—all on the fly. You could also provide access to these tags through special supervisory screens that allow a plant engineer to make changes to the communication parameters of the server if needed.

The tables below include descriptions of the following:

[Application Level System Tags](#)

[Channel Level System Tags / Serial](#)

[Channel Level System Tags / Ethernet](#)

[Device Level System Tags / Serial and Ethernet](#)

Application Level System Tags

Syntax example: `<Channel Name>.<Device Name>._System._ActiveTagCount`

<code>_ActiveTagCount</code> Class: Status Tag	The <code>_ActiveTagCount</code> is a tag that indicates the number of tags that are currently active in the server. This is a Read Only tag.
<code>_ClientCount</code> Class: Status Tag	The <code>_ClientCount</code> is a tag that indicates the number of clients that are currently connected to the server. This is a Read Only tag.
<code>_Date</code> Class: Status Tag	The <code>_Date</code> is a tag that indicates the current date of the system that the server is running on. The format of this string is defined by the operating system date/time settings. This is a Read Only tag.
<code>_DateTime</code> Class: Status Tag	The <code>_DateTime</code> is a tag that indicates the GMT date and time of the system that the server is running on. The format of the string is '2004-05-21T20:39:07.000' This is a Read Only tag.

_DateTimeLocal Class: Status Tag	The _DateTimeLocal is a tag that indicates the localized date and time of the system that the server is running on. The format of the string is '2004-05-21T16:39:07.000' This is a Read Only tag.
_FullProjectName Class: Status Tag	The _FullProjectName is a tag that indicates the fully qualified path and file name to the currently loaded project. This is a Read Only tag.
_ProjectName Class: Status Tag	The _ProjectName is a tag that indicates the currently loaded project file name and does not include path information. This is a Read Only tag.
_Time Class: Status Tag	The _Time is a tag that indicates the current time of the system that the server is running on. The format of this string is defined by the operating system date/time settings. This is a Read Only tag.
_TotalTagCount Class: Status Tag	The _TotalTagCount is a tag that indicates the total number of tags that are currently being accessed. These tags can be active or inactive. Note: This count does not represent the number of tags configured in the project. This is a Read Only tag.

Channel Level system tags for a serial port driver are as follows:

Syntax example: <channel name>._System._BaudRate

_AvailableNetworkAdapters Class: Parameter Tag	The _AvailableNetworkAdapters is a tag that lists the available NICs and will include both unique NIC cards and NICs that have multiple IPs assigned to them. Additionally this tag will also display any WAN connections you may have active such as a dialup connection. This tag is provided as a string tag and can be used to determine the network adapters available for use on this PC. The string returned will contain all of the NIC names and their IP assignments. A semicolon will separate each unique NIC in order to allow the names to be parsed within an OPC application. For a serial driver this tag will only be used if Ethernet Encapsulation is selected. This is a Read Only tag.
_NetworkAdapter Class: Parameter Tag	The _NetworkAdapter tag is a tag that allows the current NIC adapter in use by the driver to be changed on the fly. As a string tag, the name of the newly desired NIC adapter must be written to this tag in string format. The string written must match the exact description of the desired NIC in order for the change to take effect. NIC names can be obtained from the _AvailableNetworkAdapters tag listed above. For a serial driver, this tag will only be used if Ethernet Encapsulation is selected. Note: When changing the NIC selection the driver will be forced to break all current device connections and reconnect. This is a Read/Write tag.
_ComId Class: Parameter Tag	The _ComId tag is a tag that allows the comm port selection for the driver to be changed on the fly. As a string tag, the desired comm port must be written to the tag as a string value using the following possible selections: COM 1, COM 2, COM 3, COM 4, - - -, COM 16, and Ethernet Encapsulation. When selecting Ethernet Encapsulation mode, you will also need to set the IP number of the remote terminal server. This is done at the device level and will be shown below. This is a Read/Write tag.
_BaudRate	The _BaudRate tag is a tag that allows the baud rate of the driver to be changed on the fly. The _BaudRate tag is defined as a long value and

Class: Parameter Tag	<p>therefore new baud rates should be written in this format. Valid baud rates are as follows: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000, 56700, 115200, 128000, and 256000.</p> <p>This is a Read/Write tag.</p>
_Parity Class: Parameter Tag	<p>The _Parity tag is a tag that allows the parity of the driver to be changed on the fly. As a string tag, the desired parity setting must be written to the tag as a string value using the following possible selections: None, Odd, and Even.</p> <p>This is a Read/Write tag.</p>
_DataBits Class: Parameter Tag	<p>The _DataBits tag is a tag that allows the data bits of the driver to be changed on the fly. The _DataBits tag is defined as an signed 8 bit value. Valid data bits selections are 5, 6, 7 and 8.</p> <p>This is a Read/Write tag.</p>
_StopBits Class: Parameter Tag	<p>The _StopBits tag is a tag that allows the stop bits of the driver to be changed on the fly. The _StopBits tag is defined as a signed 8 bit value. Valid data bit selections are 1 and 2.</p> <p>This is a Read/Write tag.</p>
_FlowControl Class: Parameter Tag	<p>The _FlowControl tag is a tag that allows the flow control setting of the driver to be changed on the fly. As a string tag, the desired flow control setting must be written to the tag in this format. Possible selections for flow control include: None, DTR, RTS, "DTR,RTS", RTS Always, and RTS Manual. Not all drivers support the RTS Manual mode of operation.</p> <p>This is a Read/Write tag.</p>
_RtsLineRaise Class: Parameter Tag	<p>The _RtsLineRaise tag is a tag that allows the RTS Line to be raised for a user-selected period of time before the driver attempts to transmit a message. This tag will only be effective for drivers that support Manual RTS mode. The _RtsLineRaise is defined as a long value. The valid range is 10 - 2550 milliseconds. The Manual RTS mode has been designed for use with radio modems.</p> <p>This is a Read/Write tag.</p>
_RtsLineDrop Class: Parameter Tag	<p>The _RtsLineDrop tag is a tag that allows the RTS Line to be lowered for a user-selected period of time after the driver attempts to transmit a message. This tag will only be effective for drivers that support Manual RTS mode. The _RtsLineDrop is defined as a long value. The valid range is 0 - 2550 milliseconds. The Manual RTS mode has been designed for use with radio modems.</p> <p>This is a Read/Write tag.</p>
_RtsLinePollDelay Class: Parameter Tag	<p>The _RtsLinePollDelay tag is a tag that allows a user-configurable pause to be placed after each message sent from the driver. This tag will only be effective for drivers that support Manual RTS mode. The _RtsLinePollDelay is defined as a long value. The valid range is 0 - 2550 milliseconds. The Manual RTS mode has been designed for use with radio modems.</p> <p>This is a Read/Write tag.</p>
_ReportComErrors Class: Parameter Tag	<p>The _ReportComErrors tag is a tag that allows the reporting of low level communications errors such as parity and framing errors to be enabled or disabled. This tag is defined as a Boolean tag and can be set either true or false. When true, the driver will report any low-level communications error to the server event system. When set false the driver will ignore the low-level communications errors and not report them. The driver will still reject a communications transaction if it contains errors. If your environment contains a lot of electrical noise, you may wish to disable this feature to prevent your event log from filling with error messages.</p>

	This is a Read/Write tag.
_EnableDiagnostics Class: Parameter Tag	The _EnableDiagnostics tag is a tag that allows the diagnostic system of the driver to be enabled and disabled. The diagnostic system places a little additional burden on the driver while enabled. As such the server allows diagnostics to be enabled or disabled to improve the driver's performance. When disabled, the Diagnostics Tags will not be available.
	This is a Read/Write tag.
_WriteOptimizationDutyCycle Class: Parameter Tag	The _WriteOptimizationDutyCycle tag is a tag that allows the duty cycle of the write to read ratio to be changed on the fly. The duty cycle controls how many writes the driver will do for each read it performs. The _WriteOptimizationDutyCycle is defined as an unsigned long value. The valid range is 1 to 10 write per read. More about write optimizations can be found here .
	This is a Read/Write tag.

Channel Level system tags for an Ethernet driver are as follows:

Syntax example: <channel name>._System._NetworkAdapter

_AvailableNetworkAdapters Class: Parameter Tag	The _AvailableNetworkAdapters is a tag that lists the available NICs and will include both unique NIC cards and NICs that have multiple IPs assigned to them. Additionally this tag will also display any WAN connections you may have active, such as a dialup connection. This tag is provided as a string tag and can be used to determine the network adapters available for use on this PC. The string returned will contain all of the NIC names and their IP assignments. A semicolon will separate each unique NIC in order to allow the names to be parsed within an OPC application. For a serial driver, this tag will only be used if Ethernet Encapsulation is selected.
	This is a Read Only tag.
_NetworkAdapter Class: Parameter Tag	The _NetworkAdapter tag is a tag that allows the current NIC adapter in use by the driver to be changed on the fly. As a string tag, the name of the newly desired NIC adapter must be written to this tag in string format. The string written must match the exact description of the desired NIC in order for the change to take effect. NIC names can be obtained from the _AvailableNetworkAdapters tag listed above. For a serial driver, this tag will only be used if Ethernet Encapsulation is selected. Note: When changing the NIC selection, the driver will be forced to break all current device connections and reconnect.
	This is a Read/Write tag.
_EnableDiagnostics Class: Parameter Tag	The _EnableDiagnostics tag is a tag that allows the diagnostic system of the driver to be enabled and disabled. The diagnostic system places a little additional burden on the driver while enabled. As such the server allows diagnostics to be enabled or disabled to improve the driver's performance. When disabled the Diagnostics Tags will not be available.
	This is a Read/Write tag.
_WriteOptimizationDutyCycle Class: Parameter Tag	The _WriteOptimizationDutyCycle tag is a tag that allows the duty cycle of the write to read ratio to be changed on the fly. The duty cycle controls how many writes the driver will do for each read it performs. The _WriteOptimizationDutyCycle is defined as an unsigned long value. The valid range is 1 to 10 write per read. More about write optimizations can be found here .
	This is a Read/Write tag.

Device Level system tags for both Serial and Ethernet drivers are as follows:

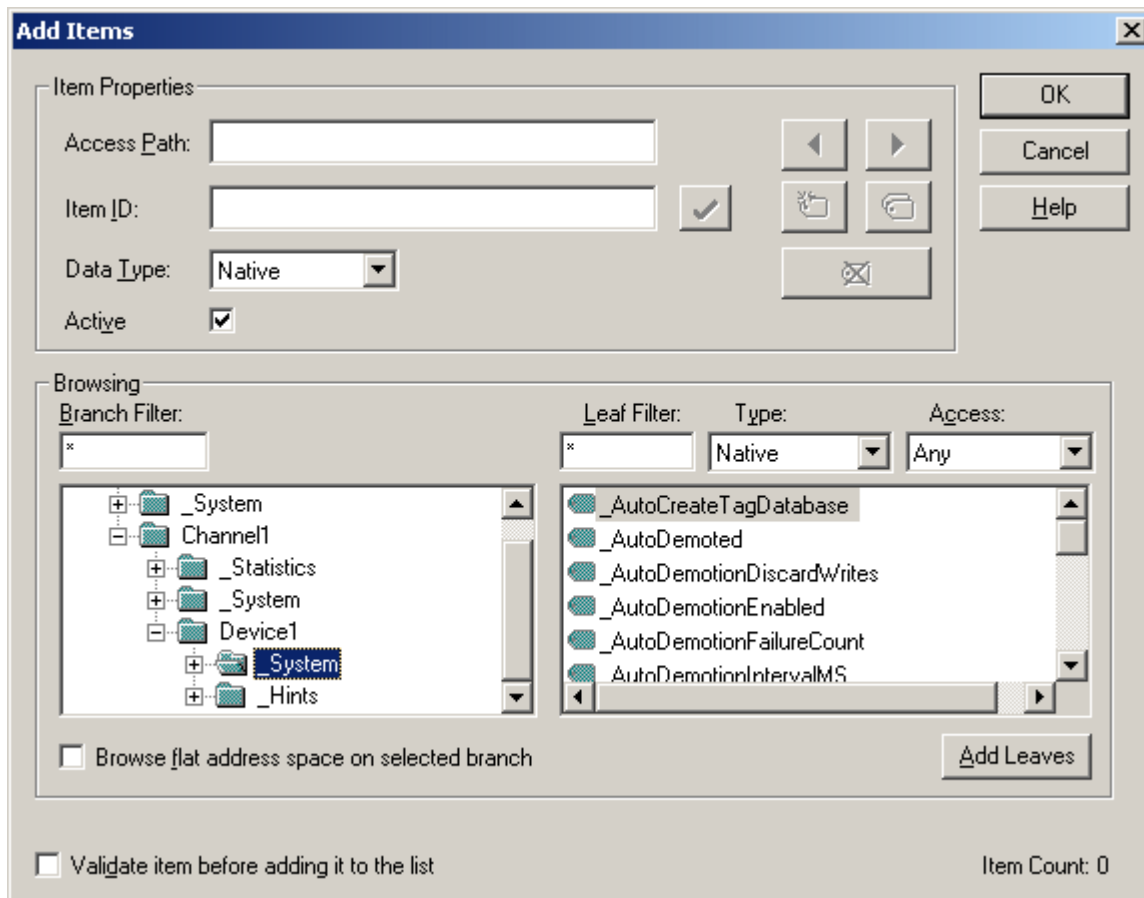
Syntax example: <Channel Name>.<Device Name>._System._Error

<p>_DeviceId</p> <p>Class: Parameter Tag</p>	<p>The <code>_DeviceId</code> tag is a tag that allows the ID of the device to be changed on the fly. The data format of the <code>_DeviceId</code> depends on the type of device. For most serial devices this tag will be a Long data type. For Ethernet drivers the <code>_DeviceId</code> will be formatted as a string tag, allowing the entry of an IP address. In either case, writing a new Device ID to this tag will cause the driver to change the target field device. This will only occur if the Device ID written to this tag is correctly formatted and within the valid range for the given driver.</p> <p>This is a Read/Write tag.</p>
<p>_ConnectTimeout</p> <p>Class: Parameter Tag</p>	<p>The <code>_ConnectTimeout</code> is a tag that allows the timeout associated with making an IP connection to a device to be changed on the fly. This tag is available when either a native Ethernet driver is in use or a Serial driver is in Ethernet Encapsulation mode. The <code>_ConnectTimeout</code> is defined as a Long data type. The valid range is 1 to 30 seconds.</p> <p>This is a Read/Write tag.</p>
<p>_RequestTimeout</p> <p>Class: Parameter Tag</p>	<p>The <code>_RequestTimeout</code> is a tag that allows the timeout associated with a data request to be changed on the fly. The <code>_RequestTimeout</code> tag is defined as a Long value. The valid range is 100 to 30000 milliseconds. This parameter tag applies to all drivers equally.</p> <p>This is a Read/Write tag.</p>
<p>_RequestAttempts</p> <p>Class: Parameter Tag</p>	<p>The <code>_RequestAttempts</code> is a tag that allows the number of retry attempts to be changed on the fly. The <code>_RequestAttempts</code> is defined as a Long value. The valid range is 1 to 10 retries. This parameter tag applies to all drivers equally.</p> <p>This is a Read/Write tag.</p>
<p>_InterRequestDelayMS</p>	<p>The <code>_InterRequestDelayMS</code> is a tag that allows the time interval between device transactions to be changed on the fly. The <code>_InterRequestDelayMS</code> is defined as a Long data type. The valid range is 0 to 30000 milliseconds. This parameter tag only applies to drivers that support this feature.</p> <p>This is a Read/Write tag.</p>
<p>_EncapsulationIp</p> <p>Class: Parameter Tag</p>	<p>The <code>_EncapsulationIp</code> tag allows the IP of a remote terminal server to be specified and changed on the fly. This parameter tag is only available on serial drivers that support Ethernet Encapsulation mode. The <code>_EncapsulationIp</code> is defined as a string data type, allowing the entry of an IP address number. The server will reject entry of invalid IP addresses. This tag is only valid for a Serial driver in Ethernet Encapsulation mode.</p> <p>This is a Read/Write tag.</p>
<p>_EncapsulationPort</p> <p>Class: Parameter Tag</p>	<p>The <code>_EncapsulationPort</code> tag allows the port number of the remote terminal server to be specified and changed on the fly. The <code>_EncapsulationPort</code> is defined as a long data type. The valid range is 0 to 65535. The port number entered in this tag must match that of the desired remote terminal server for proper Ethernet Encapsulation to occur. This tag is only valid for a Serial driver in Ethernet Encapsulation mode.</p> <p>This is a Read/Write tag.</p>
<p>_EncapsulationProtocol</p> <p>Class: Parameter Tag</p>	<p>The <code>_EncapsulationProtocol</code> tag allows the IP protocol used for Ethernet Encapsulation to be specified and changed on the fly. The <code>_EncapsulationProtocol</code> is defined as a string data type. Writing either "TCP/IP" or "UDP" to the tag specifies the IP protocol. The protocol used must match that of the remote terminal server for proper Ethernet Encapsulation to occur. This tag is only valid for a Serial driver in Ethernet Encapsulation mode.</p> <p>This is a Read/Write tag.</p>

<p>_AutoCreateTagDatabase</p> <p>Class: Parameter Tag</p>	<p>The <code>_AutoCreateTagDatabase</code> tag is a Boolean tag that allows you to initiate the automatic OPC tag database functions of this driver for the device to which this tag is attached. When this tag is set TRUE, the communications driver will attempt to automatically generate an OPC tag database for this device. This tag will not appear for drivers that do not support Automatic OPC Tag Database Generation.</p> <p>This is a Read/Write tag.</p>
<p>_Enabled</p> <p>Class: Parameter Tag</p>	<p>The <code>_Enabled</code> tag is a Boolean tag that allows the active state of the device to be turned On or Off. When this tag is set FALSE, all other user defined tags and data from this device will be marked as invalid and writes will not be accepted for the device. When this tag is set TRUE, normal communications will occur with the device.</p> <p>This is a Read/Write tag.</p>
<p>_Error</p> <p>Class: Status Tag</p>	<p>The <code>_Error</code> tag is a Boolean tag that returns the current error state of the device. When FALSE, the device is operating properly. When set TRUE, the driver has detected an error when communicating with this device. A device enters an error state if it has completed the cycle of request timeouts and retries without a response (see Device Timing Properties).</p> <p>This is a Read Only tag.</p>
<p>_NoError</p> <p>Class: Status Tag</p>	<p>The <code>_NoError</code> tag is a Boolean tag that returns the current error state of the device. When TRUE, the device is operating properly. When FALSE, the driver has detected an error when communicating with this device. A device enters an error state if it has completed the cycle of request timeouts and retries without a response (see Device Timing Properties).</p> <p>This is a Read Only tag.</p>
<p>_Simulated</p> <p>Class: Status Tag</p>	<p>The <code>_Simulated</code> tag is a Boolean tag that provides feedback about the simulation state of the current device. When read as TRUE, this device is in a simulation mode. While in simulation mode, the server will return good data for this device but will not attempt to communicate with the actual physical device. When tag is read as FALSE, communication with the physical device will be active.</p> <p>This is a Read Only tag.</p>
<p>_AutoDemoted</p>	<p>The <code>_AutoDemoted</code> tag is a Boolean tag that returns the current auto-demoted state of the device. When FALSE, the device is not demoted and is being scanned by the driver. When set TRUE, the device is in demoted and not being scanned by the driver.</p> <p>This is a Read Only tag.</p>
<p>_AutoDemotionEnabled</p>	<p>The <code>_AutoDemotionEnabled</code> tag is a Boolean tag that allows the device to be automatically demoted for a specific time period when the device is non-responsive. When this tag is set FALSE, the device will never be demoted. When this tag is set TRUE, the device will be demoted when the <code>_AutoDemotedFailureCount</code> has been reached.</p> <p>This is a Read/Write tag.</p>
<p>_AutoDemotedFailureCount</p>	<p>The <code>_AutoDemotedFailureCount</code> tag specifies how many successive failures it takes to demote a device. The <code>_AutoDemotedFailureCount</code> is defined as a long data type. The valid range is 1 to 30. This tag can only be written to if <code>_AutoDemotionEnabled</code> is set to TRUE.</p> <p>This is a Read/Write tag.</p>
<p>_AutoDemotionIntervalMS</p>	<p>The <code>_AutoDemotionIntervalMS</code> tag specifies how long, in milliseconds, a device will be demoted before re-attempting to communicate with the device. The <code>_AutoDemotionIntervalMS</code> is defined as a long data type. The valid range is 100 to 3600000 milliseconds. This tag can only be written to if <code>_AutoDemotionEnabled</code> is set to TRUE.</p>

	This is a Read/Write tag.
_AutoDemotionDiscardWrites	The _AutoDemotionDiscardWrites tag is a boolean tag that specifies whether or not write requests should be discarded during the demotion period. When this tag is set to FALSE, all writes requests will be performed regardless of the _AutoDemoted state. When this tag is set to TRUE, all writes will be discarded during the demotion period.
	This is a Read/Write tag.

When using an OPC client the system tags will be found under the _System branch of the server browse space for a given device. The following diagram taken from the supplied OPC Quick Client shows how the system tags appear to an OPC client:



The _System branch found under the DeviceName branch is always available. If referencing a system tag from a DDE application given the above example and the DDE defaults, the link would appear as follows:

= [<DDE service name>](#) |_ddedata!Channel1.Device1._System._Error

The _Enabled tag provides a very flexible means of controlling your OPC applications. In some cases, specifically in modem applications, it can be convenient to disable all devices except the device currently connected to the modem. Additionally, using the _Enable tag to allow the application to turn a particular device off while the physical device is being serviced, can eliminate harmless but unwanted communications errors in the server's event log.

Important: The _Error tag does not return TRUE when a device is set inactive via the _Enable tag or via the "Enable data collection" of the [Device ID](#) dialog.

See Also: [Statistics Tags](#) and [Property Tags](#).

Note: Modem Tags are described in the topics under [Modem Support](#).

Property Tags

The property tags are used to provide Read Only access to [Tag Properties](#) for client applications. To access a tag property, append the property name to the fully qualified tag address that has been defined in the server's tag database.

Syntax Example: If the fully qualified tag address is Channel1.Device1.Tag1, its description can be accessed by appending the description property as **Channel1.Device1.Tag1._Description**.

The following is a list of property names that are currently supported:

_Name	The _Name property tag indicates the current name for the tag it is referencing.
_Address	The _Address property tag indicates the current address for the tag it is referencing.
_Description	The _Description property tag indicates the current description for the tag it is referencing.
_RawDataType	The _RawDataType property tag indicates the raw data type for the tag it is referencing.
_ScalingType	The _ScalingType property tag indicates the scaling type (None, Linear or Square Root) for the tag it is referencing.
_ScalingRawDataType	The _ScalingRawDataType property tag indicates the raw data type for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingRawLow	The _ScalingRawLow property tag indicates the raw low range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingRawHigh	The _ScalingRawHigh property tag indicates the raw high range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingScaledDataType	The _ScalingScaledDataType property tag indicates the scaled to data type for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingScaledLow	The _ScalingScaledLow property tag indicates the scaled low range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingScaledHigh	The _ScalingScaledHigh property tag indicates the scaled high range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingClampLow	The _ScalingClampLow property tag indicates whether the scaled low value should be clamped for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingClampHigh	The _ScalingClampHigh property tag indicates whether the scaled high value should be clamped for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingUnits	The _ScalingUnits property tag indicates the scaling units for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.

See Also:

[Statistics Tags](#)

[System Tags](#)

[Property Tags](#)

Note: Modem Tags are described in the topics under [Modem Support](#).

CSV Import and Export

The server supports the import and export of tag data in a CSV (comma separated variable) file. The CSV functions are only available when a Device or Tag Group is selected. When using CSV import and export, tags are created quickly in the desired application.

Note: The [General Options](#) dialog is used to choose the character to use as the variable, either a comma (default) or a semicolon.

[Creating a Template](#)

[Exporting a Server Tag List](#)

[Note for Those Using Other Characters as the Delimiter](#)

[Importing a Server Tag List into the Server](#)

Creating a Template

The easiest way to create an import CSV file is to create a template using **File|Export CSV**. Define the channels and devices that the project will contain. Next define a tag for each device. Export each device or tag group as a CSV file. Use this template in a spreadsheet application that supports CSV files and modify the file. The resulting CSV file can then be saved to disk and imported back into the server under the same device or tag group or under a new device or tag group.

Exporting a Server Tag List

This generates a .CSV (comma separated variable) text file that contains a heading record followed by a record for each tag defined under the selected device or tag group. The heading record contains the following fields.

Tag Name - Name of the tag as it will be referenced in an OPC client.

Address - The device location referenced by the tag.

Data Type - The data type used for the tag as shown in the server Tag Data Type drop down list box.

Respect Data Type - This forces the tag to follow its defined data type not the OPC client request. (1, 0)

Client Access - Read/ Write access (RO, RW, WO)

Scan Rate - The rate in milliseconds that the tag address will be scanned when used with most non-OPC clients.

Scaling - Scaling mode (Linear, Square Root)

Raw Low - Low raw value

Raw High - High raw value

Scaled Low - Scaled low value

Scaled High - Scaled high value

Scaled Data Type - The data type used for the tag after scaling is applied.

Clamp Low - Force the resulting scaled value to stay within the limit of Scaled Low. (1, 0)

Clamp High - Force the resulting scaled value to stay within the limit of Scaled High. (1, 0)

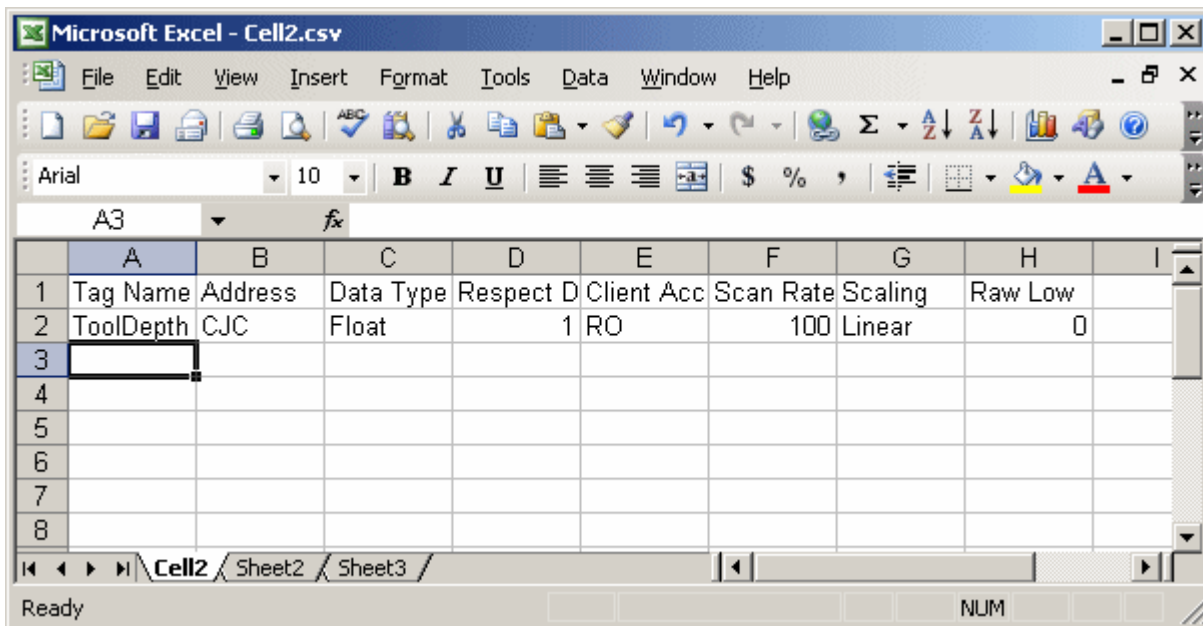
Eng. Units - Units string.

Description - Tag description

Negate Value - Forces the resulting value to be negated before being passed to the client when scaling is applied. (1, 0)

Each tag record contains the data for each field in a tag record.

Microsoft's Excel makes an excellent tool for editing large groups of tags outside of the server. Once a template CSV file has been exported it can be loaded directly into Excel for editing. A CSV file load in Excel would appear as show below.



Importing a CSV tag list into the Server

Once you have editing your tag list it can be imported back into the server using the import CSV function **File|Import CSV**. This option is available only when you have either a device or tag group selected.

When Using Other Characters as the Delimiter

When using a CSV file that uses a delimiter other than a comma or semicolon, note the following.

- With the addition of XML support, save your project in XML and perform mass configuration on the XML file instead of using CSV.

–or–

- Perform a search-and-replace on the delimiter in your CSV file and replace the delimiter with a comma or semicolon. Be sure the delimiter being used by the OPC server (either comma or semicolon) is set to the replacement character.

See Also: [General Options](#)

Project Properties

The Project Properties dialog is used to attach a title and comment to a project for reference. The title field supports a string of up to 64 characters. The comments field has no practical limit; however, limiting the comment to the area available within the comment box will improve project load time.

NetDDE: Using DDE Across a Network

DDE provides a way to share data between windows applications as long as those applications existing on the same machine. NetDDE is used to share data from a DDE server located on a local PC with DDE client applications located on remote PCs. To do this, the PC must have a unique computer name. If a network is already installed and running, then all of the computers will already have unique names. The PC's computer name can be found by looking at the network settings from the Windows Control Panel. To do so, simply double-click on the network icon. The computer name should be the first field you see. Once you have the computer name for the PC that is running the DDE server, you're ready to use that information to access DDE data from remote client PCs using the following format:

Note: NetDDE must be enabled in order to use Net DDE services with the server. For more information, refer to [DDE Options](#).

NetDDE using the default topic

The following assumptions will be made for this example: While using the Modbus Serial (RTU) driver, the user has

defined the channel as **Modbus** and has defined the station a device name of **PLC1**. The computer name is JohnDoe. In order to access register 40001:

Syntax: \\<PC name>**NDDE\$**|<topic>**\$!**<item>

Example: '\\JohnDoe\NDDE\$'|'_ddedata\$'!Modbus.PLC1.40001

Where:

application= \\<computer name>**NDDE\$**
 topic= **_ddedata** (Default topic for all DDE data not using an Alias Map entry)
alias name\$ (\$ appended to indicate a DDE share)
 item=**tag name**

See Also: [Alias Map](#)

NetDDE using alias names

As before, the following assumptions will be made for this example: While using the Modbus Serial (RTU) driver, the user has defined the channel as **Modbus** and has defined the station a device name of **PLC1**. This time, an [Alias Map](#) entry has been defined for this channel and device with a name of **ModPLC1**. The computer name to be **JohnDoe**. In order to access register 40001:

Syntax: \\<PC name>**NDDE\$**|<topic>**\$!**<item>

Example: '\\JohnDoe\NDDE\$'|'ModPLC1\$'!40001

Where:

application= **JohnDoe\NDDE\$**
 topic = **ModPLC1\$**
 item = **40001**

Notes:

1. The <> characters above are for clarity only and are not a normal part of a DDE link reference. Some DDE applications do not recognize the dollar symbol \$ as valid character in their DDE links.
2. When making a NetDDE connection from a DDE application to the Server, you may need to enclose each section of the formula in single quotes ' ' due to its use of special characters. Refer to the DDE client's specific help information for syntax guidelines.
3. When overriding the default configuration of a dynamic tag from a DDE application, you will need to place the entire item name in single quotes. Like the dollar symbol, the at @ symbol is not normally recognized as a valid character in a formula.

Server Configuration

To use NetDDE, the server needs to be told to use NetDDE services before connecting from a remote client. To enable NetDDE services, select the **Tools|Options DDE** tab in the server. Then, click the **Enable Net DDE** check box to enable Net DDE.

For optimum server performance under Windows 95 and Windows NT versions previous to 4.0, do not enable NetDDE services unless they are needed.

Client Configuration

To use NetDDE, the NetDDE services may need to be manually started on the client computer before running the client application. From Windows NT based operating systems, use the Services Applet from the Control Panel. From Windows95, run the program **NETDDE.EXE**.

DDE Shares

If NetDDE services are enabled, a DDE share is created each time a client application connects to the server. When the server is taken off line, these shares are removed.

Note: If you see a warning message indicating failure to create a DDE share for a given device it probably means that a share of that name already exists. If the server created the share, then the message is harmless. If the share was created by another application, then the device name you assigned cannot be used and must be renamed or the existing DDE share should be removed.

Removing a DDE Share

To remove a DDE share under Windows NT, use the program **DDESHARE.EXE**. To remove a DDE share under Windows95, remove the share name from the registry under **HKEY_LOCAL_MACHINE\Software\Microsoft\NetDDE\DDE Shares**. Please consult the documentation for the appropriate operating system prior to attempting this operation.

XP Service Pack 2 Configuration

XP with Service Pack 2 sets **Network DDE DSDM** and **Network DDE** services to Disabled by default. To use remote connectivity, enable these services by opening the Control Panel and then opening **Administrative Tools**. Next, double-click **Services** and then scroll down to **Network DDE DSDM** service. Right-click and choose **Properties**. Then, change the **Startup Type** to 'Manual', and click **OK**. Repeat this procedure with the **Network DDE** service. This procedure sets up services allowing the DDE clients to connect.

Note: With 'Manual' settings, the services must be started each time they will be used. The DDE client may start them automatically.

DDE: Using DDE in the application

DDE, or **Dynamic Data Exchange**, is a Microsoft communications protocol that provides a method for exchanging data between applications running on a windows operating system. The DDE client program opens a channel to the DDE server application and requests item data using a hierarchy of three names:

- Application (Service) name
- Topic name
- Item name

Example 1: Accessing a Register Locally using the Default Topic

Syntax: <application>|<topic>!<item>

Example: MyDDE|_ddedata!Modbus.PLC1.40001

Where:

application = [DDE service name](#)

topic = **_ddedata** (Default topic for all DDE data not using an [Alias Map](#) entry)

item = **Modbus.PLC1.40001**

Example 2: Accessing a Register Locally using an Alias Name as a Topic

Syntax: <application>|<topic>!<item>

Example: MyDDE|ModPLC1!40001

Where:

application = [DDE service name](#)

topic = **ModPLC1** (Topic now using [Alias Map](#) entry)

item = **40001**

See Also:

[DDE Options](#)

[NETDDE](#)

[FastDDE & SuiteLink](#)

[Alias Name](#)

Note 1: Please refer to your DDE client's specific help information for possible additional syntax.

Note 2: To connect to remote applications using DDE, see [NetDDE: Using DDE Across a Network](#).

Processing Array Data in a DDE Client

Many of the drivers available for the server allow clients to access data in an array format. However, array data is only available to the client when using CF_TEXT or AdvancedDDE clipboard formats.

For client applications using AdvancedDDE, the number of elements in the array is specified in the

sPACKDDE_DATAHDR_TAG struct. Only single dimensional arrays are supported by this protocol. This structure should be used when poking array data to the server.

For clients using CF_TEXT, one or two-dimensional arrays are supported. Data in each row is separated by a TAB (0x09) character and each row is terminated with a CR (0x0d) and a LF (0x0a) character. When a client wants to poke an array of data values, the text string written should have this delimiter format.

When poking to an array tag in either format, the entire array does not need to be written, but the starting location is fixed. If you attempt to poke data in an array format to a tag that was not declared as an array, only the first value in the array will be written. If you attempt to poke more data than the tag's array size, only as much data as the tag's array size will be written.

OPC Data Exchange Plug-in

The OPC Data Exchange plug-in is used to add OPC Data Exchange (OPC-DX) support to the server.

OPC Data Exchange (OPC-DX) is a standard that builds upon the existing OPC Data Access (OPC-DA) standard. OPC-DA provides interfaces that allow for the transfer of data between an OPC-DA server and an OPC-DA client. Typically, OPC-DA clients subscribe to one or more OPC-DA servers to acquire data, and OPC-DA servers provide data to one or more OPC-DA clients. Over time, and with the large acceptance of OPC-DA by the market, end-users demanded the ability not only to share data between clients and servers, but to share data between one or more servers. Several companies developed a middleware solution to fill this need. This solution involved a bridging technology where the middleware acts as a client to one or more servers, and using standard OPC-DA, reads data from one server (source server) and writes the data to another server (target server), creating what is now referred to as a connection. While this solution worked, it had a couple of pitfalls. The middleware adds a new point-of-failure in the communications system, as it is yet another component the end-user must maintain and diagnosis when errors occur. The other issue is that the end-user must adhere to the configuration interface of the middleware supplier. OPC-DX was created to address these needs.

OPC-DX defines two key components. The first component is how an OPC-DX server (target) should acquire data from a source server, by embedding the appropriate OPC-DA client functionality inside an OPC-DA server. This eliminates the middleware component as a runtime dependency. The second component is an open-standard for clients to configure one or more connections within an OPC-DX server.

The OPC-DX plug-in can be installed using the server installation program. The installation will provide the OPC-DX plug-in which enables the server to run as a DX server. The installation also provides a Data Exchange Client for configuring the DX server. The Data Exchange Client can be used with any OPC-DX server.

Since DX connections depend on target items defined in a given server project, each server project (e.g., project.opf) will have a unique DX configuration file (e.g., project.opf.dxconfig). When DX is enabled, opening a new project file will stop the DX functionality for the currently loaded project (e.g., stop all DX connections associated with the currently loaded DX configuration). Upon loading the new project (assuming the new project has previously been configured with DX connections), the DX configuration for the new project file will be loaded and will startup its configured DX connections.

If the server receives source data updates faster than it can write the data to the target, the write optimization settings will need to be set to 'Write only latest value for all tags' to minimize memory consumption. For more information, refer to [Write Optimizations](#).

OPC-DX support can be disabled at anytime through the [OPC-DX Options](#) dialog.

For more information on OPC-DX and how to configure and utilize this technology, please consult the Data Exchange Client help file that is available when the OPC-DX plug-in component is installed.

Modem Support

The server supports the use of modems to connect to remote devices. This is established through the use of special modem tags that become available at the channel level when a dial-up network connection has been created. These channel level modem tags can be used to dial a remote device, monitor the modem status while connected, and terminate the call when completed. **Not all serial drivers support the use of modems.** Check the driver help file to determine whether modems are supported or not.

To access the new modem tags, the channel name can now be used as either a base group or topic name. If your project contains more than one channel definition you will need to configure the channel names so that each channel name is unique. This is also the case with all device names. Channel names can no longer match the device name when

the project needs to be configured to use a modem. These channel name requirements do not apply to projects that are not using a modem.

To use a modem you need to configure a modem with the operating system. Modems can be configured through control panel settings. Consult your Windows and modem documentation on how to set up your modem. Once the modem has been properly installed, you can enable its use by selecting the **Use Modem** checkbox using the [Channel Wizard](#).

Important: Many new commercial modems are designed to dial-up network server connections and negotiate the fastest and clearest signal. When communicating to a serial automation device, the modem needs to connect at a specific Baud (Bits per Second) and Parity. For this reason, we strongly recommend the use of an **external modem**, which can be configured to dial using specific Baud Rate and Parity settings. Check with technical support to determine which modem will best suit your needs.

Using a Modem in Your Server Project

Modems convert the serial data from your RS-232 port into signal levels that can be transmitted over the phone line. To do this, modems take the serial data given to them and break each byte of the data down into bits used to generate the signal to be transmitted. Most modems can handle converting up to ten bits of information for every byte of data that they send. To communicate with the device via a modem, it must be able to use 10 bits or less. You can determine how many bits the device is using from this formula:

Start Bits + Data Bits + Parity + Stop Bits=Total Bit Count

Using the Modbus RTU driver as an example let's assume that the Modbus device is configured to use 8 Data Bits, Even Parity, and 1 Stop Bit, (1 Start bit is always implied). Using the formula we would have $1 + 8 + 1 + 1 = 11$ Bits. A normal modem would not be able to transmit data to this Modbus device using these settings. If we change the Parity to None the result becomes $1 + 8 + 0 + 1 = 10$ Bits. Now a normal modem can be used to transmit data to the Modbus device.

From these examples you may be wondering if there are some drivers, which simply cannot use a normal modem connection. Some devices do not allow you to configure them to use a 10 bit or less data format. These devices simply cannot use standard off the shelf modems. There are manufacturers who provide modems that can handle transmitting 11 data bits. If your driver falls into this category please consult the device manufacturer for recommendations on an appropriate modem vendor. Modem operation will be enabled for all serial drivers regardless of their suitability for modem operation. See your driver's help file for more specific information regarding modem support.

Now that we know what a modem does and how a device needs to be configured to use a modem, let's start by not using the modem. Create a project making sure to set your [channel](#) settings to use 10 bits or less for the connection. Configure the device for the same settings.

Configuring the Initiating Modem

The server uses the Windows TAPI interface to access modems attached to the PC. The TAPI interface was designed to provide Windows programs a common interface that could be accessed by a range of modems existing in a PC. A set of drivers for the Windows OS, which are provided by the modem's manufacturer, must be installed before the server can use the modem in a project. The Windows Control Panel can be used to install new modems. Consult both the Windows and modem documentation for information regarding modem installation and setup.

Once you have your modem properly installed you're ready to begin using it in a server project but let's start on the receiving end (device modem) first. As I said before there's a lot of magic that goes into a modem connection. Getting the receiving modem properly configured can mean the difference between success and hours of failure. Each driver contains a Modem configuration page in its help file. This page contains a listing of the receiving modem's active profile. You'll need to confirm that your receiving modem matches the profile provided by your driver. Your next thought may be "How do I configure the receiving modem?".

Configuring the Receiving Modem

The Hyperterminal program that is included with Windows can be used to easily configure the Receiving Modem.

1. Connect the desired receiving modem to the PC by using an available serial port. Start **Hyperterminal** and open a new connection. Give this new connection a name similar to **ModemSetup**.
2. The **Connect To** dialog will pop up. In the **Connect Using** drop-down menu, select the communication port that the receiving modem is attached to. Other modems may appear in this list, but should be ignored at this moment.
3. In the **COMx Properties** dialog, configure the communications port settings that will be used to talk to the receiving modem. **Important:** The COMx Properties settings must match the Baud Rate, Data Bits, Parity, and Stop Bits used by your target device. Modems remember the Baud Rate, Data Bits, Parity, and Stop Bits that were used to talk to it last;

thus, if the receiving modem was configured at 19,200 baud but the device was configured for 9600 baud, the modem will never be able to speak to the device. Although it could connect, the receiving modem would send all the data to the device at 19,200 baud. This is true even if the modem connects at 9600 baud or if the transmitting modem is being spoken to at 9600 baud. Any disparity between the settings will cause the modem application to fail. To avoid the error, match the settings between the newly created server project and one that has a direct cable connection.

5. Enter the **port settings**. Then, click **OK**.

Note: At this point, you should be able to issue commands to the receiving modem. On many modems, this can be tested by typing **ATI4**, followed by **Enter**. Check the modem manufacturer's documentation to be sure that this is a valid test for your specific modem. If the modem is properly attached to the PC, it will respond by displaying its current profile settings.

Set the desired profile for your receiving modem; afterwards, save the settings by issuing a write command to the modem. To do so, type **AT&W0** followed by **Enter**. To test the receiving modem's configuration afterwards, simply turn it off for a moment and then turn it back on. Pause, and then type **AT14** followed by **Enter** (or another applicable command for your modem). The modem should display its current profile, including any changes that have been made.

Important: The profile settings and reference documents provided are to be used as examples. All settings used will need to be verified by the modem manufacturer because different configuration commands and codes may be used.

Sample Receiving Modem Profile

The follow are the USRobotics Sportster 33600 Fax Settings.

```
B0 E0 F1 M1 Q1 V1 X4 Y0
BAUD=9600 PARITY=N WORDLEN=8
DIAL=HUNT ON HOOK CID=0
```

```
&A3 &B1 &C1 &D0 &G0 &H0 &I0 &K2
&M4 &N0 &P0 &R1 &S0 &T5 &U0 &Y0
```

```
S00=001 S01=000 S02=043 S03=013 S04=010 S05=008 S06=002
S07=060 S08=002 S09=006 S10=014 S11=070 S12=050 S13=000
S15=000 S16=000 S18=000 S19=000 S21=010 S22=017 S23=019
S25=005 S27=000 S28=008 S29=020 S30=000 S31=128 S32=002
S33=000 S34=000 S35=001 S36=014 S38=000 S39=000 S40=001
S41=000 S42=000
LAST DIALED #:
```

Cables

Just when you thought everything was ready to go there's still one more detail that must be done correctly if you want success. You need to have the correct cable connection between your receiving modem and the device. The connection to your modem is normally done with what is called a NULL modem cable. A NULL modem cable means that all the pins are connected to the same pins on both ends of the cable. The cable you use to connect to your target device usually has pins 2 and 3 reversed. Since you know the cable you use to talk to the device for the direct connection is working properly you can use it on your receiving modem by attaching a NULL modem adapter. NULL modem adapters can be found at most computer stores. While you are at the computer store pickup a PC modem cable. This will go from your PC to your initiating modem. To summarize, the cables you will need are: your existing device communication cable for direct connection, a null modem adapter, and a null modem cable. With the cables in place you are now ready to use a modem in the application.

Configuring a Server Modem Project

Once your modems have been properly configured and installed, you are now ready to enable their use with the server. Let's start with the direct connect project you did earlier when testing the device configuration and communications. After loading your direct connect project, double-click on the Channel Name. The first thing you should be presented with is the current direct connect settings of Baud Rate, Parity, etc. At the bottom of this dialog there is a checkbox selection called Use Modem. Click this checkbox and you will notice that the settings on this page now become unavailable. With the Use Modem checkbox set, now click on the Modem Tab at the top of this dialog. The dialog will change to show you a list of modems available on your computer. If you don't see any modems in the list you will need to exit the server and attempt to reinstall your modem using the modem configuration tools supplied with the operating system.

With the modem list displayed, select the modem you intend to use with your server project. Once you select a modem you will notice that the Dialing... and Properties... buttons become available. Click on the properties button. At this point you will be presented with a new dialog that allows you to configure the characteristics of your initiating modem.

This new dialog has three sections available. The first, General, allows you set connection speed and speaker volume. We recommend that you set the connection speed to match the speed used by your target device. Simply set this to match the setting used when this was a direct connect project. The next section, Connection, allows you to configure Data Bits, Parity, and Stop Bits. Once again we recommend that you configure these settings to match the target device settings. The other options on this page can be left at their defaults. At the bottom of the Connection page is the Advanced... button. Click this button. The Advanced Connection dialog allows you to configure error correction and flow control for the initiating modem. Consult your driver help for the recommended settings. In most cases we recommend that you enable error correction and data compression. Flow control can go either way, we did enable it during testing and development. Once the settings on the Advanced Connections page have been set you can finish the initiating modem configuration by clicking the OK button on each dialog until you are back at the modem list within the server. Once you are back at the modem list in the server you can click the OK button, seen when editing an existing project, or the Next button when editing a new project. Assuming at this point you have just edited an existing direct connect project your server project is now ready for modem operation. Save the project.

Using a Modem in the application

When you enabled modem operation for your driver you may have noticed that a list of predefined tags became available in your driver's tag window. These special modem tags are contained under the channel name. The channel name becomes significant as it now becomes an active OPC Access path used to access the modem tags. The modem tags (See descriptions below) allow you to control and monitor an attached modem. Operationally the server knows very little about what you or the application may need for modem control. With this in mind the server does not imply any type of control over the modem. Using the predefined modem tags you can use the control or scripting capabilities of your client application to control the server's use of the selected modem.

The following tags are created automatically for the channel when modem use is selected.

[Dial](#)
[DialNumber](#)
[Hangup](#)
[LastEvent](#)
[Mode](#)
[PhoneNumber](#)
[Status](#)
[StringLastEvent](#)
[StringStatus](#)

Additional [Phonebook](#) tags can be created for the channel.

To dial a number simply write the number to be dialed to the [PhoneNumber](#) tag. The value written should be a complete phone number. Once the number to be dialed has been loaded, write any value to the [Dial](#) tag. The Dial tag will then attempt to dial the phone number that has been loaded into the PhoneNumber tag. Once you have initiated a dial operation you can use the [Status](#) tag to monitor the condition of your modem link. Once the server dials a number and establishes a link to your remote device it will release the line to the driver and allow normal driver communications to occur over the modem link. Once the application has gathered the information it needs from the remote site, you can hang-up the modem by writing any value to the [Hangup](#) tag. All modem operation is completely under the control of your client application. Using the control and scripting capabilities of your client application you can create modem applications that are as simple or complex as you require.

See Also:

[Statistics Tags](#)
[System Tags](#)
[Property Tags](#)

Dial (Modem Tag)

Data Type	Privilege
Long	Read/Write

Writing any value to this tag initiates dialing of the current [PhoneNumber](#). The write is ignored unless the current [Status](#) is 3 (Idle). An error is reported if the is current phone number has not been initialized. Attempting to issue a dial command while the [Mode](#) tag is set to 2, incoming call only, will generate an error.

DialNumber (Modem Tag)

Data Type	Privilege
String	Read Only

"DialNumber" tag shows the phone number that is actually dialed, after any dialing preference translations have been applied such as the addition of an area code. This tag is intended for debugging purposes. It can provide useful feedback to an operator if phone numbers are entered manually.

Hangup (Modem Tag)

Data Type	Privilege
Long	Read/Write

Writing any value to this tag hangs up the current connection. The "Hangup" tag will also hang-up the current connection when an external device has called the server. Writes to the "Hangup" tag will be ignored if the [Status](#) <= 3 (Idle) meaning that there is no currently open connection.

LastEvent (Modem Tag)

Data Type	Privilege
Long	Read Only

Whenever the [Status](#) changes, the reason for the change is set in this tag as a number.

LastEvent	Reason for change
-1	<blank> [no events have occurred yet]
0	Initialized with TAPI
1	Line closed
2	Line opened
3	Line connected
4	Line dropped by user
5	Line dropped at remote site
6	No answer
7	Line busy
8	No dial tone
9	Incoming call detected
10	User dialed
11	Invalid phone number
12	Hardware error on line caused line close

Mode (Modem Tag)

Data Type	Privilege
Long	Read/Write

This allows for configuring the line for calling only, answering only, or both.

Writing a 1 to the Mode tag sets the line for outgoing calls only, no incoming calls will be answered when in this mode.

Writing a 2 to the Mode tag sets the line for incoming calls only, requests to dial out (writes to the Dial tag) are ignored. The default setting is 0, which allows for both outgoing and incoming calls.

This value can only be changed when the [Status](#) is ≤ 3 (Idle).

PhoneNumber (Modem Tag)

Data Type	Privilege
String	Read/Write

This is the current phone number that will be dialed. This value can be written to at any time, but the change is only effective if [Status](#) is ≤ 3 (Idle). If the phone number is written to while the status is greater than 3, the number is queued and as soon as the status drops to 3 or less, the new number will be transferred to the tag. The queue is of size 1 so only the last phone number written is remembered.

If users wish to apply the dialing preferences, the phone number should be in canonical format. If the canonical format is used, the resulting number that will be dialed (after dialing preferences have been applied) can be displayed as the [DialNumber](#).

Canonical format is the following:

+<country code>[space](<area code>)[space]<phone number>

example: +1 (207) 846-5881

Note: The country code for the U.S. is 1.

If the number is not in canonical form, dialing preferences will not be applied. The number will be dialed exactly how it is entered.

Note: A Phonebook tag name can be entered instead of a phone number. In this case, the current value of the phonebook tag will be used.

Status (Modem Tag)

Syntax example: <Channel Name>._Modem._Status

Data Type	Privilege
Long	Read Only

This is the current status of the modem assigned to a channel.

Currently only the five lowest bits of the 32 bit status variable are used:

Bit	Meaning
0	Initialized with TAPI
1	Line open
2	Connected
3	Calling
4	Answering

The **value** of the status tag when read as an integer will always be one of the following:

Value	Meaning
0	Un-initialized, the channel is not usable
1	Initialized, no line open
3	Line open and the state is idle
7	Connected
11	Calling

19	Answering
----	-----------

StringLastEvent (Modem Tag)

Data Type	Privilege
String	Read Only

This contains a textual representation of the [LastEvent](#) tag value.

LastEvent	StringLastEvent
-1	<blank> [no events have occurred yet]
0	Initialized with TAPI
1	Line closed
2	Line opened
3	Line connected
4	Line dropped by user
5	Line dropped at remote site
6	No answer
7	Line busy
8	No dial tone
9	Incoming call detected
10	User dialed
11	Invalid phone number
12	Hardware error on line caused line close
13	Unable to dial

StringStatus (Modem Tag)

Data Type	Privilege
String	Read Only

This contains a textual representation of the [Status](#) tag value.

Status value	StringStatus text
0	Uninitialized, channel is unusable
1	Initialized, no line open
3	Idle
7	Connected
11	Calling
19	Answering

Phonebook Tags

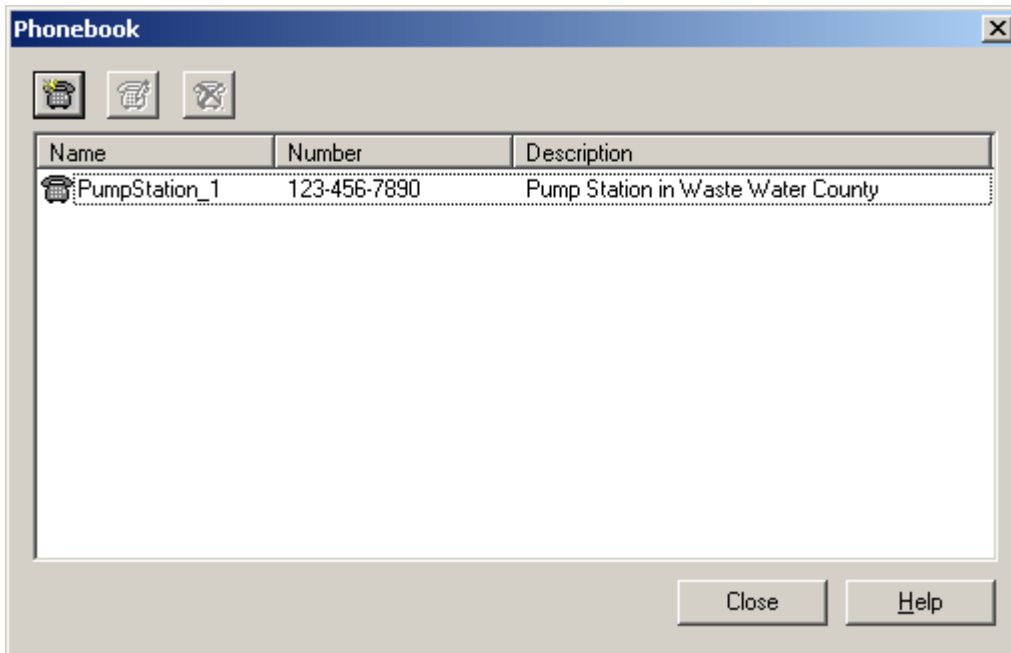
Syntax example: <Channel Name>._Phonebook.<Phonebook Tag Name>

Data Type	Privilege
String	Read/Write

Instead of specifying a telephone number by directly writing to the [PhoneNumber](#) tag, a phonebook tag can be used. A phonebook tag can be created on the channel, along with the other modem system tags previously described. The

data associated with a Phonebook tag is a phone number that can be assigned when the tag is created and/or later modified when the server has an active client connection. The phone number stored in a phonebook tag can be used to dial by simply writing anything to the tag. The act of writing will cause the selected phonebook tag to dial.

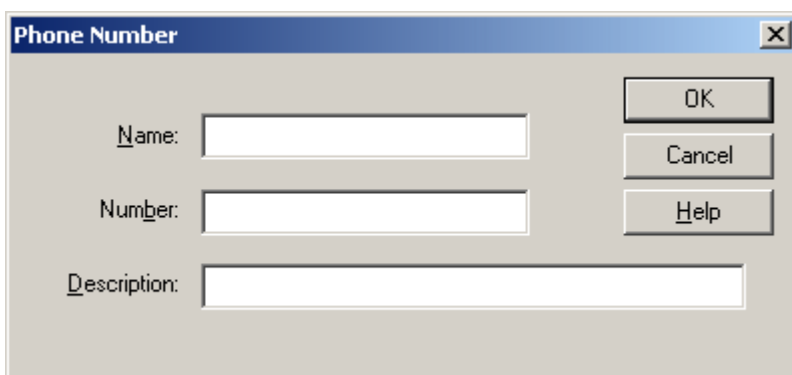
Phonebook tags are entered using the following dialog:



To add a new phonebook tag, simply click on the New Phonebook icon to display the [Phone Number](#) dialog.

Phone Number

The Phone Number dialog is used to enter a new Phonebook tag. Each phonebook tag can then be used to dial a desired phone number. If the OPCclient application can not store the phone number for a device location, using phonebook tags keeps the list in the server. To invoke a phonebook tag, the OPC client must write any string value to the desired phonebook tag. The phone number dialog should appear as shown below.



The **Name** parameter is used to enter the string that will represent the phone number available from this phonebook tag. The name can be up to 256 characters in length. While using long descriptive names is generally a good idea, keep in mind that some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The phonebook tag name is part of the OPC browse data Phonebook tag names must be unique within a given device.

The **Number** parameter is used to enter the phone number that will be dialed with this tag is invoked from an OPC client application. A string of up to 64 digits can be entered for the phone number.

The **Description** parameter is used to attach a comment to this tag. A string of up to 64 characters can be entered for the description.

Note: With the server's online full time operation, users can change any of these parameters at any time. Changes made to Tag Properties will take effect immediately, however OPC clients that have already connected to this tag will not be affected until they release and reacquire this tag. To prevent operators from changing these parameters, use the [User Manager](#) to restrict access rights to server features.

Quick Reference for USRobotics Sportster® 33.6 kbps internal and external modems

The following information has been taken directly from the manual of a USRobotics Sportster® 33.6 kbps with the permission of 3Com Corporation.

This section includes information about:

- Front Panel Lights
- Command Summary
- DIP Switches
- S-Registers
- The Serial Interface (cable information)

Front Panel Lights (External Modems)

Symbol Meaning Status

AA Auto Answer Answer mode: ON when register S0 is set to 1 or higher (Auto Answer), and when answering a call; OFF when modem originates a call. Light flashes when there is an incoming call.

CD Carrier Detect ON if modem receives a valid data signal (carrier) from a remote modem, indicating that data transmission is possible. Always ON if CD override is ON (&C0).

RD Received Data Flashes when modem sends result codes or passes received data bits from remote.

SD Send Data Flashes when computer sends a data bit to the modem.

TR Data Terminal Ready ON if modem receives a DTR signal from computer. Always ON (modem ignores DTR) if the DTR override is ON (&D0).

CS Clear to Send ON until modem lowers CTS when Transmit Data hardware flow control is enabled (&H1, &H3).

ARQ/ Error Control/

FAX Fax Operations Data Mode: Automatic Repeat Request. ON if modem is set to &M4 or &M5 and successfully establishes an error control connection. Flashes when modem retransmits data to remote modem. Fax Mode: Flashes to indicate fax mode.

Command Summary

Type commands in either upper or lower case, not a combination. Use the Backspace key to delete errors. (You cannot delete the original AT command since it is stored in the modem buffer.) If a command has numeric options and you don't include a number, zero is assumed. For example, if you type **ATB**, the command **ATB0** is assumed. Every command except A/ and +++ must begin with the AT prefix and be entered by pressing **ENTER**. The maximum command length is 58 characters. The modem doesn't count the AT prefix, carriage returns, or spaces.

NOTE: All defaults are based on the &F1 Hardware Flow Control template loaded in NVRAM when the modem is shipped. Defaults are listed in *italics*

Command Set

\$ Use in conjunction with *D*, *S*, or *&* commands (or just AT) to display a basic command list; online help.

A Manual Answer: goes off hook in answer mode. Pressing any key aborts the operations.

A/ Re-executes the last issued command. Used mainly to redial. This does not require the AT prefix or a Carriage Return.

Any key Aborts off-hook dial/answer operation and hangs up.

AT Required command prefix, except with **A/** and **+++**. Use alone to test for OK result code.

Bn U.S./ITU-T answer sequence.

B0 ITU-T answer sequence

B1 U.S. answer tone

Dn Dials the specified phone number. Includes the following:

L *Dials the last dialed number.*

P Pulse (rotary) dial

R Originates call using answer (reverse) frequencies.

Sn Dials the phone number string stored in NVRAM at position n ($n=03$). Phone numbers are stored with the $\&Zn=s$ command.

T Tone dial

, (Comma) Pause, See S8 definition; which it's linked to.

; (Semicolon) Return to Command mode after dialing.

" Dials the letters that follow (in an alphabetical phone number).

! (Exclamation point) Flashes the switch hook.

/ Delays for 125 ms. before proceeding with dial string.

W Wait for second dial tone (X2 or X4); linked to S6 register.

@ Dials, waits for quiet answer, and continues (X3 or higher).

\$ Displays a list of Dial commands.

En Sets local echo.

EO Echo OFF

E1 Modem displays keyboard commands

Fn Sets online local echo of transmitted data ON/OFF.

FO Local echo ON. Modem sends a copy of data it sends to the remote system to your screen.

F1 Local echo OFF. Receiving system may send a remote echo of data it receives.

Hn Controls ON/OFF hook.

H0 Hangs up (goes on hook).

H1 Goes off hook.

In Displays the following information.

I0 Four-digit product code

I1 Results of ROM checksum

I2 Results of RAM checksum

I3 Product type

I4 **Current modem settings**

I5 Nonvolatile memory (NVRAM) settings

I6 Link diagnostics

I7 Product configuration

Ln Controls speaker volume (internals only).

L0 Low volume

L1 Low volume

L2 *Medium volume*

L3 High volume

Mn Operates speaker.

M0 Speaker always OFF.

M1 Speaker ON until CONNECT.

M2 Speaker always ON.

M3 Speaker ON after dial, until CONNECT.

On Returns online.

O0 Returns online.

O1 Returns online and retrains.

P Sets pulse dial (for phone lines that don't support touch-tone dialing).

Qn Displays/suppresses result codes.

Q0 Displays result codes.

Q1 Quiet mode; no result codes.

Q2 Displays result codes only in Originate mode.

Sr.b=n Sets bit $.b$ of register r to n (0/OFF or 1/ON).

Sr=n Sets register r to n .

Sr? Displays contents of S-Register r .

S\$ Displays a list of the S-Registers.

T Sets tone dial.

Vn Displays verbal/numeric result codes.

V0 Numeric codes

V1 Verbal codes

Yn **Selects power-on/reset default configuration.**

Y0 Default is profile 0 setting in NVRAM

Y1 Default is profile 1 setting in NVRAM

Z Resets modem.

Z0 *Resets modem to NVRAM profile selected by Y command or dip 7.*

Z1 Resets modem to NVRAM profile 0

Z2 Resets modem to NVRAM profile 1

Z3 Resets modem to factory default profile 0 (&F0)
Z4 Resets modem to factory default profile 1 (&F1)
Z5 Resets modem to factory default profile 2 (&F2)
&\$ Displays a list of ampersand (&) commands.
&A Enables/disables additional result code subsets (see *Xn*).
&A0 ARQ result codes disabled
&A1 ARQ result codes enabled
&A2 V.32 modulation indicator added
&A3 Protocol indicators added LAPM/MNP/NONE (error control) and V42bis/MNP5 (data compression)
&Bn Manages modem's serial port rate.
&B0 Variable, follows connection rate
&B1 Fixed serial port rate
&B2 Fixed in ARQ mode, variable in non-ARQ mode
&Cn Controls Carrier Detect (CD) signal.
&C0 CD override
&C1 Normal CD operations
&Dn Controls Data Terminal Ready (DTR) operations.
&D0 DTR override
&D1 DTR toggle causes online Command mode
&D2 Normal DTR operations
&D3 Resets on receipt of DTR
&Fn Loads a Read Only (non-programmable) factory configuration.
&F0 Generic template
&F1 Hardware flow control template
&F2 Software flow control template
&Gn Sets Guard Tone.
&G0 No guard tone, U.S. and Canada
&G1 550 Hz guard tone, some European countries, requires B0 setting.
&G2 1800 Hz guard tone, U.K., requires B0 setting.
&Hn Sets Transmit Data (TD) flow control (See Also &Rn).
&H0 Flow control disabled
&H1 Hardware flow control, Clear to Send (CTS)
&H2 Software flow control, XON/XOFF
&H3 Hardware and software flow control
&In Sets Receive Data (RD) software flow control (See Also &Rn).
&I0 Software flow control disabled
&I1 XON/XOFF signals to your modem and remote system
&I2 XON/XOFF signals to your modem only
&Kn Enables/disables data compression.
&K0 Data compression disabled
&K1 Auto enable/disable
&K2 Data compression enabled
&K3 MNP5 compression disabled
&Mn Sets Error Control (ARQ) for connections at 1200 bps and higher.
&M0 Normal mode, error control disabled
&M1 Reserved
&M2 Reserved
&M3 Reserved
&M4 Normal/ARQ
&M5 ARQ mode
&Nn Sets connect speed. If connection cannot be established at this speed, the modem will hang up. Sets ceiling connect speed if &Un is greater than 0. See &Un.
&N0 *Variable rate*
&N1 300 bps
&N2 1200 bps
&N3 2400 bps
&N4 4800 bps
&N5 7200 bps
&N6 9600 bps
&N7 12,000 bps
&N8 14,400 bps
&N9 16,800 bps
&N10 19,200 bps
&N11 21,600 bps
&N12 24,000 bps
&N13 26,400 bps

&N14 28,800 bps
&N15 31,200 bps
&N16 33,600 bps
&Pn Sets pulse (rotary) dial make/break ratio.
&P0 U.S./Canada ratio, 39%/61%
&P1 U.K. ratio, 33%/67%
&Rn Sets Receive Data (RD) hardware flow control, Request to Send (RTS) (See Also &Hn).
&R0 Reserved
&R1 Modem ignores RTS
&R2 Received Data to computer only on RTS
&Sn Controls Data Set Ready (DSR) operations.
&S0 DSR override; always ON
&S1 Modem controls DSR
&Tn Begins test modes (used in conjunction with s register 18).
&T0 Ends testing
&T1 Analog Loopback
&T2 Reserved
&T3 Local Digital Loopback
&T4 Enables Remote Digital Loopback
&T5 Prohibits Remote Digital Loopback
&T6 Initiates Remote Digital Loopback
&T7 Remote Digital with self-test and error detector
&T8 Analog Loopback with self-test and error detector
&Un Sets floor connect speed when &Un is set greater than 0. &Nn is the ceiling connect speed. See &Nn.
&U0 *Disabled*
&U1 300 bps
&U2 1200 bps
&U3 2400 bps
&U4 4800 bps
&U5 7200 bps
&U6 9600 bps
&U7 12,000 bps
&U8 14,400 bps
&U9 16,800 bps
&U10 19,200 bps
&U11 21,600 bps
&U12 24,000 bps
&U13 26,400 bps
&U14 28,800 bps
&U15 31,200 bps
&U16 33,600 bps
&Wn Writes current configuration to NVRAM templates.
&W0 Modifies the NVRAM 0 template (Y0)
&W1 Modifies the NVRAM 1 template (Y1)
&Yn Sets break handling.
&Y0 Destructive, but doesn't send break
&Y1 Destructive, expedited
&Y2 Nondestructive, expedited
&Y3 Nondestructive, unexpedited
&Zn=s Writes phone number string s to NVRAM at position n (n=03).
&Zn=L Writes last executed dial string to NVRAM at position n (n=03).
&Zn? Displays the phone number stored at position n (n=03).
&ZL? Displays the last executed dial string.
#CID=n Controls Caller ID feature.
#CID=0 Caller ID disabled.
#CID=1 *Caller ID enabled.*
#CID=2 *Caller ID enabled with extended information, including caller's name.*
+++ Escapes to online-command mode.

DIP Switches (Modems with DIP Switches Only)

Note: If a DIP switch is on, it is down. If a DIP switch is off, it is up. *Defaults are in italics.* DIP switches which control a modem function override related modem commands written to the modem (i.e. You cannot write **AD0** to NVRAM if DIP 1 is up.).

Factory

Switch Setting Function**1 OFF** Data Terminal Ready (DTR) Override

OFF Normal DTR operations: computer must provide DTR signal for the Modem to accept commands; dropping DTR terminates a call

ON Modem ignores DTR (Override)

2 OFF Verbal/Numeric Result Codes

OFF Verbal (word) results

ON Numeric results

3 ON Result Code Display

OFF Suppresses result codes

ON Enables result codes

4 OFF Command Mode Local Echo Suppression

OFF Displays keyboard commands

ON Suppresses echo

5 ON Auto Answer Suppression

OFF Modem answers on first ring, or higher if specified in NVRAM

ON Disables auto answer

Factory**Switch Setting** Function**6 OFF** Carrier Detect (CD) Override

OFF Modem sends CD signal when it connects with another modem, drops CD on disconnect

ON CD always ON (Override)

7 OFF Power-on and ATZ Reset Software Defaults

OFF Loads Y or Y1 configuration from user-defined nonvolatile memory (NVRAM)

ON Loads &F0Generic template from read only memory (ROM)

8 ON AT Command Set Recognition

OFF Disables command recognition (Dumb Mode)

ON Enables recognition (Smart mode)

S-Registers

To change a setting, use the `ATSr=n` command, where *r* is the register and *n* is a decimal value from 0 -255 (unless otherwise indicated).

Register Default Function

S0 0 Sets the number of rings on which to answer in Auto Answer Mode. When set to 0, Auto Answer is disabled.

S1 0 Counts and stores the number of rings from an incoming call. (S0 must be greater than 0.)

S2 43 Stores the ASCII decimal code for the escape code character. Default character is +. A value of 128 - 255 disables the escape code.

S3 13 Stores the ASCII code for the Carriage Return character. Valid range is 0 - 127.

S4 10 Stores the ASCII decimal code for the Line Feed character. Valid range is 0 - 127.

S5 8 Stores the ASCII decimal code for the Backspace character. A value of 128255 disables the Backspace key's delete function.

Register Default Function

S6 2 Sets the number of seconds the modem waits before dialing. If *Xn* is set to *X2* or *X4*, this is the time-out length if there isn't a dial tone.

S7 60 Sets the number of seconds the modem waits for a carrier. May be set for much longer duration if, for example, the modem is originating an international connection.

S8 2 Sets the duration, in seconds, for the pause (,) option in the Dial command.

S9 6 Sets the required duration, in tenths of a second, of the remote modem's carrier signal before recognition by the Sportster modem.

S10 7 Sets the duration, in tenths of a second, that the modem waits to hang up after loss of carrier. This guard time allows the modem to distinguish between a line disturbance from a true disconnect (hang up) by the remote modem.

NOTE: While we don't recommend connecting the modem to a line with call waiting, if you have it, you may wish to adjust this setting upward to prevent the modem from misinterpreting the second call signal as a disconnect by the

remote modem.

Register Default Function

S10 (cont.) A better alternative is to ask your phone company how to temporarily disable call waiting (usually *70W). For example: ATDT *70W *phone number*.

Note: If you set S10=255, the modem will not hang up when carrier is lost. Dropping DTR hangs up the modem.

S11 70 Sets the duration and spacing, in milliseconds, for tone dialing.

S12 50 Sets the duration, in fiftieths of a second, of the guard time for the escape code sequence (+++).

S13 0 Bit-mapped register. Select the bit(s) you want on and set S13 to the total of the values in the Value column. For example, ATS13=17 enables bit 0 (value is 1) and bit 4 (value is 16).

Bit Value Result

0 1 Reset when DTR drops.

1 2 Reset non-MNP transmit buffer from 1.5K to 128 bytes.*

2 4 Set backspace key to delete.

3 8 On DTR signal, auto dial the number stored in NVRAM at position 0.

Register Default Function

S13 (cont.)

Bit Value Result

4 16 At power on/reset, Auto Dial the number stored in NVRAM at position 0.

5 32 Reserved

6 64 Disable quick retrains.

7 128 Disconnect on escape code.

* The 1.5K-byte non-ARQ buffer allows data transfer with Xmodem- and Ymodem-type file transfer protocols without using flow control. The 128-byte option lets remote users with slower modems keep data you're sending from scrolling off their screens. When remote users send your computer an XOFF (Ctrl-S) and you stop transmitting, the data in transit from your modem's buffer doesn't exceed the size of their screen. This is also very helpful in situations when a remote modem/printer application is losing characters.

S15 0 Bit-mapped register setup. To set the register, see instructions for S13.

Bit Value Result

0 1 Disable ARQ/MNP for V.22.

1 2 Disable ARQ/MNP for V.22bis.

2 4 Disable ARQ/MNP V.32/V.32bis/V.32terbo.

3 8 Disable MNP handshake.

4 16 Disable MNP level 4.

5 32 Disable MNP level 3.

Register Default Function

S15 (cont.)

Bit Value Result

6 64 MNP incompatibility.

7 128 Disable V.42 operation.

To disable V.42 detect phase, select the total of the values for bits 3 and 7.

S16 0 Bit-mapped register setup. To set the register, see instructions for S13.

Bit Value Result

0 1 Reserved

1 2 Reserved

2 4 Touch tone test.

4 8 Use internal test pattern.

S18 0 Test timer for &T loop back testing. Sets the time in seconds of testing before the modem automatically times out and terminates the test. When set to 0, the timer is disabled. Valid range is 1-255.

S19 0 Sets the duration, in minutes, for the inactivity timer. The timer activates when there is no data activity on the phone line; at time-out the modem hangs up. S19=0 disables the timer.

S21 10 Sets the length, in 10-millisecond units, of breaks sent from the modem to the computer; applies to MNP or V.42 mode only.

Register Default Function

S22 17 Stores the ASCII decimal code for the XON character.

S23 19 Stores the ASCII decimal code for the XOFF character.

S25 20 Sets the duration, in hundredths of a second, that DTR must be dropped so that the modem doesn't interpret a random glitch as a DTR loss. (Most users will want to use the default; this register is useful for setting compatibility with older systems running under older operating software.)

S27 0 Bit-mapped register setup. To set the register, see instructions for S13.

Bit Value Result

0 1 Enables ITU-T V.21 modulation at 300 bps for overseas calls; in V.21 mode, the modem answers both overseas and domestic (U.S. and Canada) calls, but only originates V.21 calls. (Default Bell 103)

1 2 Enables unencoded (non-trellis coded) modulation in V.32 mode.

2 4 Disables V.32 modulation.

3 8 Disables 2100 Hz answer tone to allow two V.42 modems to connect faster.

4 16 Enables V.23 fallback mode.

5 32 Disables V.32*bis* mode.

6 64 Disable V.42 selective reject.

Register Default Function**S27 (cont.)****Bit Value Result**

7 128 Software compatibility mode. This setting disables the codes and displays the 9600 code instead. The actual rate of the call can be viewed on the ATI6 screen. Used for unusual software incompatibilities. Some software may not accept 7200, 12,000, and 14,400 bps or greater result codes.

S28 0 Eliminates the V.32 answer tones for a faster connection.

8 Default item, all times are in tenths of seconds.

255 Disables all connections except V.32 at 9600 bps.

S29 20 Sets the duration, in tenths of a second, of the V.21 answer mode fallback timer.

S30 - Reserved

S31 - Reserved

S32 2 Bit-mapped register setup. To set the register, see the instructions for S13.

Bit Value Result

0 1 V.8 Call Indicate enabled.

1 2 Enables V.8 mode.

Register Default Function**S32 (cont.)****Bit Value Result**

2 4 Reserved.

3 8 Disable V.34 modulation.

4 16 Disable V.34+ modulation.

5-7 32-128 Reserved.

S33 0 Bit-mapped register setup. To set the register, see the instructions for S13.

Bit Value Result

0 1 Disable 2400 symbol rate.

1 2 Disable 2743 symbol rate.

2 4 Disable 2800 symbol rate.

3 8 Disable 3000 symbol rate.

4 16 Disable 3200 symbol rate.

5 32 Disable 3429 symbol rate.

6 64 Reserved

7 128 Disable shaping.

S34 0 Bit-mapped register setup. To set registers, see instructions for S13.

Bit Value Result

0 1 Disable 8S-2D trellis encoding.

1 2 Disable 16S-4D trellis encoding.

Register Default Function

S34 (cont.)**Bit Value Result**

- 2 4** Disable 32S-2D trellis encoding.
- 3 8** Disable 64S-4D trellis encoding.
- 4 16** Disable nonlinear coding.
- 5 32** Disable TX level deviation.
- 6 64** Disable Pre-emphasis.
- 7 128** Disable Pre-coding.
- S35 0** Bit-mapped register setup. To set registers, see instructions for S13.

Bit Value Result

- 0-2 1-7** Limit back channel rate.
 - 1=28800 max.
 - 2=26400 max.
 - 3=24000 max.
 - 4=21600 max.
 - 5=19200 max.
 - 6=16800 max.
 - 7=14400 max.
- 3 8** Force RBS operation
- 4-7 12-128** Reserved.

Register Default Function

S38 0 Sets an optional delay, in seconds, before a forced hang-up and clearing of the Transmit buffer when DTR drops during an ARQ call. This allows time for a remote modem to acknowledge receipt of all transmitted data before it is disconnected. The modem immediately hangs up when DTR drops. This option only applies to connections terminated by dropping DTR. If the modem receives the ATH command, it ignores S38 and immediately hangs up.

S39-S40 Reserved

S41 0 Bit-mapped register setup. To set registers see instructions for S13.

Bit Value Result

- 0 1** Distinctive ring enabled.
- 1-7 2-128** Reserved

The Serial Interface

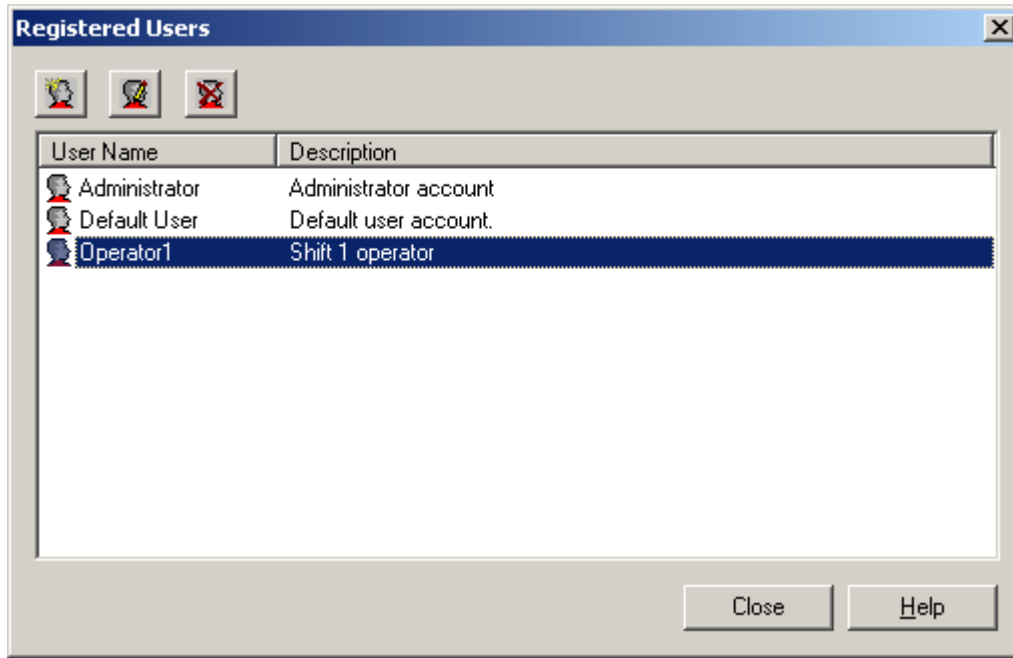
The serial interface is a standard developed by the Electronic Industries Association (EIA). It defines the signals and voltages used when data is exchanged between a computer and a modem or serial printer. The entire standard covers many more functions than are used in most data communications applications. Data is transmitted between the devices over a shielded serial cable with a 25-pin male (DB-25) connector to the modem and a 25-pin, 9-pin, 8-pin, or custom-built connector to the computer. FCC regulations require the use of a shielded cable when connecting a modem to a computer to ensure minimal interference with radio and television. Pin assignments are factory-set in the Sportster modem to match the standard DB-25 assignments in the following table. DB-9 connectors for IBM/AT-compatible computers should be wired at the computer end of the cable as shown in the DB-9 column.

Serial Interface Pin Definitions**Signal Source**

DB-25	DB-9	Circuit	Function	Computer/Modem
1	AA	Chassis Ground	Both	
2	3	BA Transmitted Data	Computer	
3	2	BB Received Data	Modem	
4	7	CA Request to Send	Computer	
5	8	CB Clear to Send	Modem	
6	6	CC Data Set Ready	Modem	
7	5	AB Signal Ground	Both	
8	1	CF Carrier Detect	Modem	
12	-	SCF Speed Indicate	Modem	
20	4	CD Data Terminal Ready	Computer	
22	9	CE Ring Indicate	Modem	

User Manager

The User Manager is used to control what actions operators can access in the server. With the addition of online full time operation, protecting the application is crucial. The User Manager features of the server are provided for users' convenience. The default user account gives users full access to the features of the server. If access to the project does not need to be limited, the User Manager system does not need to be edited.



By default, the Administrator account and the Default User account are always available. Only the Administrator account can be used to add additional users to the system or to change the settings of existing accounts. By default, the password for the Administrator account is "Administrator." It should be changed immediately if the user management system is to be used. The Administrator account cannot be deleted, but its name and password can be changed.

The Default user account is used when no other account is active. This is the normal condition of the server. Like the Administrator account, the Default User account cannot be deleted. However, the name and password of the Default User account are fixed. The only way to disable the Default account is to deny all privileges to the account. This can only be done when logged in as the Administrator.

When logged in as the Administrator, additional user accounts can be added. By clicking on the new user icon the [User Properties](#) dialog will be displayed. Existing user accounts can be edited by selecting the account and double-clicking or by pressing the edit user icon. To delete a user account, simply select the account and press the delete user icon.

When the User Management system is used, the server will log the current account name to the event log for all server actions taken by the user. With this in mind the "Reset Event Log" should be disabled on all accounts to prevent the log from being lost.

User Properties

The User Management system of the server is used to control the actions a user can take within a server project. The User Properties dialog is used to configure the name, password, and privileges available for the account.

The screenshot shows a 'User Properties' dialog box with the following content:

- Identification:**
 - Name: Operator1
 - Description: Shift 1 operator
- Password:**
 - Password: [Empty]
 - Confirm: [Empty]
- Privileges:**
 - Make changes to project files.
 - Make changes to application settings.
 - Perform functions that cause active clients to be disconnected.
 - Reset the event log.

Buttons: OK, Cancel, Help

The **Name** parameter is used to specify a name for the user. The name can be up to 31 characters in length.

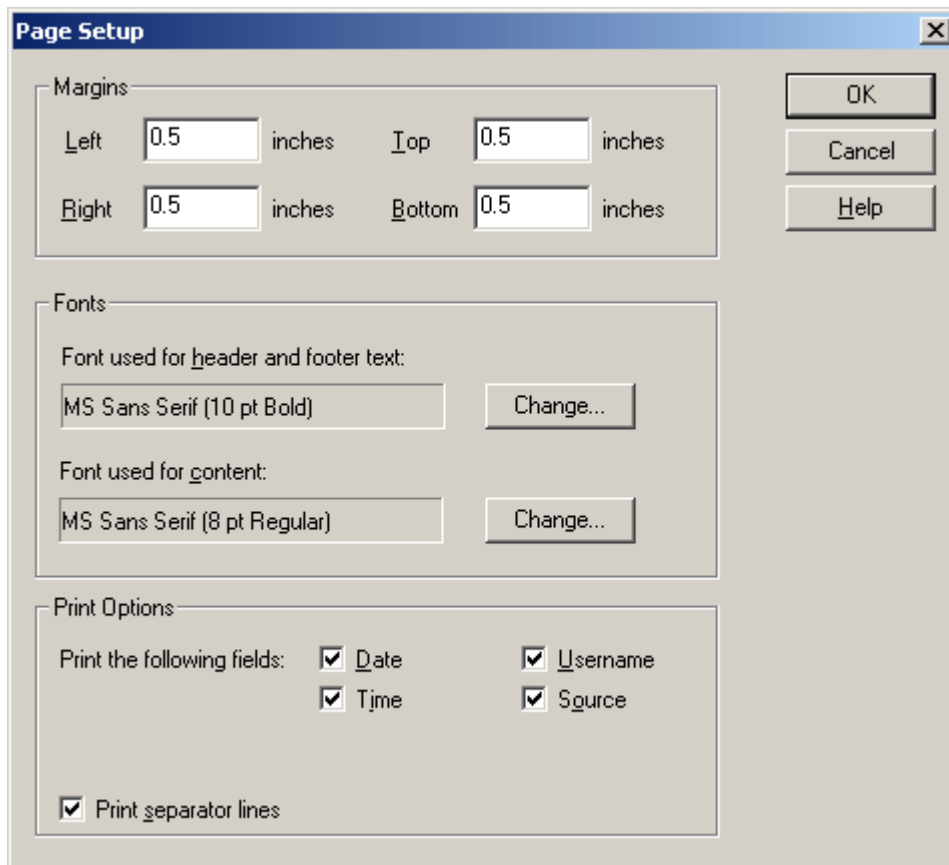
The **Description** parameter is used to give each user account a brief description. This can be used to help ensure that operators are using the proper account when they log in.

The **Password** parameter is used to specify the password the user must enter to log in to the system. The password can be up to 15 characters in length. You must enter it correctly in both the Password and Confirm fields of this dialog for the change to be accepted. Each time a user account is edited, the password must be re-entered. If the Password field is left blank the password will be removed from the account.

The **Privileges** selections are used to control what actions a given user account can access. When enabled, the **Make changes to project** selection will allow the user to modify the server project freely. If disabled, the user will not be able to make any changes to the project. The **Make changes to application** selection will allow an operator to make changes to the server **Options**. When enabled, the **Disconnect client** selection will allow the user to perform actions that may cause clients to be disconnected from the server. When **Disconnect client** is disabled, the user will not be able to cause disruptions to currently active clients.

Event Log Print - Page Setup

The event logging system of the server now supports printing of the log contents. To better suit the printer's nature, the event logging system allows the format of the page to be configured. The results of changes to page setup can be viewed by using the **Tools|Event Log|Page Setup** selection. The changes to page setup are done using the dialog shown below.



The **Margin** settings are used to tailor the distance top and bottom, left and right that the resulting event log printout will maintain from the edge of the printed page. All margin settings are entered in inches.

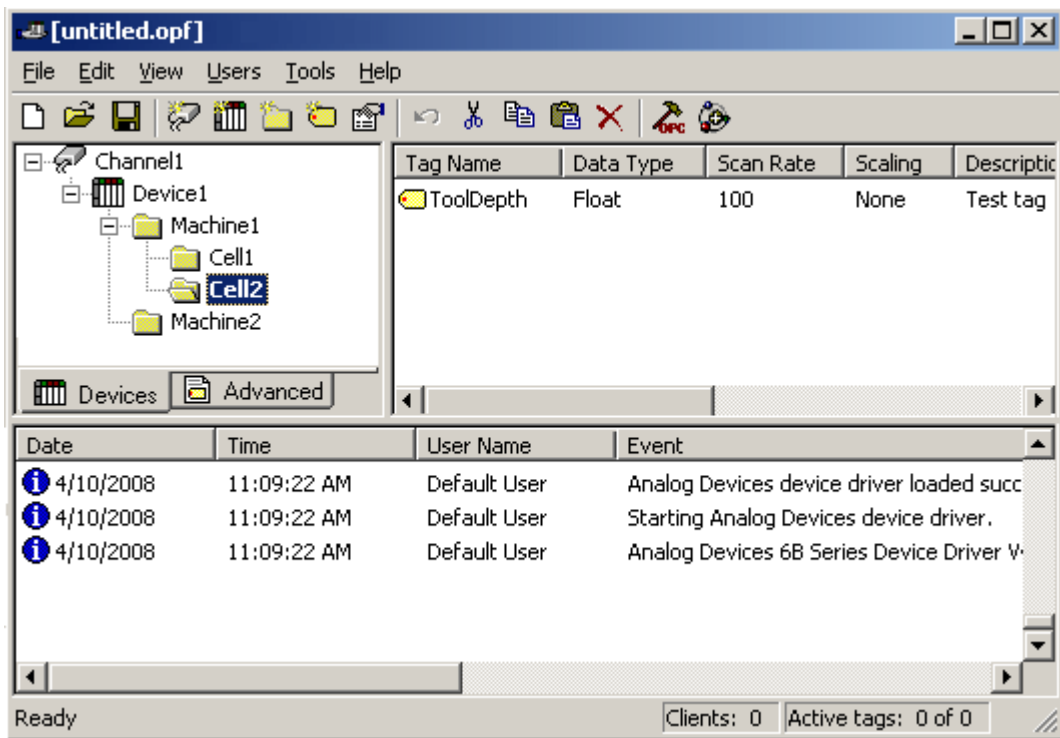
The **Fonts** settings are used to select a font to be used as the header and footer text and for the actual log event records. Only fixed space fonts will appear in the Fonts dialog. The default fonts for these selections are shown in the dialog above. To change a font, simply click on the change button. When invoked, the standard font selection dialog will be displayed. If you make changes to the font selections for either field type the **Tools|Event Log|Print Preview** should be used to confirm the desired results.

Once the format of the event log print has been established, the content of the event log output must be selected. The event logging system allows you to select date, time, username, and source of the event to be printed as part of the output. If the User manager is not being used, the username field will not appear in event log and can be disabled here.

The **Print separator lines** option will force a single line to be drawn between each group of five event records on the resulting event log printout.

Basic Server Components

The server core provides a uniform interface to all of its available communications drivers. Click on any area of the figure below to learn more about specific elements of the server.



Channel Functions

A **channel** represents a communication medium from the PC to one or more external devices. A channel can be used to represent either a serial port or a card installed in the PC.

Before adding devices to a project, users must define the channel to use when communicating with the devices. A channel and a device driver are closely tied. After creating a channel, only devices that the selected driver supports can be added to this channel.

Add Channel

Adding channels is done using the **Channel Wizard**. The wizard will guide users through the channel definition process. The steps are as follows:

1. The wizard first prompts for a logical name to use for the channel. This name must be unique among all channels and devices defined in the project.
2. The wizard then prompts for the device driver to use. A list box will be presented displaying all of the device drivers that are installed in the system. All serial drivers can be used with multiple channels in the same project. For hardware card drivers, refer to the driver help for ability to use with multiple channels in a single project.
3. If the device driver supports multiple channels, the wizard will then prompt for the **communication parameters** to use. Multiple channels cannot share identical communication parameters, i.e. two serial drivers cannot use COM1. Refer to the manufacturer's documentation and the driver's online help for the correct communication parameters to use for a particular device.
4. The **flow control** settings for serial drivers are primarily used when connecting RS422/485 network devices to the RS232 serial port via a converter. Most RS232 to RS422/485 converters require either no flow control (**None**) or that the RTS line be on when the PC is transmitting and off when listening (**RTS**.)
5. The wizard finishes with a summary of the new channel.

Remove Channel

To remove a channel from the project, either select the desired channel and press the DEL key or click **Edit|Delete** from the edit menu or tool bar.

Channel Properties

To display the Channel Properties, select the channel and then click **Edit|Properties** from the edit menu or tool bar.

New Channel - Name Page

The server supports the use of multiple communications drivers simultaneously. Each protocol or driver used in a project is referred to as a Channel. A channel refers to a specific communications driver. A server project can consist of many channels each with unique communications drivers or each with the same communications driver. A channel acts as the basic building block of an OPC link. Properties like communications port, baud rate, and parity are contained at the channel level.

Each channel name must be unique in a server project. The channel name can be up to 256 characters long. While using long descriptive names is generally a good idea, keep in mind that some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The channel name entered here will be part of the OPC browser information.

Channel Properties - Identification

Each protocol or driver used in a server project is referred to as a Channel. A channel refers to a specific communications driver. A server project can consist of many channels each with unique communications drivers or each with the same communications driver.

Each channel name must be unique in a server application. The channel name can be up to 256 characters long. While using long descriptive names is generally a good idea, keep in mind that some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The channel name entered here will be part of the OPC browser information.

Selecting the **Enable diagnostics** check box will enable diagnostic information to be available to your OPC application for this channel. With diagnostic functions enabled, [diagnostic tags](#) are available for use within client applications. In addition to diagnostic tags, a [diagnostic window](#) is also available when this feature is enabled. The diagnostic features of the server do require a minimal amount of overhead processing. For this reason it is recommended that you only use the diagnostic features when needed and disable them when not in use which is the default case.

Note: With the server's online full time operation, these parameters can be changed at any time. This includes changing the channel name that can prevent clients from registering data with the server. If a client has already acquired an item from the server before you change the channel name those items will be unaffected. However, after the channel name has been changed, if the client application releases the item and attempts to re-acquire using the old channel name the item will not be accepted. With this in mind you don't want to make changes to parameters like channel name once you have a large client application developed. To prevent operators from changing these parameters, use the [User Manager](#) to restrict access rights to server features.

New Channel - Driver Page

Once the new channel has been named, a communications driver must be selected from the "**Device driver:**" drop down list. Depending on the communications drivers installed, the drop down list will contain the available drivers for use in your server project.

Since the server supports the use of multiple communications drivers simultaneously, a number of channels can be added to the project. It is not necessary to select a different communications driver for each channel. Many of the communications drivers available for the server support operation on either multiple communications ports or across multiple network connections. If the chosen driver does not support multiple channels or the number of channels supported has been exceeded, a dialog will be displayed to that fact.

A new and useful feature of the server is the addition of diagnostic functions. By selecting the **Enable diagnostics** check box, diagnostic information will be available to the OPC application for this channel. With diagnostic functions enabled, [diagnostic tags](#) are available for use within client applications. In addition to diagnostic tags, a [diagnostic window](#) is also available when this feature is enabled.

Channel Properties - Comm. Parameters

The Comm. Parameters page will vary depending on the device driver being used. If a serial device driver is being used, users will be asked to provide the information described below.

ID: The COM ID that will be used when communicating with devices assigned to the channel.

Baud Rate: The baud rate that should be used to configure the selected COM port.

Data Bits: The number of data bits per data word (5, 6, 7, or 8).

Parity: The type of parity the data should use (Odd, Even, or None)

Stop Bits: The number of stop bits per data word (1 or 2)

Flow Control: Determines how the [RTS and DTR](#) control lines should be utilized.

Use Modem: Check this box if planning to communicate with devices connected to the port using a modem. This selection is only available if a modem has been defined for the PC. For more information, refer to [Modem Support](#).

Report comm errors: The Report comm errors selection is used to turn the reporting of low level communications errors On or Off. When enabled, low-level errors like parity, framing, and overrun errors can be posted to the event log if they occur. When disabled these same errors will not be posted, however normal request failures will still be posted.

If you select Use Modem, the next page is used to select a modem. Each channel requires a unique modem. Afterwards, users will be able to set the dialing properties and modem properties for the channel. Dialing properties are used to specify where users are calling from and how the server should dial an outside line or long distance number. For these settings to take effect, the phone number that is dialed must be in [canonical format](#). The properties dialog is used to set up the modem. Consult the modem documentation for more information on these settings.

Use Ethernet Encapsulation: Many serial drivers also support Ethernet Encapsulation mode, which is used to use an Ethernet based serial port gateway instead of your normal PC based serial port. **See Also:** [Channel Properties - Ethernet Encapsulation](#).

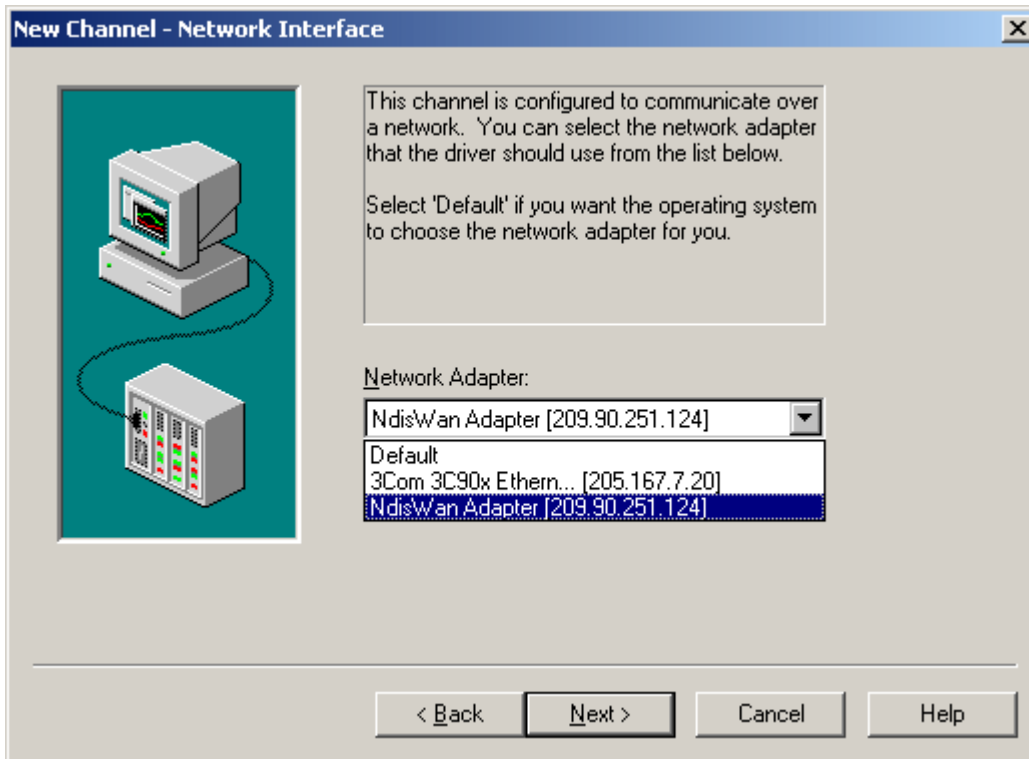
Note: With the server's online full time operation, these parameters can be changed at any time. This includes switching communications port, baud rate, and etc. To prevent operators from changing these parameters, use the [User Manager](#) to restrict access rights to server features. Keep in mind that changes made to these parameters can temporarily disrupt communications.

Channel Properties - Network Interface

With the addition of Ethernet Encapsulation, virtually all of the drivers currently available support some form of Ethernet communications. Whether you are using a natively Ethernet based driver or a serial driver configured for [Ethernet Encapsulation](#) you will be using some form of a network interface. In most cases that interface takes the form of a NIC (Network Interface Card). For a PC that has networking installed, usually this means that a single NIC is installed providing a connection to either your IT or plant floor network or both. In most cases this configuration works very well for your typical network configurations and loading. While this typical configuration might be fine for normal performance levels what happens if you need to get data from an Ethernet device at a very regular interval? If the plant floor network is mixed with your IT network, a large batch file transfer could completely disrupt the interval of your plant floor data.

The most common way to deal with this issue is to install a second NIC in your PC and use one NIC for accessing your IT network and another NIC to access your plant floor data. While this sounds reasonable, simple problems arise when you try to create a clear separation between these networks. When using multiple NICs in your PC you must determine the bind order. The bind order determines what NIC is used to access different portions of your Ethernet network. In many cases these bind settings can be easily managed using operating systems tools. When there is a clear separation between the types of protocols and services that will be used on each NIC card the bind order can be done by your operating system. But what if there isn't a clear way to select a specific bind order? When this is the case you may find that your Ethernet device connection is being routed to the wrong network and it may take hours to determine how to set the bind order properly. There is a better way.

The Network Interface selection shown here allows you to specifically select a NIC card for use with your Ethernet driver. As shown in the dialog below the Network Interface selection allows you to select a specific NIC card based on either the NIC name or its currently assigned IP address. This list of available NICs will include either unique NIC cards or NICs that have multiple IP assigned to them. Additionally the selection will also display any WAN connections you may have active such as a dialup connection.



By selecting a specific NIC interface you will be able to force the driver to send all Ethernet communication through the specified NIC. When a NIC is selected the normal operating system bind order is completely by-passed. This ensures that you have control over how your network will operate, which will eliminate any guesswork.

The selections shown in the **Network Adapter** drop down menu will be completely dependent upon your network configuration settings and the number of unique NICs installed in your PC or based on the number of unique IPs assigned to your NICs. If you would like the operating system to make the bind order selection for you, you can select "Default" as your network adapter. This will allow the driver to use the operating system's normal bind order to set the NIC that will be used.

Note 1: Select the **Default** condition is unsure of which NIC should be used. Select the default condition if an Ethernet based device is being used and is exposed to this feature through a product upgrade.

Note 2: With the server's online full time operation, this parameter can be changed at any time. To prevent operators from changing this parameter use the [User Manager](#) to restrict access rights to server features. Keep in mind that changes made to this parameter can temporarily disrupt communications.

Flow Control RTS & DTS

Flow control may be required to communicate with the specific serial device. The available flow control options are as follows:

1. **None:** No control lines are toggled or asserted.
2. **DTR:** The DTR line is asserted when the communications port is opened and remains on.
3. **RTS:** Specifies that the RTS line will be high if bytes are available for transmission. After all buffered bytes have been sent; the RTS line will be low. This is normally used with RS232/RS485 converter hardware.
4. **RTS, DTR:** Combination of DTR and RTS as described above.
5. **RTS Always:** The RTS line is asserted when the communication port is opened and remains on.
6. **RTS Manual:** The RTS line is asserted based upon the timing parameters entered for [manual RTS control](#).

Note: The Windows 95/98 serial device driver (serial.vxd) does not properly support toggling of the RTS line when using the RTS or RTS, DTR flow control settings. Although the serial device drivers are capable of performing this function under Windows 95/98, it is recommended that users purchase a hardware converter that does not require the RTS line to be toggled.

Channel Properties - Manual RTS Flow Control

The Flow Control selection of **RTS Manual** allows the server to control the operation of the RTS line for use with external devices such as radio modems that require additional timing to properly initiate communications. For those drivers that support it, the **RTS Manual** selection enables the setting of three timing parameters. These settings will only appear if the driver in use supports **RTS Manual**.

Raise the RTS line: This setting controls how long the RTS line will be high **before** any data is transmitted from the communications port. The time entered is in milliseconds and has a valid range of 0 to 2550 milliseconds. The default is 10 milliseconds.

Drop the RTS line: This setting controls how long the RTS line will be held high **after** any data is transmitted from the communications port. The time entered is in milliseconds and has a valid range of 0 to 2550 milliseconds. The default is 10 milliseconds.

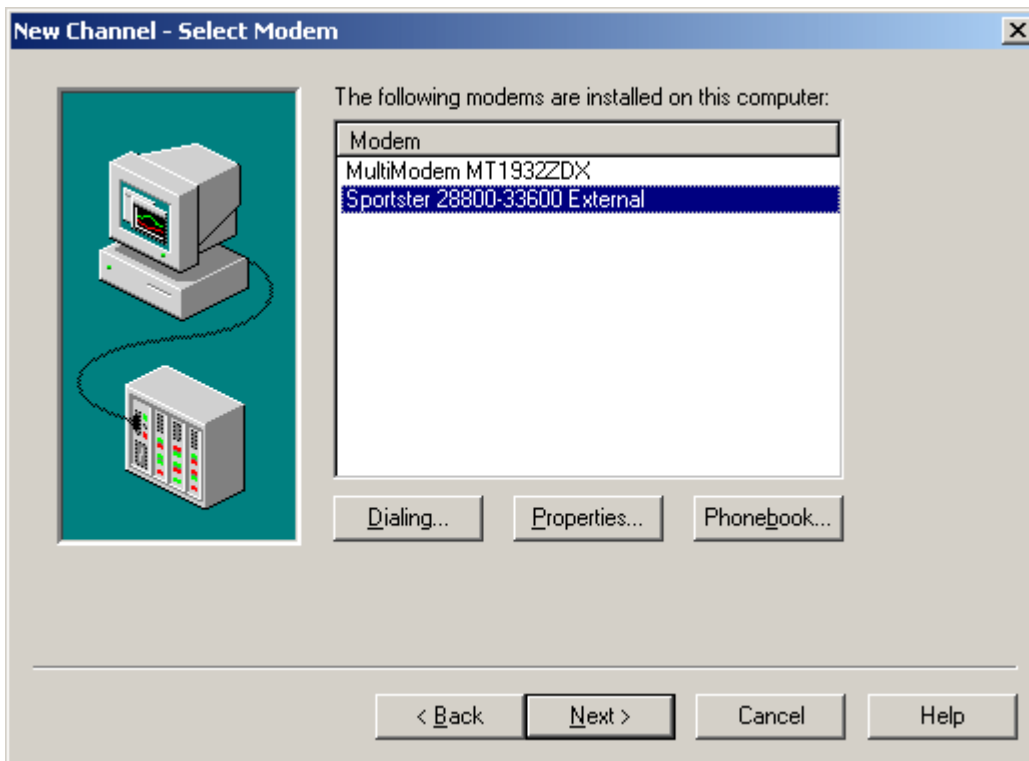
Poll Delay: This setting allows a delay to be introduced between each communication request. Some radio modems need a defined amount of settling time before the next transmission can occur. The Poll Delay allows you to configure that delay. The time entered is in milliseconds and has a valid range of 0 to 2550 milliseconds. The default is 10 milliseconds.

Note: With the server's online, full-time operation, any of these parameters can be changed at any time. To prevent operators from changing these parameters, use the [User Manager](#) to restrict access rights to specific server features.

Channel Properties - Modem

If modem support has been selected for the application, users must select the modem they intend to use for the channel. This dialog is used to select a modem from the list of modems configured for the PC, set the properties of that modem, and enters phonebook tags. Once the modem has been chosen, its settings must be configured to match the settings of the target device. The help file section, [Using Modems](#), provides additional details that must follow when configuring the modem application.

Once the modem has been configured properly, modem tags can be used in the application to set the desired phone number to dial. A list of phone numbers can also be set that can be seen as OPC tags from the OPC client application. The [Phonebook tags](#), once configured, can be used to automatically dial a desired number simply by writing anything to the tag. To enter Phonebook tags, click on **Phonebook**.

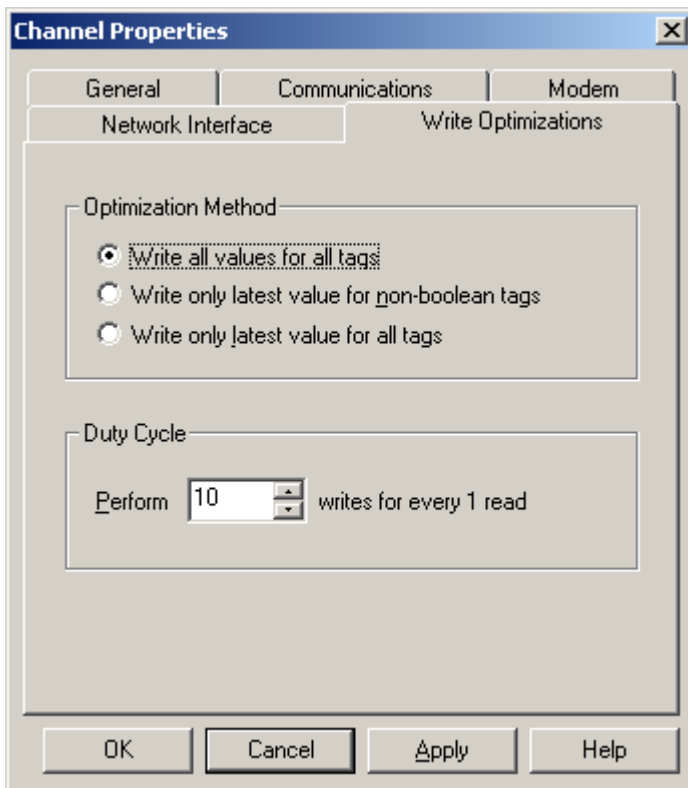


Note: With the server's online full time operation, these parameters can be changed at any time. The [User Manager](#) can be used to prevent operators from changing the parameters.

Channel Properties - Write Optimizations

As with any OPC server, writing data to the device may be the most important aspect of the application. Ensuring that the data written from the OPC client application gets to the device in a timely fashion is the goal of the server. Given this goal, the server provides a number of optimization settings that can be used to improve the application's responsiveness.

The Write Optimization page shown below is used to control how write data is passed to the underlying communications driver as well as adjust the ratio at which those writes will be processed and sent to the device.



There are currently three write optimization modes. **Write all values for all tags**, the default mode, forces the server to attempt to write every value to the controller. In this mode the server will continue to gather OPC write requests and add them to the server's internal write queue. The server will then process this write queue and attempt to empty the queue by writing data to the device as quickly as possible. This mode ensures that everything written from your OPC client applications will be sent to the target device. This mode should be selected if the order of the write operations or the content of every write item must uniquely be seen at the target device.

While writing every value to the device may seem like the best course of action there are a number of applications where writing every value, many of which may be the same value, over and over may be simply a waste of communications bandwidth. A good example of this would be a slider switch in an HMI application. As the slide switch is moved, the HMI sends writes to the server. If the user moves the slide switch fast enough the server may start to accumulate writes in the queue. Once the user stops moving the slide switch the write queue empties and the value in the device finally reaches the same value of the slide switch.

Many consecutive writes to the same value can accumulate in the write queue. The accumulation of writes in the queue occurs due to the time required to actually send the data to the device. If the server were to simply update a write value that has already been placed in the write queue, far fewer writes would need to be done to reach the same final output value. In this way no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. This is the mode of operation that the second write optimization mode **Write only latest value for non-Boolean tags** allows. As the mode states, any value that is not a Boolean value will be updated in the server's internal write queue and will then be sent to the device at the next possible opportunity. This can dramatically improve the overall performance of the application.

Like all good things this feature must be used with a clear understanding of how it will affect the application's operation. As stated in the selection text "Write only latest value for non-Boolean tags", does not attempt to optimize writes to Boolean values. This is used to optimize the operation of HMI data such as the slide switch example without causing problems with Boolean operations like a momentary push button.

The final write optimization mode, **Write only the latest value for all tags**, takes the operation described for the second mode and applies it to all tags. If the application only needs to send the latest value to the device, this mode will optimize all writes by updating the tags currently in the write queue before they are sent.

The **Duty Cycle** selection is used to control the ratio of write operations to read operations. The ratio is always based on one read for every one to ten writes. By default the duty cycle is set to ten. This means that ten writes will occur for

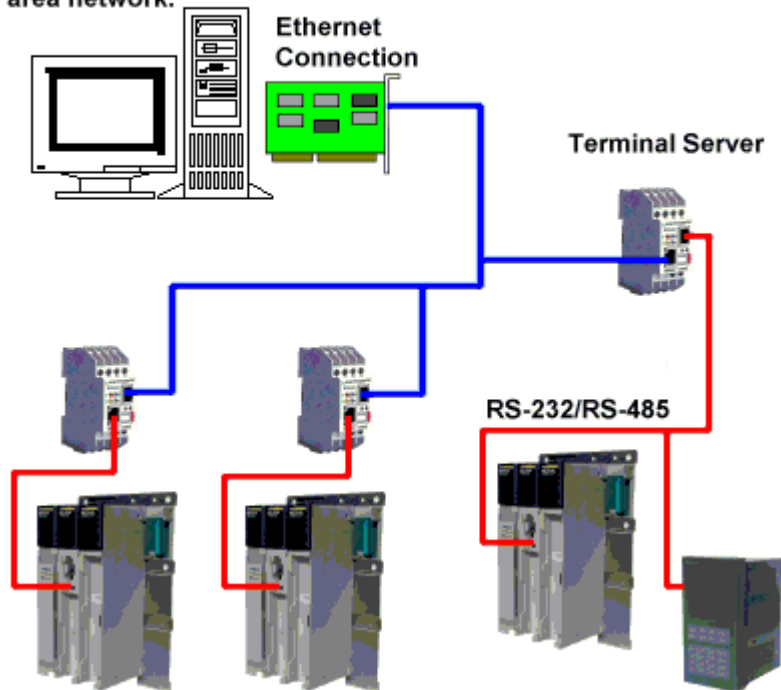
each read operation. Reduce the duty cycle to ensure that read data is given time to process if the application is doing a large number of continuous writes. A setting of one will result in one read operation for every write operation. In all cases if there are no write operations to perform, reads will be processed continuously.

Note: We strongly suggest that the application be characterized for compatibility with these write- optimization enhancements before they are used in a production environment.

Channel Properties - Ethernet Encapsulation

The Ethernet Encapsulation mode has been designed to provide communications with serial devices connected to terminal servers on your Ethernet network. A terminal server is essentially a virtual serial port. The terminal server converts TCP/IP messages on your Ethernet network to serial data. Once the message has been converted to a serial form you can connect standard devices that support serial communications to the terminal server. The following diagram provides a demonstration of how the Ethernet Encapsulation mode should be employed.

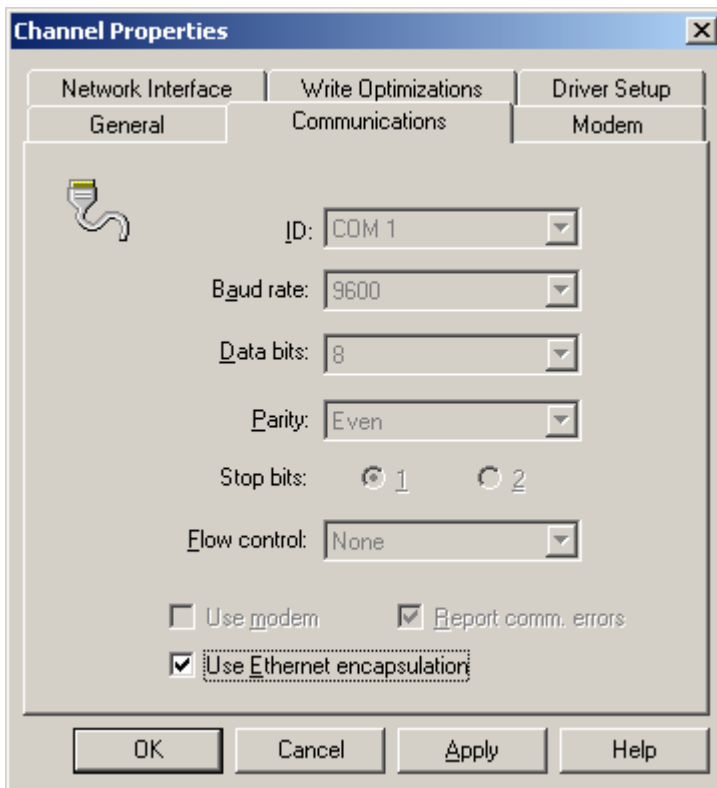
Ethernet Encapsulation can be used to access multiple Serial devices spread across a local area network.



Additional uses include use over wireless network connections such as 802.11b and CDPD packet networks. The Ethernet Encapsulation mode has been developed to support a wide range of serial devices. Using a terminal server device is used to place RS-232 and RS-485 devices throughout plant operations while still allowing a single localized PC to access the remotely mounted devices. The Ethernet Encapsulation mode allows an individual Network IP address to be assigned to each device as needed. By using multiple terminal servers, users can access hundreds of serial devices from a single PC.

Configuring Ethernet Encapsulation Mode

To begin using Ethernet Encapsulation mode, select the **Use Ethernet Encapsulation** checkbox either in the [Communications Parameters](#) dialog of the Channel Wizard or by clicking **Channel Properties | Communications** tab as shown below.

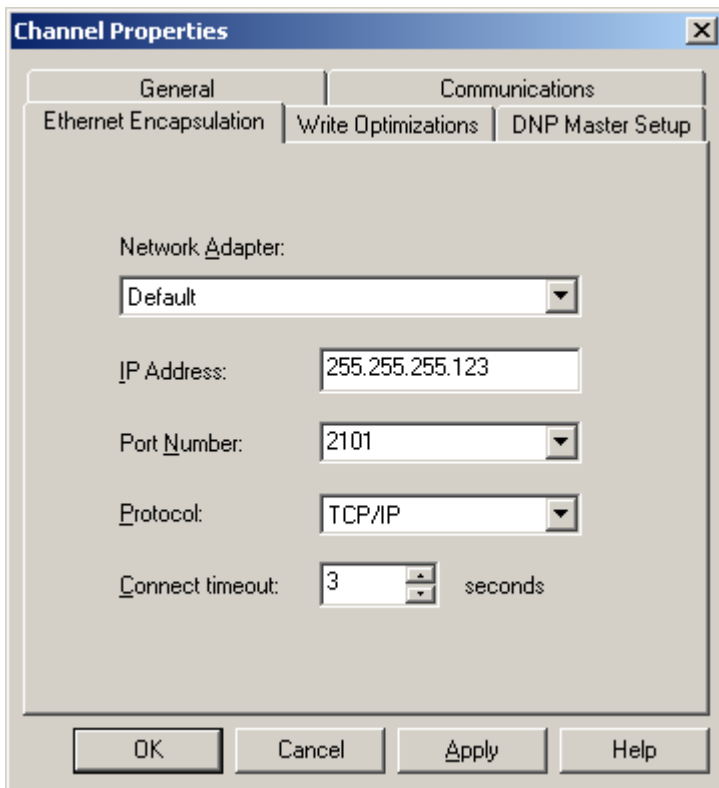


Important: When Ethernet Encapsulation mode is selected, the serial port settings such as baud rate, data bits, and parity become grayed out. This occurs because these settings will not be used in Ethernet Encapsulation mode. The terminal server being used must have its serial port properly configured to match the requirements of the serial device that will be attached to the terminal server.

The multiple channel support of the server is used to have up to 16 channels on each driver protocol. When using multiple channels you can have a channel defined to use the local PC serial port and have a channel defined to use Ethernet Encapsulation mode.

Channel-level Ethernet Encapsulation Settings

Once the Ethernet Encapsulation mode has been enabled, the settings must be configured. The settings may be reached either through the Channel Wizard dialog or after the channel has been added by right-clicking on the channel and then selecting **Properties | Ethernet Encapsulation**.



Use the **Network Adapter** drop-down list to select the network adapter being used.

The **IP Address** selection is used to enter the four-field IP address of the terminal server to which this device is attached. IPs are specified as YYY.YYY.YYY.YYY The YYY designates the IP address (each YYY byte should be in the range of 0 to 255). Each channel will have its own IP address.

The **Port Number** selection is used to configure the Ethernet port to be used when connecting to a remote terminal server. The valid range is from 1 to 65535 with some numbers reserved. The default is 2101.

The **Protocol** selection is used to use either TCP/IP or UDP communications. The selection depends completely on the nature of the terminal server you are using. Consult your terminal server's documentation for more information on the protocol available. The default protocol selection is TCP/IP.

Note: It is important to remember that the Ethernet Encapsulation mode is completely transparent to the actual serial communications driver. With this in mind you must still configure the remaining device settings just as you would if you were connecting to the device directly on your local PC serial port.

The **Connect Timeout** setting defines the amount of time (in seconds) required to establish a socket connection to a remote device to be adjusted. In many cases the connection time to a device can take longer than a normal communications request to that same device. The valid range is 1 to 999 seconds. The default is 3 seconds.

Note: With the server's online full time operation, these parameters can be changed at any time. To prevent operators from making changes, use the [User Manager](#) to restrict access rights to server features.

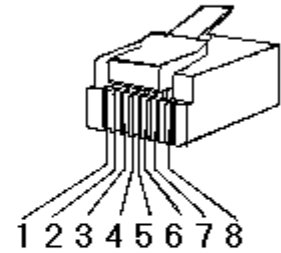
Cable Diagrams

Patch Cable (Straight Through)

TD + 1	OR/WHT	OR/WHT	1	TD +
TD - 2	OR	OR	2	TD -
RD + 3	GRN/WHT	GRN/WHT	3	RD +
4	BLU	BLU	4	
5	BLU/WHT	BLU/WHT	5	
RD - 6	GRN	GRN	6	RD -
7	BRN/WHT	BRN/WHT	7	
8	BRN	BRN	8	

RJ45 RJ45

10 BaseT



Crossover Cable

TD + 1	OR/WHT	GRN/WHT	1	TD +
TD - 2	OR	GRN	2	TD -
RD + 3	GRN/WHT	OR/WHT	3	RD +
4	BLU	BLU	4	
5	BLU/WHT	BLU/WHT	5	
RD - 6	GRN	OR	6	RD -
7	BRN/WHT	BRN/WHT	7	
8	BRN	BRN	8	

RJ45 RJ45

8-pin RJ45

New Channel - Summary

The Channel Summary page allows users to review the selections made for the new channel. If any of the selections displayed in the summary report need to be changed, simply click on the Back button until the required dialog is displayed.

Device Functions

[Devices](#) represent PLCs or other hardware with which the server will communicate. The device driver that the channel is using restricts device selection.

Add Device

Devices can be added using the [New Device Wizard](#). Change the method using the **Edit | New Device** menu option. The device name is user defined and should be a logical name for the device. This will be the browser branch name used in OPC links to access tags assigned to this device.

The [Network ID](#) is a number or a string that uniquely identifies the device on the device's network. Networked devices, multi-dropped, must have a unique identifier so the server's requests for data can be routed correctly. Devices that are not multi-droppable do not need an ID, and this setting is not available.

Remove Device

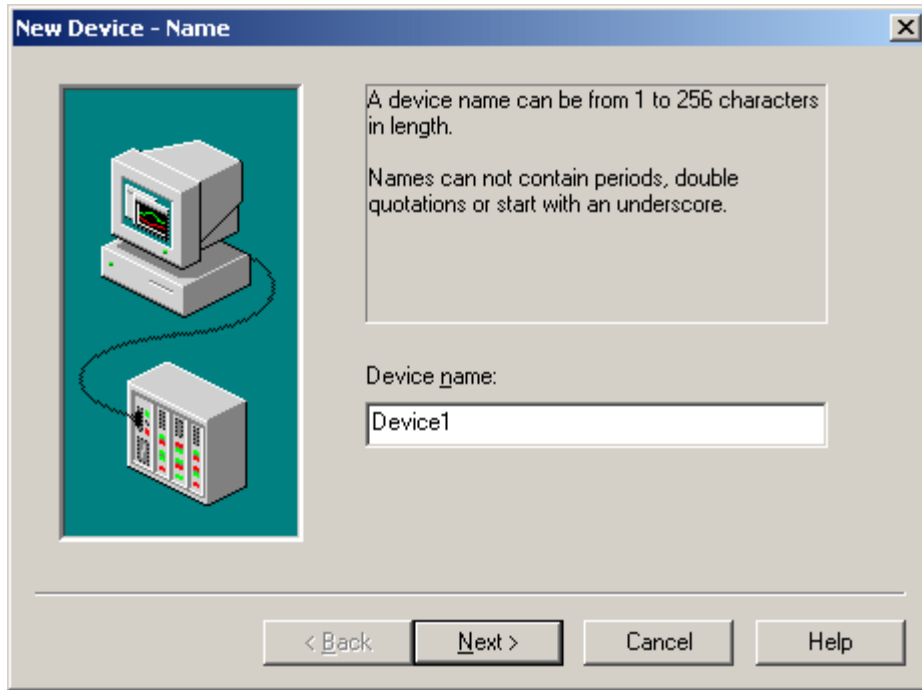
To remove a device from the project, select the desired device then press the DEL key. Alternatively, select **Edit | Delete** from the edit menu or tool bar.

Device Properties

To display the properties of a device, select the device and select **Edit | Properties** from the edit menu or tool bar.

New Device - Name

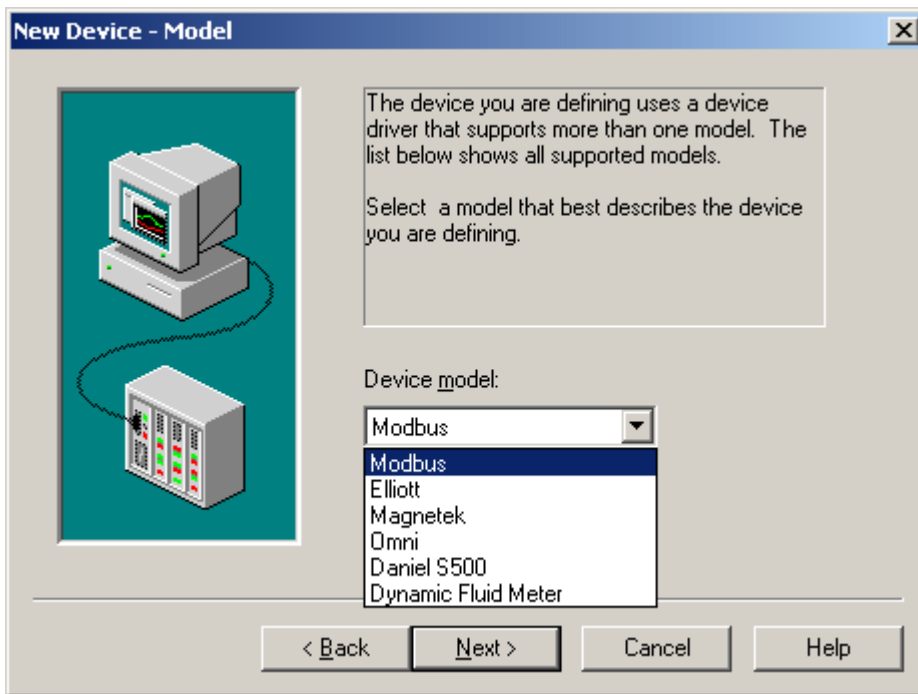
Unlike the channel name, a **Device name** can be the same from one channel to the next, although each device under a channel must have a unique name. The device name is a user defined logical name for the device. The device name can be up to 256 characters long. While using long descriptive names is generally a good idea, keep in mind that some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The device name and channel name will be part of the browse tree information.



Within an OPC client the combination of channel name and device name would appear "**ChannelName.DeviceName**".

New Device - Model

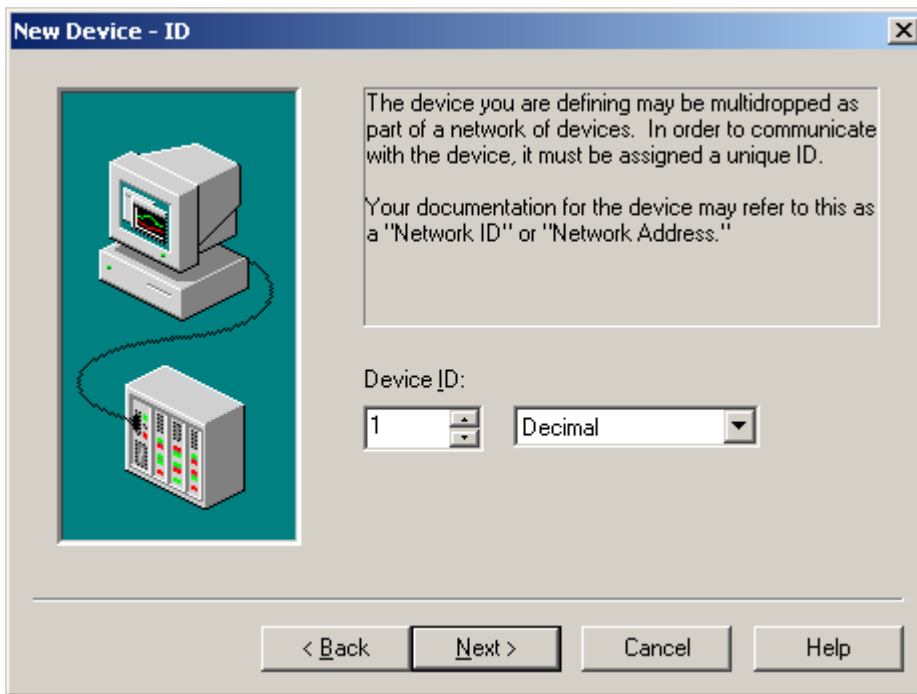
The **Model** parameter is used to select the specific type of the device associated with a Device ID. The contents of the model selection drop down will vary depending on the chosen communication driver. In some cases, a driver may not support any model selection in which case this option will be unavailable. If the communication driver being used supports multiple models, try to match the model selection to the physical device. If the device being used is not represented in the model drop down, select a model that conforms closest to the target device. This can be determined from the driver help specific to each available model. In some drivers a model selection of Open is available. For these drivers, the Open selection is used to communicate without knowing the specific details of your target device.



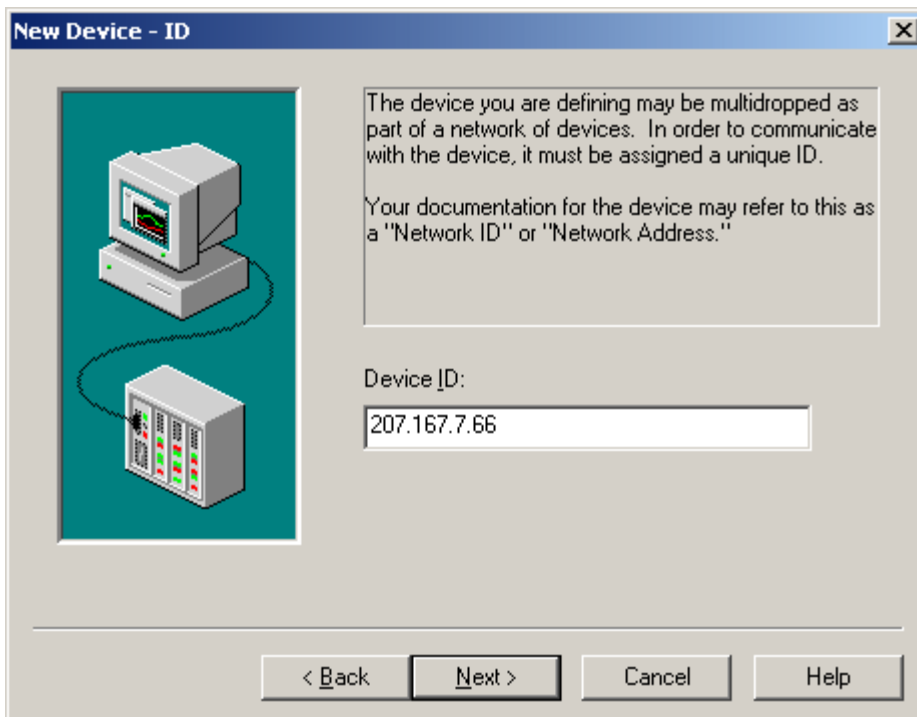
Note: With the server's online full time operation you can change many of parameters at any time. If the chosen communications driver supports multiple device models, the model selection can only be changed if there are currently no client applications connected to the device. Since this device is being added at this time, any model can be selected. To prevent operators from changing these parameters, use the [User Manager](#) to restrict access rights to server features.

New Device - ID

The **Device ID** parameter is used to specify the driver specific station or node for a given device. Depending on the communication driver being used, the type of ID entered will vary. For many communication drivers the ID is a numeric value. As shown in the dialog below, when a driver supports a numeric ID the menu option is used to enter a numeric value. Additionally the format of the entered numeric value can be changed to suit the needs of either the application or the characteristics of the chosen communication driver. The format is set by the driver by default. Possible formats are Decimal, Octal, and Hexadecimal.



If the driver in use is either an Ethernet based driver or supports a unconventional station or node name, the dialog below may be shown. In this case the Device ID is a TCP/IP ID. TCP/IP or UDP IDs consist of four values separated by periods. Each value has a range of 0 to 255. Some Device IDs are string based. Additionally there may be more parameters within the ID field depending on the communications driver you are using. Details on the specific nature of the Device ID can be found in help for the driver.



Note: With the server's online full time operation, these parameters can be changed at any time. The Device ID parameter can be changed at any time and will take effect immediately. To prevent operators from making any changes, use the [User Manager](#) to restrict access rights to server features.

Device Properties - General

The first step required to configure a project is to add a channel. Next, add a device to that channel. A device represents a single target on a communications channel. If the chosen driver supports multiple controllers, a Device ID must be created for each controller. The following two figures demonstrate the appearance of general device properties pages:

The screenshot shows the 'Device Properties' dialog box with the 'General' tab selected. The 'Channel Assignment' section shows 'Name: Channel4' and 'Driver: DIRECT-NET'. The 'Device' section shows 'Name: Device1', 'Model: DL-250(-1)', and 'ID: 1' with a 'Decimal' format. Both 'Enable data collection' and 'Simulate Device' checkboxes are checked. Buttons for 'OK', 'Cancel', 'Apply', and 'Help' are at the bottom.

The screenshot shows the 'Device Properties' dialog box with the 'General' tab selected. The 'Channel Assignment' section shows 'Name: Channel2' and 'Driver: Modbus Ethernet'. The 'Device' section shows 'Name: Device1', 'Model: Modbus', and 'ID: <207.167.7.66>.0'. 'Enable data collection' is checked, but 'Simulate Device' is unchecked. Buttons for 'OK', 'Cancel', 'Apply', and 'Help' are at the bottom.

The same device names can be used on multiple channels. The device name is a user defined logical name for the device, and it can be up to 256 characters long. While using long descriptive names is generally a good idea, keep in mind that some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The device name and channel name will be part of the browse tree information as well.

Within an OPC client the combination of channel name and device name would appear "**ChannelName.DeviceName**".

The **Model** parameter can be used to select the specific type of the device associated with this ID. The contents of the model selection drop down will vary depending on the chosen communication driver. In some cases, a driver may not support a model selection, in which case the model will be grayed out. If the communication driver you are using does support multiple models, try to match the model selection to your physical device. If the device you are using is not represented in the model drop down, select a model that conforms most closely to your target device. You can determine this from the driver help specific to each available model. In some drivers, a model selection of Open is available. For these drivers the Open selection can be used to communicate without knowing the specific details of your target device.

The **Device ID** parameter can be used to specify the driver specific station or node for a given device. Depending on the communication driver you are using the type of ID entered will vary. For many communication drivers the ID is a numeric value. As shown in the first dialog above, when a driver supports a numeric ID the menu option can be used to enter a numeric value. Additionally the format of the entered numeric value can be changed to suit the needs of either the application or the characteristics of the chosen communication driver. By default the format is set by the driver. Possible formats are Decimal, Octal, and Hexadecimal.

If the driver in use is either an Ethernet based driver or supports an unconventional station or node name, the second dialog above may be shown. In this case the Device ID is a TCP/IP ID. TCP/IP or UDP IDs consist of four values separated by periods. Each value has a range of 0 to 255. Some Device IDs are string based. Additionally there may be more parameters within the ID field depending on the communications driver you are using. Details on the specific nature of the Device ID can be found in the help file for the driver that you are using.

The **Enable data collection** parameter can be used to control the active state of this device. By default, device communications are enabled. However, if you need to disable a physical device for servicing, you can use this parameter to disable the device. Once a device is disabled no communications with that device will be attempted. From a client standpoint, the data will be marked as invalid and write operations will not be accepted. Due to the full time online nature of the server this parameter can be changed at any time. This can be done through the menu selection shown here or by accessing the [System Tags](#) for this device.

The **Simulate Device** parameter can be used to place this device into a simulation mode. While in simulation mode, the driver will not attempt to communicate with the physical device, however, the server will continue to return valid OPC data. Unlike the **Enable data collection** parameter that stops physical communications with a given device and places the OPC data into an error state, the **Simulate Device** stops physical communications with the device but allows OPC data to be returned to your OPC client as valid data. While in Simulation mode, the server will treat all data for this device as reflective. This means that whatever is written to the simulated device will be read back. While in simulation mode every OPC item is treated individually and the memory map for these items are based on the group update rate. The data will not be saved if the server removes the item such as when the server is reinitialized. By default, simulation mode is disabled.

Caution: Simulation mode is for test and simulation purposes only.

Note 1: Due to the full time online nature of the server, this parameter can be changed at any time. This can be done through the menu selection shown here or by accessing the [System Tags](#) for this device. The system tags also allow this parameter to be monitored from your OPC client. You may turn off the ability to write to system tags under [OPC Settings](#) in the server.

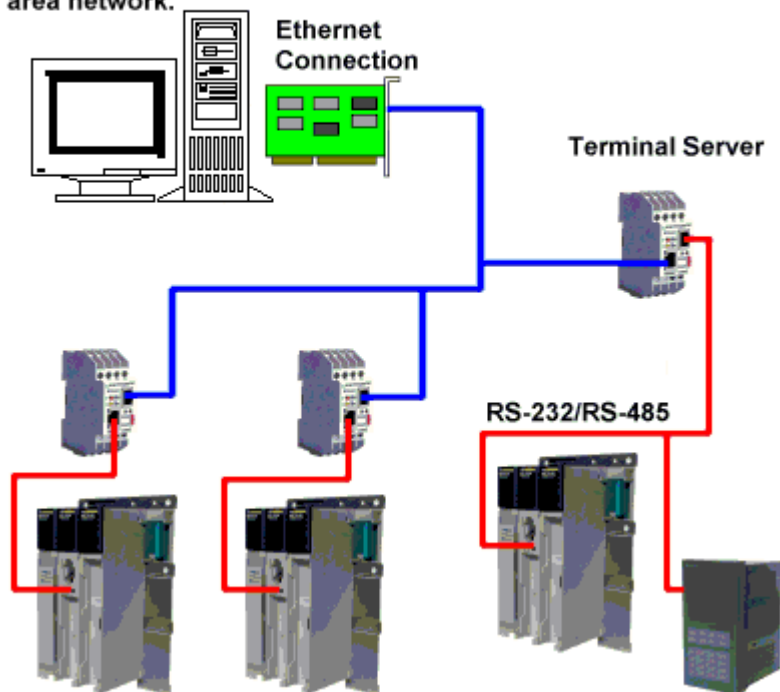
Note 2: With the server's online full time operation, these parameters can be changed at any time. This includes changing the device name, which can prevent clients from registering data with the server. If a client has already acquired an item from the server before you change the device name those items will be unaffected. However after you change the device name, if the client application releases the item and attempts to reacquire using the old device name the item will not be accepted. With this in mind you don't want to make changes to parameters like device name once you have a large client application developed. The Device ID parameter can be changed at any time and will take effect immediately. If the chosen communications driver supports multiple device models, the model selection can only be changed if there are currently no client applications connected to the device. To prevent operators from making any changes, use the [User Manager](#) to restrict access rights to server features.

Device Properties - Ethernet Encapsulation

The Ethernet Encapsulation mode has been designed to provide communications with serial devices connected to terminal servers on your Ethernet network. A terminal server is essentially a virtual serial port. The terminal server converts TCP/IP messages on your Ethernet network to serial data. Once the message has been converted to a serial form you can connect standard devices that support serial communications to the terminal server. The following diagram provides a demonstration of how the Ethernet Encapsulation mode should be employed:

Note: For unsolicited drivers that support Ethernet encapsulation you will configure the port and the protocol settings at the channel level. This will allow the driver to bind to the specified port and process incoming requests from multiple devices. An IP address is not entered at the channel since the channel will accept incoming requests from all devices.

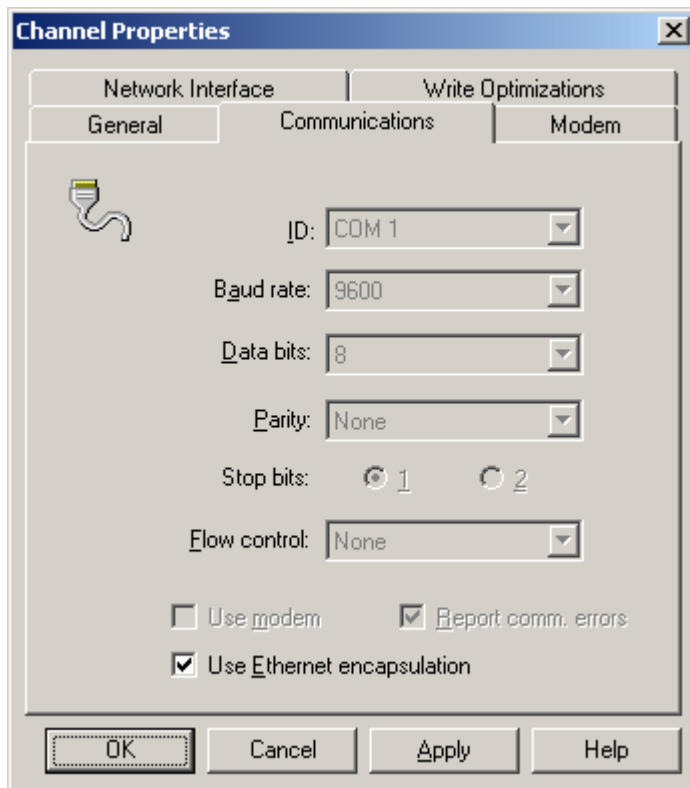
Ethernet Encapsulation can be used to access multiple Serial devices spread across a local area network.



Additional uses include use over wireless network connections such as 802.11b and CDPD packet networks. The Ethernet Encapsulation mode has been developed to support a wide range of serial devices. Using a terminal server device allows you to place RS-232 and RS-485 devices throughout your plant operations while still allowing a single localized PC to access the remotely mounted devices. The Ethernet Encapsulation mode allows an individual Network IP address to be assigned to each of the devices as needed. Using multiple terminal servers you can access hundreds of serial devices from a single PC.

Configuring Ethernet Encapsulation Mode:

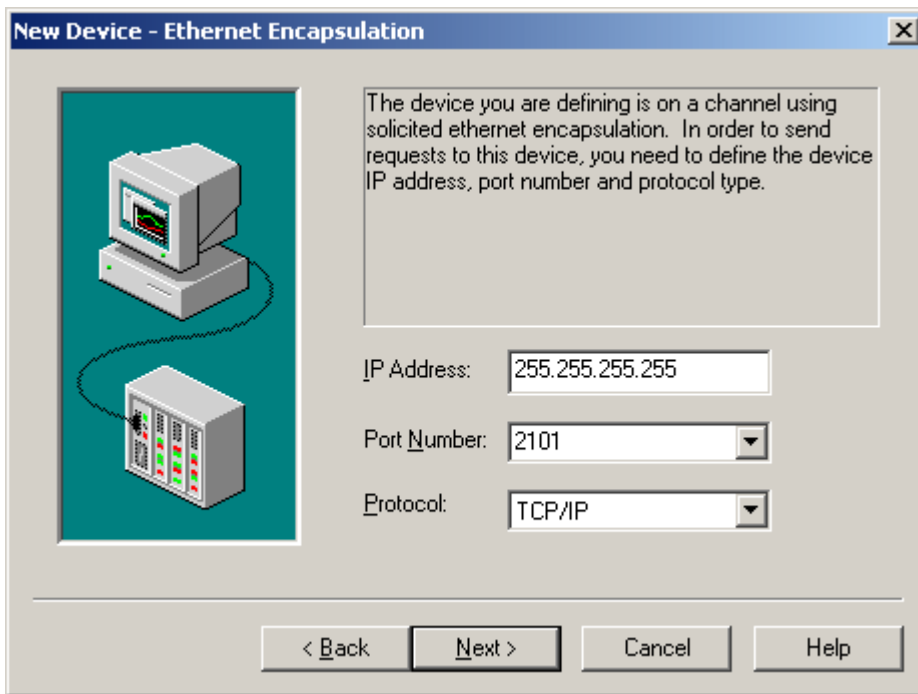
Using Ethernet Encapsulation mode starts by selecting the mode on the channel properties page. A driver that supports Ethernet Encapsulation will allow the Ethernet Encapsulation mode to be selected from the Com port ID selection as shown here:



If you desire to run the driver in Ethernet Encapsulation mode simply select it from the com port list. The multiple channel support of the server allows you to have up to 16 channels on each driver protocol. Using multiple channels you can have a channel defined to use the local PC serial port and have a channel defined to use Ethernet Encapsulation mode.

Important Note: When you select Ethernet Encapsulation mode you will notice that the serial port settings such as baud rate, data bits, and parity become grayed out. This occurs because these settings will not be used in Ethernet Encapsulation mode. The terminal server you are using must however have its serial port properly configured to match the requirements of the serial device you plan to attach to the terminal server.

Once you have configured the channel for Ethernet Encapsulation mode you will need to configure the device for Ethernet operation. When a new device is added to the channel the Ethernet Encapsulation settings will allow you to select an Ethernet IP address, an Ethernet Port number, and the Ethernet protocol to be used. The device dialog for Ethernet Encapsulation appears as follows:



The **IP Address** selection allows you to enter the four-field IP address of the terminal server to which this device is attached. IPs are specified as YYY.YYY.YYY.YYY The YYY designates the IP address (each YYY byte should be in the range of 0 to 255). Each serial device may have its own IP address or devices may have the same IP address if you have multiple devices multi-dropped from a single terminal server.

The **Port** selection allows you to configure the Ethernet port to be used when connecting to a remote terminal server.

The **Protocol** selection allows you to use either TCP/IP or UDP communications. The selection depends completely on the nature of the terminal server you are using. Consult your terminal server's documentation for more information on the protocol available. The default protocol selection is TCP/IP.

Note: It is important to remember that the Ethernet Encapsulation mode is completely transparent to the actual serial communications driver. With this in mind you must still configure the remaining device settings just as you would if you were connecting to the device directly on your local PC serial port.

Feature Note: With the server's online full time operation, these parameters can be changed at any time. To prevent operators from making any changes, use the [User Manager](#) to restrict access rights to server features.

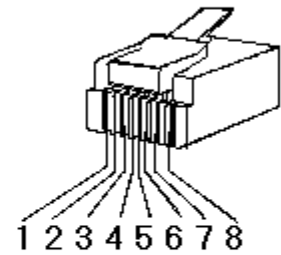
Cable Diagrams

Patch Cable (Straight Through)

TD +	1	OR/WHT	OR/WHT	1	TD +
TD -	2	OR	OR	2	TD -
RD +	3	GRN/WHT	GRN/WHT	3	RD +
	4	BLU	BLU	4	
	5	BLU/WHT	BLU/WHT	5	
RD -	6	GRN	GRN	6	RD -
	7	BRN/WHT	BRN/WHT	7	
	8	BRN	BRN	8	

RJ45 RJ45

10 BaseT



Crossover Cable

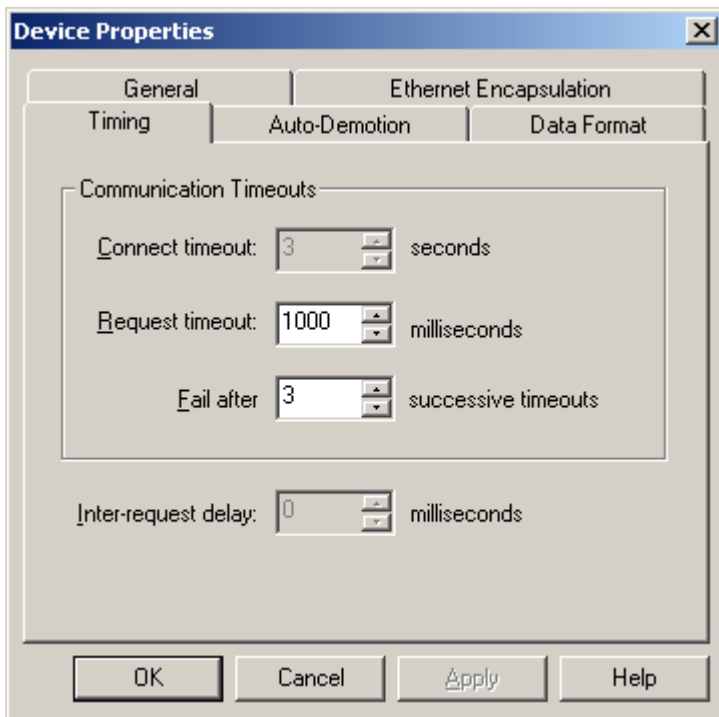
TD +	1	OR/WHT	GRN/WHT	1	TD +
TD -	2	OR	GRN	2	TD -
RD +	3	GRN/WHT	OR/WHT	3	RD +
	4	BLU	BLU	4	
	5	BLU/WHT	BLU/WHT	5	
RD -	6	GRN	OR	6	RD -
	7	BRN/WHT	BRN/WHT	7	
	8	BRN	BRN	8	

RJ45 RJ45

8-pin RJ45

Device Properties - Timing

Device timing parameters allow a driver's response to error conditions to be tailored to the needs of the application. In many cases the environment in which the application runs may require changes to the timing parameters. Factors such as electrically generated noise, modem delays, and bad physical connections can all influence how many errors or timeouts a communications driver may encounter. The timing parameters are specific to each device you configure.



The **Connection timeout** is used primarily by Ethernet based drivers. The connection timeout allows the time required to establish a socket connection to a remote device to be adjusted. In many cases the connection time to a device can take longer than normal communications request to that same device. The valid range is 1 to 30 seconds. The default is typically 3 seconds, but can vary depending on the specific nature of the chosen driver. This setting will be disabled if it is not supported by the driver.

UDP Note: Due to the nature of UDP connections, the connection timeout setting is not applicable when communicating via UDP.

The **Request timeout** is an interval used by all drivers to determine how long the driver will wait for a response from the target device. The request timeout interval has a valid range of 100 to 30000 milliseconds. The default is typically 1000 milliseconds but can vary depending on the specific nature of the chosen driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using the driver at lower baud rates, you may need to increase the timeout to compensate for the increased time required to acquire data.

The **Fail after** parameter is used to determine how many times the driver will retry a communications request before considering the request to have failed and the device to be in error. The valid range is 1 to 10 retries. The default is typically 3 retries but can vary depending on the specific nature of the chosen driver. The number of retries you configure for the application is dependent largely on your communications environment.

A Note About Timeouts

If your environment is prone to noise induced communications failures, you may want to set up your devices for [auto-demotion](#) or increase the number of retries the driver performs. If you decide to increase the number of retries, keep in mind that when the driver does encounter a communication issue, it will attempt to reacquire the data for any lost requests. Based on the "Request timeout" and the "Fail after" count, the driver will pause on a specific request until either the device responds or the timeout and retries have been exceeded thus, potentially decreasing the communications of other devices that may be configured on that channel. In this situation, it may be more appropriate to utilize the [auto-demotion](#) functionality to optimize communications with other devices on the same channel.

The **Inter-request delay** is used to determine how long the driver will wait before sending the next request to the target device. It will override the normal polling frequency of tags associated with the device, as well as one-shot reads and writes. This delay can be useful when dealing with devices with slow turnaround times, and in cases where network load is a concern. Be aware that configuring a delay for a device will affect communications with all other devices on the channel. Because of this, it is recommended that you segregate any device(s) that requires an inter-request delay to a separate channel if possible. The inter-request delay has a valid range of 0 to 30000 milliseconds. The default is 0, indicating that there will be no delay between requests with the target device. This setting will be disabled if it is not supported by the driver. Please refer to your driver's specific help to verify if this is supported.

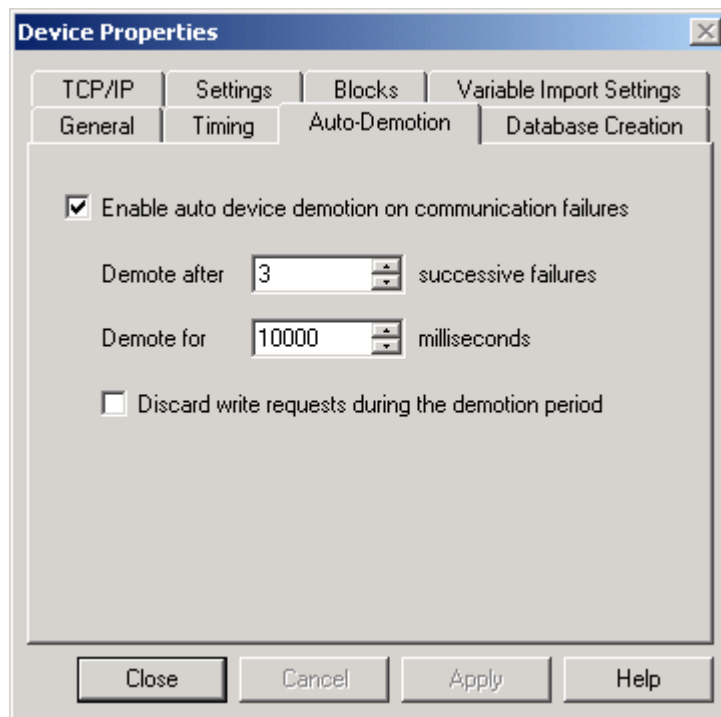
Feature Note:

- You can determine when communication errors are occurring by using the [_System Tag](#), `_Error` for this device.
- With the server's online full time operation, these parameters can be changed at any time. To prevent operators from changing these parameters use the [User Manager](#) to restrict access rights to server features.

Device Properties - Auto-Demotion

Device auto-demotion parameters allow a driver to temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device off-line, the driver can continue to optimize its communications with other devices on the same channel by stopping communications with the non-responsive device for a specific time period. After the specific time period has been reached, the driver will re-attempt to communicate with the non-responsive device. If the device is responsive, the device will be placed on-scan, otherwise it will restart its off-scan time period.

Auto-demotion can be enabled by checking the **Enable auto device demotion on communication failures** as shown below.



The **Demote after** parameter indicates how many successive cycles of request timeouts and retries will occur before placing the device off-scan. The valid range is 1 to 30 successive failures. The default is 3 successive failures.

The **Demote for** parameter indicates how long the device should be placed off-scan when the "Demote after" parameter has been reached. During this period no read requests will be sent to the device and all data associated with these read requests will be set to bad quality. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds. When this period expires, the driver will place the device on-scan allowing for another attempt at communications.

The **Discard write requests during the demotion period** parameter allows you to control whether or not write requests should be attempted during the off-scan period. The default setting is to always send write requests regardless of the demotion period. If you choose to discard writes, the server will automatically fail any write request received from a client and will not post an "Unable to write..." messages to the server event log.

To determine when a device is off-scan by monitoring its demoted state, use the `_AutoDemoted` [System Tag](#) for this device.

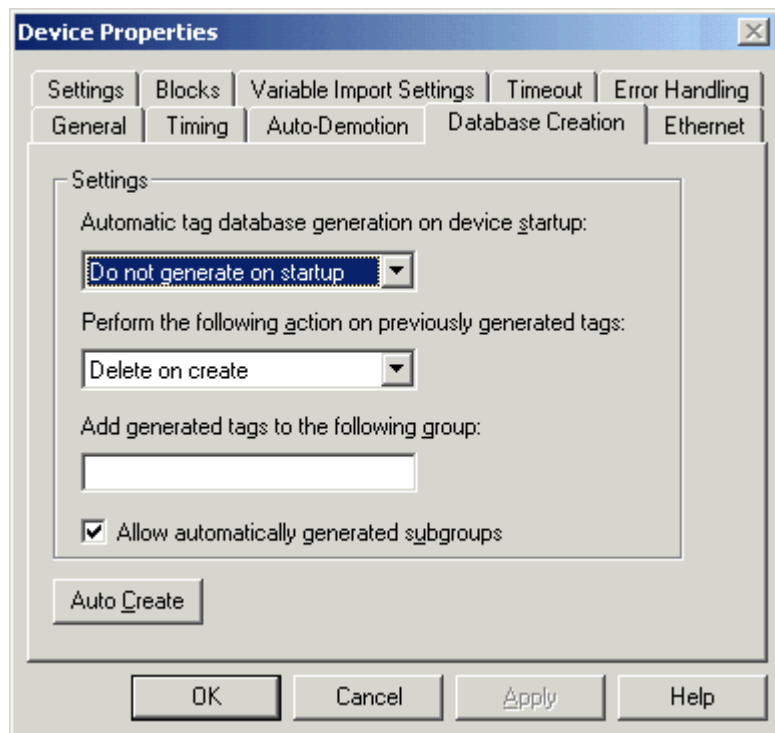
Automated OPC Tag Database Generation

The Automatic OPC Tag Database Generation features of the server have been designed to make the setup of your OPC application a Plug and Play operation. For communication drivers that support this feature, you can configure them to automatically build a list of OPC tags within the server that correspond to device specific data. The automatically generated OPC tags can then be browsed from your OPC client. The OPC tags that are generated are dependent upon the nature of the supporting driver.

If the target device supports its own local tag database, the driver will read the device's tag information and use this data to generate OPC tags within the server. If the device does not natively support its own named tags, the driver will create a list of tags based on information specific to the driver. An example of these two conditions may be as follows:

1. If a data acquisition system supports its own local tag database, the communications driver will use the tag names found in the device to build the server's OPC tags.
2. If an Ethernet I/O system supports detection of its own available I/O module types, the communications driver will automatically generate OPC tags in the server that are based on the types of I/O modules plugged into the Ethernet I/O rack.

The mode of operation for automatic tag database generation is completely configurable. The following dialog is used to configure how the server and the associated communications driver will handle Automatic OPC Tag Database Generation:



The **Automatic tag database generation on device startup** setting is used to configure when OPC tags will be automatically generated. There are three possible selections:

- **Do not generate on startup**, the default selection, prevents the driver from adding any OPC tags to the tag space of the server.
- **Always generate on startup** causes the driver to always evaluate the device for tag information and to add OPC tags to the tag space of the server each time the server is launched.
- The final selection **Generate on first startup** causes the driver to evaluate the target device for tag information the first time this project is run and to add any OPC tags to the server tag space as needed.

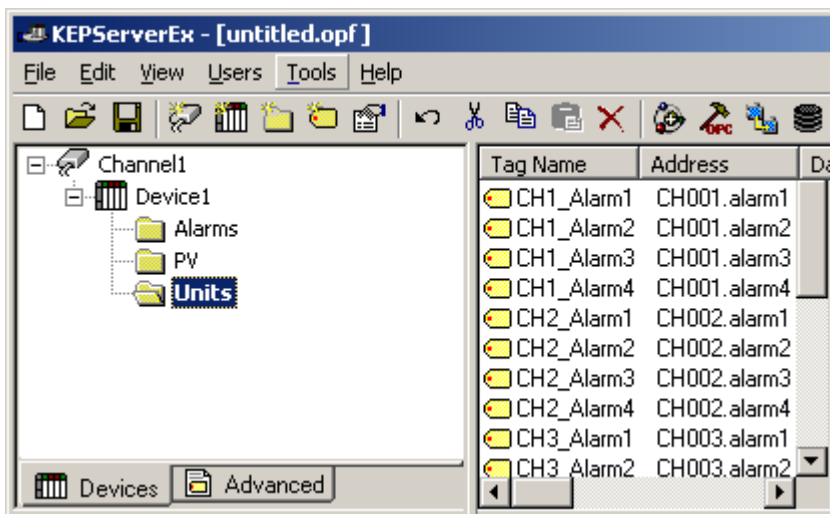
When the automatic generation of OPC tags is selected, any tags that are added to the server's tag space must be saved with the project. You can configure your project to auto save from the [Tools|Options](#) menu.

When automatic tag generation is enabled, the server needs to know what to do with OPC tags that it may have added from a previous run or with OPC tags that have been added or modified after the communications driver added them originally. The **Perform the following action** setting is used to control how the server will handle OPC tags that were automatically generated and currently exist in the project. This feature prevents automatically generated tags from accumulating in the server. For example, using the Ethernet I/O example mentioned above, if you continued to change the I/O modules in the rack with the server configured to always generate new OPC tags on startup, new tags would be added to the server every time the communications driver detected a new I/O module. If the old tags were not removed, many unused tags could accumulate in the server's tag space. **Perform the following action** is used to tailor the server's operation to best fit the application's needs. Descriptions of the selections are as follows:

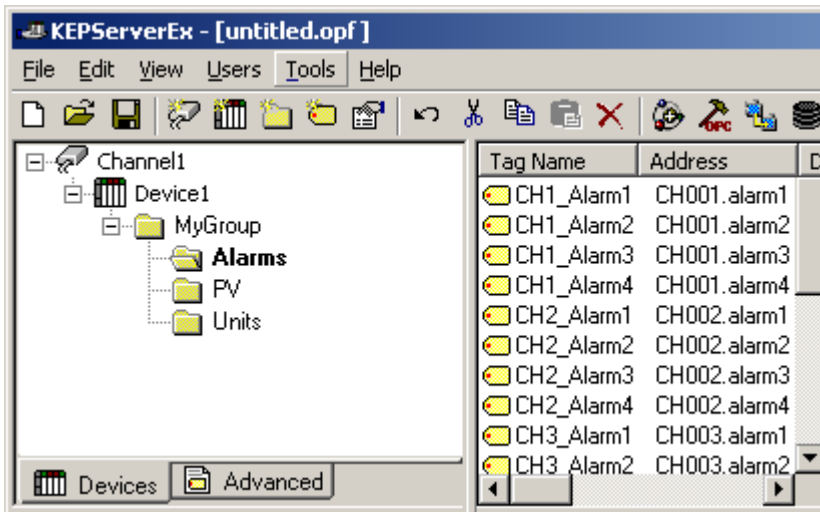
1. **Delete on create**, the default condition, deletes any tags that had previously been added to the tag space before the communications driver adds any new tags.
2. **Overwrite as necessary** instructs the server to remove only those tags that the communications driver is replacing with new tags. Any tags that are not being overwritten will remain in the server's tag space.
3. **Do not overwrite**, will prevent the server from removing any tags that had been previously generated or may have already existed in the server. With this selection, the communications driver can only add tags that are completely new.
4. **Do not overwrite, log error**, has the same effect as the third; however in addition, an error message will be posted to the server's event log when a tag overwrite would have occurred.

Note: The removal of OPC tags effects tags that have been automatically generated by the communications driver and any tags that have been added using names that match generated tags. It is recommended that users avoid adding tags to the server using names that match tags that may be automatically generated by the driver.

Add generated tags to the following group can be used to keep automatically generated tags from mixing with tags that have been entered manually. This parameter is used to specify a subgroup that will be used when adding all automatically generated tags for this device. The name of the subgroup can be up to 256 characters in length. The following screens demonstrate how this parameter works, i.e., where automatically generated tags are placed in the server's tag space. As shown below, this parameter provides a root branch to which all automatically generated tags will be added.



Add generated tags to the following group is blank.



"MyGroup" was entered in the "Add generated tags to the following group" field.

The **Allow automatically generated subgroups** setting controls whether or not the server will automatically create subgroups for the auto-generated tags.

Allow automatically generated subgroups	
Checked (default)	<p>The server will auto-generate the device's tags and organize them into subgroups. In the server project, the resulting tags will retain their tag names.</p>
Unchecked	<p>The server will auto-generate the device's tags in a simple list without any subgrouping. In the server project, the resulting tags will be named with the address value; i.e., tag names will not be retained during the auto-generation process. In the example shown below, note how the tag names were created using the tag's address.</p> <p>Note: As the server is generating tags, if a tag would be assigned the same name as an existing tag, the system will automatically increment to the next highest number so that the tag name is not duplicated. For example, if the auto-generation process were to create a tag named AI22 but there already existed a tag with that name, the auto-generation process would create the tag as AI23.</p>

Auto Create is used to manually initiate the creation of automatically generated OPC tags. If the device's configuration has been modified, clicking **Auto Create** will force the communications driver to reevaluate the device for possible tag changes. **Auto Create** can be accessed from the [System Tags](#) for this device, which allows OPC client application to

initiate tag database creation.

Note: With the server's online full time operation, these parameters can be changed at any time. To prevent operators from changing these parameters, use the [User Manager](#) to restrict access rights to server features.

New Device - Summary

The Device Summary page is used to review the selections that have been made for the device. If any of the selections displayed in the summary report need to be changed, simply click on the Back button until the required dialog is displayed.

Tag Functions

A tag represents addresses with in the PLC or other hardware device that the server communicates with. The server allows both dynamic tags, (tags created in the client that access addresses directly) and user defined static tags. User defined static tags are created in the server and benefit the user by allowing the tag to be browsed from OPC clients that support tag browsing. The user-defined tags also support tag scaling.

See Also:

[Tag Properties](#)

[Scaling](#)

[Tag Group Properties](#)

[Tag Selection](#)

Tag Properties

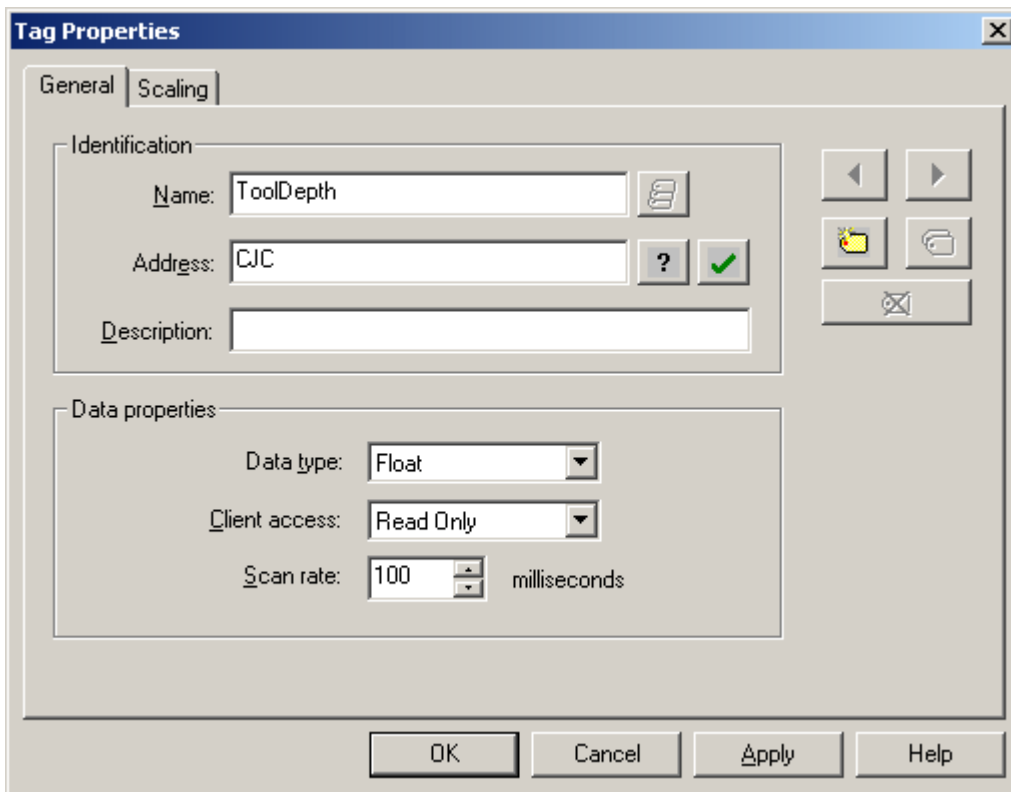
The server allows both

[dynamic](#)

tags, (tags entered directly

in the OPC client that specify device data) and

[static user defined](#) tags. User defined tags have the benefit of allowing the tag to be browsed from an OPC client that supports tag browsing. User defined tags also support tag scaling. Unlike many of the dialogs you will find in the server, the Tag Properties dialog has a number of features that are driven by icons. To learn about these features click on the figure below. The properties that require user input will be described below.



The **Tag Name** parameter is used to enter the string that will represent the data available from this tag. The tag name can be up to 256 characters in length. While using long descriptive names is generally a good idea, keep in mind that some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The tag name is part of the OPC browse data. Tag names must be unique within a given device branch or tag group branch. If the application is best suited for using blocks of tags with the same names, then use [tag groups](#) to segregate the tags.

The **Address** parameter is used to enter the desired driver address for this tag. The format of the address entered here is based entirely upon the driver being used. To determine how an address should be entered you can use the Hints button next to the address parameter. Hints provide a quick reference guide to the address format of the driver. The primary driver help can also be invoked from the hints dialog if needed. The address entered can be up to 128 characters in length. Once an address has been entered, it can be tested by using the Check Address button. When pressed, the check address button attempts to validate the address with the driver. If the driver accepts the address as entered no message will be displayed. A popup will inform of any error. Keep in mind that some errors will be related to the data type selection and not the address string.

The **Description** parameter is used to attach a comment to this tag. A string of up to 64 characters can be entered for the description. If you are using an OPC client that supports Data Access 2.0 Tag Properties the description parameter will be accessible from the Item Description property of the tag.

The **Data type** selection is used to specify the format of this tag's data as it is found in the physical device. In most cases this is also the format of the data as it returned to the client. The data type setting is an important part of how a communication driver reads and writes data to a device. For many drivers the data type of a particular piece of data is rigidly fixed. In these cases the driver knows what format it needs to use when reading data from the device. In some cases however, the interpretation of device data is largely in the user's hands. An example would be a device that uses 16 bit data registers. Normally this would indicate that the data is either a Short or Word. Many register-based devices also support values that span two registers. In these cases the double register values could be a Long, Dword, or Float. When the driver being used supports this level of flexibility, users must tell it how to read data for this tag. By selecting the appropriate data type, the driver is being told to read either one register or two or possibly a Boolean value. The driver governs the data format being chosen. The driver's help system can be accessed through the Hints button to get specific help on what data types are available for a given driver. Available data type selections are as follows.

Default - This type allows the driver to choose its default data type. See the specific driver help for details.

Boolean - Single bit data On or Off

Char - Signed 8 bit data
Byte - Unsigned 8 bit data
Short - Signed 16 bit data
Word - Unsigned 16 bit data
Long - Signed 32 bit data
Dword - Unsigned 32 bit data
Float - 32 bit Real value IEEE format
Double - 64 bit Real value IEEE format
String - Null terminated ASCII string
BCD - Two byte-packed BCD value range is 0-9999
LBCD - Four byte-packed BCD value range is 0-99999999.

The **Client access** selection is used to specify whether this tag is **Read Only** or **Read/Write**. By selecting **Read Only**, users can prevent client applications from changing the data contained in this tag. By selecting **Read/Write**, users allow client applications to change this tag's value as needed. The **Client access** selection also has an effect upon how this tag will appear in the browse space of an OPC client. Many OPC client applications allow you to filter tags based on their attributes. Changing the access method of this tag may change how and when the tag will appear in the browse space of your OPC client.

The **Scan rate** parameter is used to specify the update interval for this tag when used with a non-OPC client. OPC clients can control the rate at which data is scanned by using the update rate that is part of all OPC groups. Normally non-OPC clients don't have that luxury. The server is used to specify an update rate on a tag per tag basis for non-OPC clients. Using the scan rate you can tailor the bandwidth requirements of the server to suit the needs of the application. If, for example, you need to read data that changes very slowly there is no reason to read the value very often. Using the scan rate this tag can be forced to read at a slower rate reducing the demand on the communications channel. The valid range is 10 to 99999990 ms., with a 10 ms. increment. The default is 100 milliseconds.

Note: With the server's online full time operation, these parameters can be changed at any time. Changes made to Tag Properties will take effect immediately; however, OPC clients that have already connected to this tag will not be effected until they release and reacquire this tag. To prevent operators from making any changes, use the [User Manager](#) to restrict access rights to server features.

Dynamic Tags

The second method of defining tags is called Dynamic Tag addressing. Dynamic tags are used to define tags solely in the client application. Instead of creating a tag item in the client that addresses another tag item created in the server, tag items created in the client directly accesses the device driver's addresses. On client connect, the server will create a virtual tag for that location and start scanning for data automatically.

To specify an optional data type, append one of the following strings after the '@' symbol:

Boolean
Byte
Char
Short
Word
Long
DWord
Float
Double
BCD
LBCD
String

If the data type is omitted, the driver will choose a default data type based on the device and address you are referencing. The default data types for all locations are documented in the individual driver help files. If the data type specified is not valid for the device location, the server will not accept the tag and an error will be posted in the Event Log window.

Examples

Scan the 16-bit location 'R0001' on the Simulator device. The following dynamic tag examples assume users are using the project created as part of this example.

DDE Client Using Dynamic Addressing:

1. Define a tag in the DDE client.
2. Create a DDE link for the client tag as follows: [<DDE service name>|_ddedata!Device1.R0001@Short](#)
3. Run the client project. The default data type for address R0001 in the Simulator device is 'Word'. To override the default data type to "Short", the "@Short" has been appended.

Example: OPC Client Using Dynamic Addressing:

1. Start the OPC client application and connect to the server.
2. Create a channel (called channel1) and device (called Device1) using the simulator driver.
3. In the client application, define an item name of "**Channel1.Device1.R0001@Short**".
3. The client project will automatically start receiving data. The default data type for address R0001 in the Simulator device is 'Word'. To override this, the "@Short" has been appended to select a data type of Short.

When using dynamic tags in an OPC client application, the use of the @[Data Type] modifier is not normally required. OPC clients can specify the desired data type as part of the request when registering a link for a specific data item. The data type specified by the OPC client will be used if the communication driver supports the data type. The @[Data Type] modifier can be handy when you want to ensure that a communication driver interprets a piece of data exactly as you desire.

For non-OPC clients, you can override the update rate on a per-tag basis by appending @[Update Rate]. For example, [<DDE service name>|_ddedata!Device1.R0001@500](#) to override just the update rate, or [<DDE service name>|_ddedata!Device1.R0001@500,Short](#) to override both update rate and data type.

See Also: [Static Tags \(User Defined\)](#) and [Designing a Project: Adding Tags to the Project](#).

Note 1: The server creates a special Boolean tag for every device in a project that can be used by a client to determine whether that device is functioning properly. To use this tag, specify the item in the link as Error. The value of this tag is zero if the device is communicating properly otherwise it is one.

Note 2: If a device address is used as the item of a link such that the address matches the name of a user defined tag in the server, the link will reference the address pointed to by the user defined tag.

Note 3: Static tags must be used in order to scale data in the server.

Static Tags (User Defined)

There are two ways to get data from a device to your client application using the server. The first method and most common method requires that you define a set of tags in the server project and then use the name you assigned to each tag as the item of each link between the client and the server. The primary benefit to this method is that all user-defined tags are available for browsing within most OPC clients (its best to see if your client can browse or import tags from the server before deciding on creating static tags). Additionally, user defined tags also support [scaling](#).

See Also:

[Dynamic Tags](#)

[Designing a Project \(Adding Tags to the Project\)](#)

[Designing a Project \(Adding Tag Scaling\)](#)

Scaling

The server supports tag scaling. Scaling allows raw data from the device to be scaled to a more appropriate range for the application. There are two types of scaling: **Linear** and **Square Root**.

To enable scaling operations for this tag, select either **Linear** or **Square Root**. Scaling is not enabled while **None** is checked.

Type	Formula for Scaled Value
Linear	$\frac{((\text{ScaledHigh} - \text{ScaledLow}) / (\text{RawHigh} - \text{RawLow})) * (\text{RawValue} - \text{RawLow})}{1} + \text{ScaledLow}$
Square root	$(\text{Square root}((\text{RawValue} - \text{RawLow}) / (\text{RawHigh} - \text{RawLow})) * (\text{ScaledHigh} - \text{ScaledLow})) + \text{ScaledLow}$

The **Raw Value Range** settings allow you to specify the range of raw data from the device. The raw value **High** setting must be greater than the **Low** setting. The valid range will depend upon the data type of the raw tag value. For example, if the raw value is Short, the valid range of the raw value would be from -32768 to 32767.

The **Scaled Value Range** settings allow you to specify the range of the resulting scaled value.

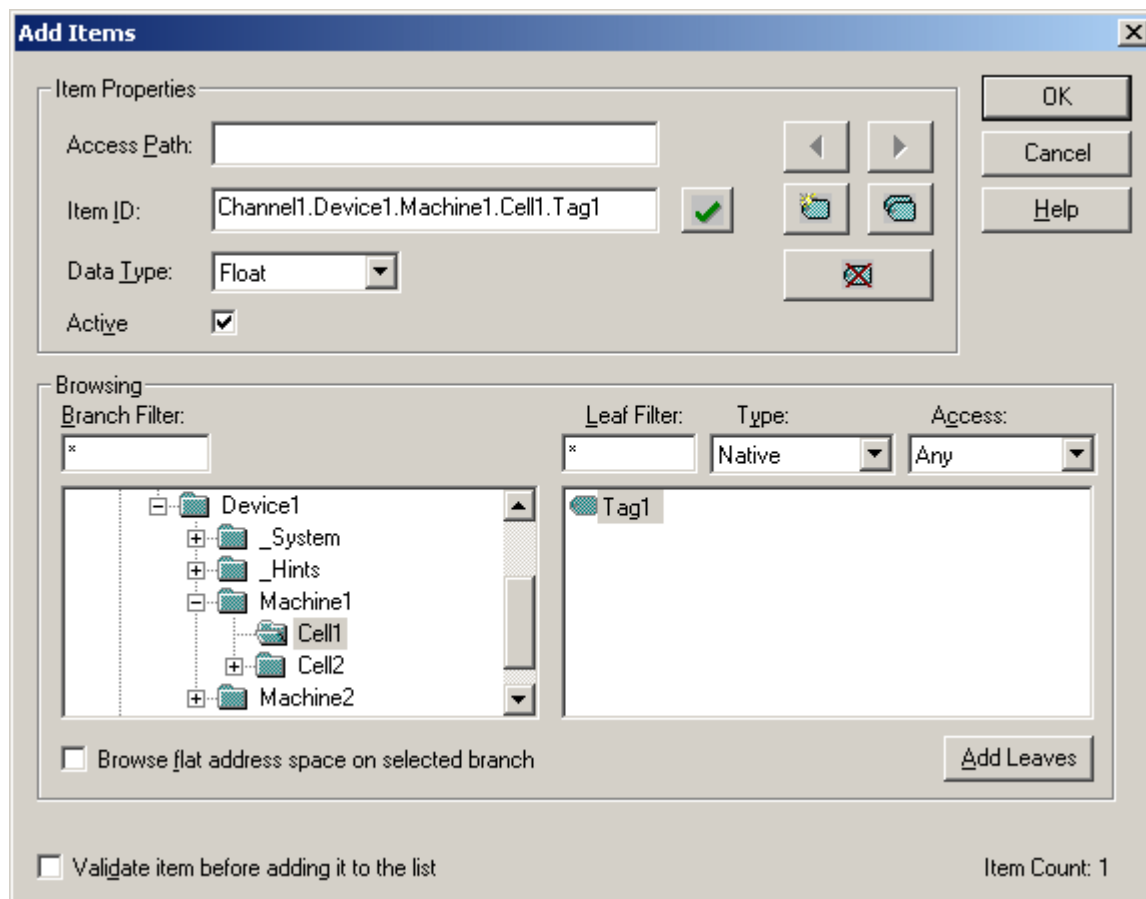
Data Type	Normally a scaled value is assumed to result in a floating-point value. The server does not make that assumption for you. The data type can be set to any valid OPC data type. This gives you the ability to scale from a raw data type such as Short to an engineering value with a data type of Long if needed. The default scaled data type is Double.
High and Low	The scaled value High must be greater than the scaled value Low . The valid range will depend upon the data type of the scaled value. For example, if the scaled data type is set to Long, then the valid range would be -2147483648 to 2147483647.
Clamp	In many cases the raw data from the device exceeds the range you have specified for the raw data. When this occurs, the scaled value is also forced outside of the range you have established. To prevent this, the High and Low Clamps can be used to constrain the scaled value to the range specified.
Units	The server also allows a unit's string to be assigned to a scaled tag. The units' string can be up to 32 characters long.
Negate scaled value	Forces the resulting value to be negated before being passed to the client.

Notes:

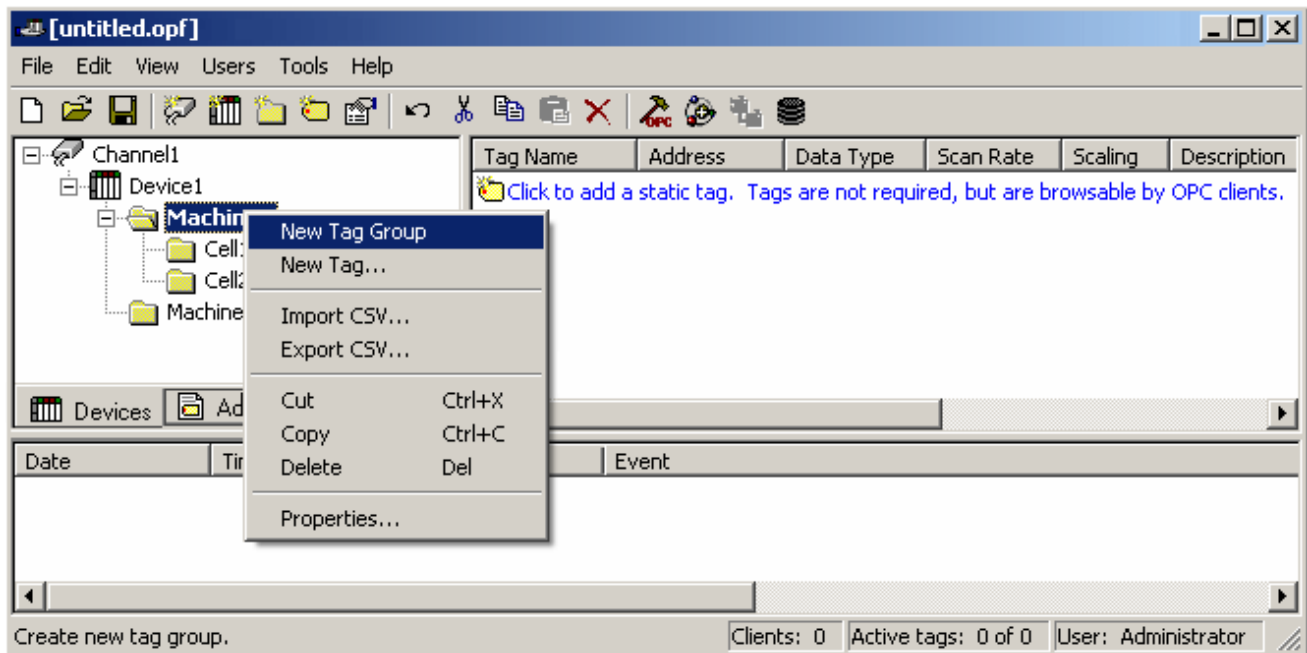
- The OPC server supports the OPC Tag Properties available in the 2.0 Data Access specifications. If the OPC client that you are using supports these properties, it can automatically configure the range of objects like user input objects or displays, using the Scaling settings entered here.
- With the server's full time, online operation, you can change any of these settings at any time. Changes made to Tag Properties such as scaling will take effect immediately; however OPC clients that have already connected to this tag will not be affected until they release and reacquire this tag.
- To prevent any unauthorized operator from changing these parameters, use the [User Manager](#) to restrict access rights to server features.

Tag Group Properties

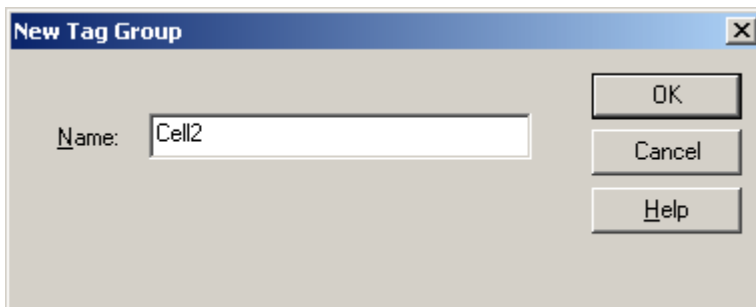
The server allows tag groups to be added to your project. Tag groups allow you to tailor the layout of OPC data in logical groupings that fit the needs of the application. Using tag groups allows multiple sets of identical [tags](#) to be added under the same device. This can be very convenient when a single device handles a number of similar machine segments. From an OPC client standpoint, the use of tag grouping allows you to segregate your OPC data into smaller tag lists, which can make finding a specific tag easier when browsing the server. Using the supplied OPC Quick client, the Cell1 and Cell2 tag groups shown in the following figure can simplify OPC client browsing:



To add a new tag group to your project, right-click on either an existing device or tag group branch and select **New Tag Group** from the context menu as shown in the figure below:



When adding a new tag group to your project you will be presented with the following dialog:

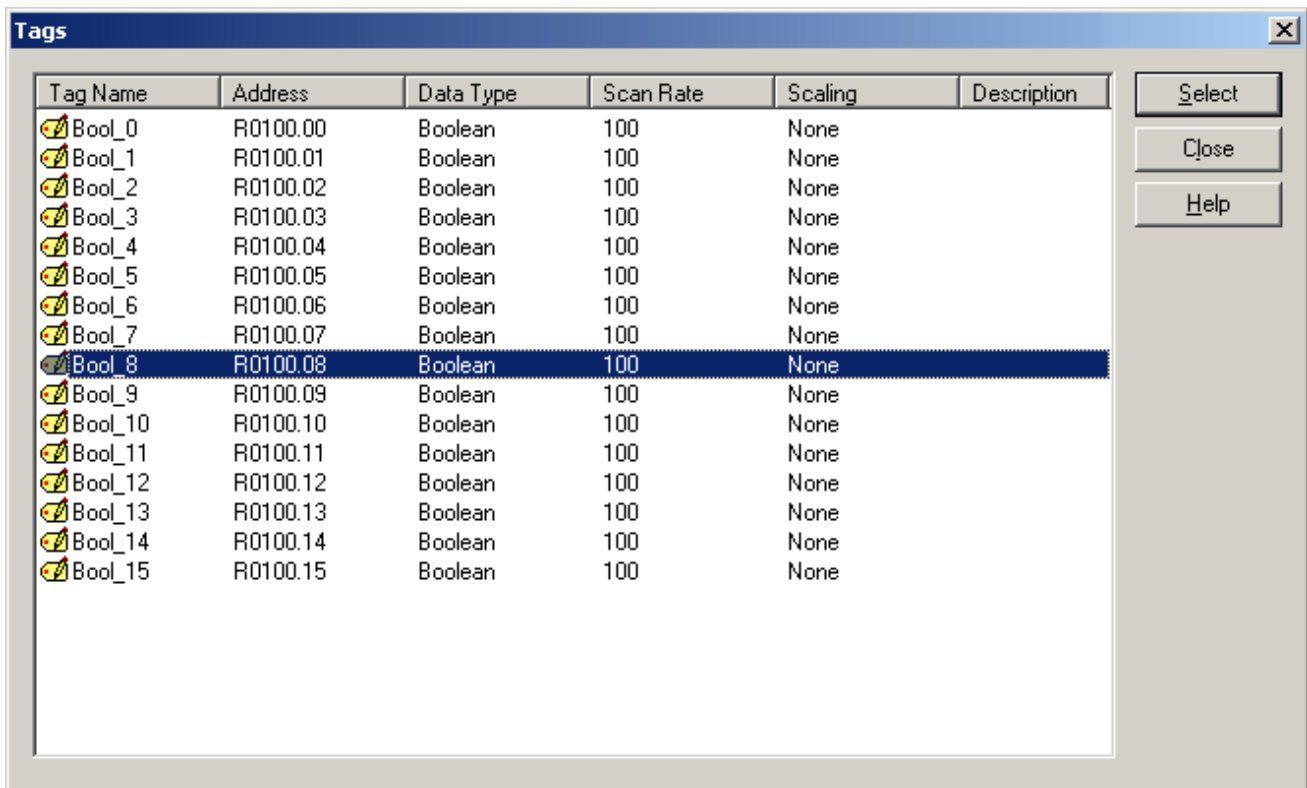


Tag groups can be added at any level from the device level down. Multiple tag groups can be nested together to fit the needs of the application. As seen in the OPC Quick Client dialog above, the fully qualified OPC item path is "**Channel1.Device1.Machine1.Cell1.Tag1**". For this OPC item "**Machine1**" and "**Cell1**" segments are nested tag groups.

Feature Note: With the server's online full time operation, these parameters can be changed at any time. Changes to tag groups will take effect immediately. If you change the name of a tag group, OPC clients that have already used that tag group as part of an OPC item request will not be affected until they release the item and attempt to reacquire it. New tag groups added to your project will immediately allow browsing from an OPC client. To prevent operators from making any changes, use the [User Manager](#) to restrict access rights to server features.

Tag Selection

The Tag Selection dialog allows users to quickly search for and find an existing tag to select it for editing.

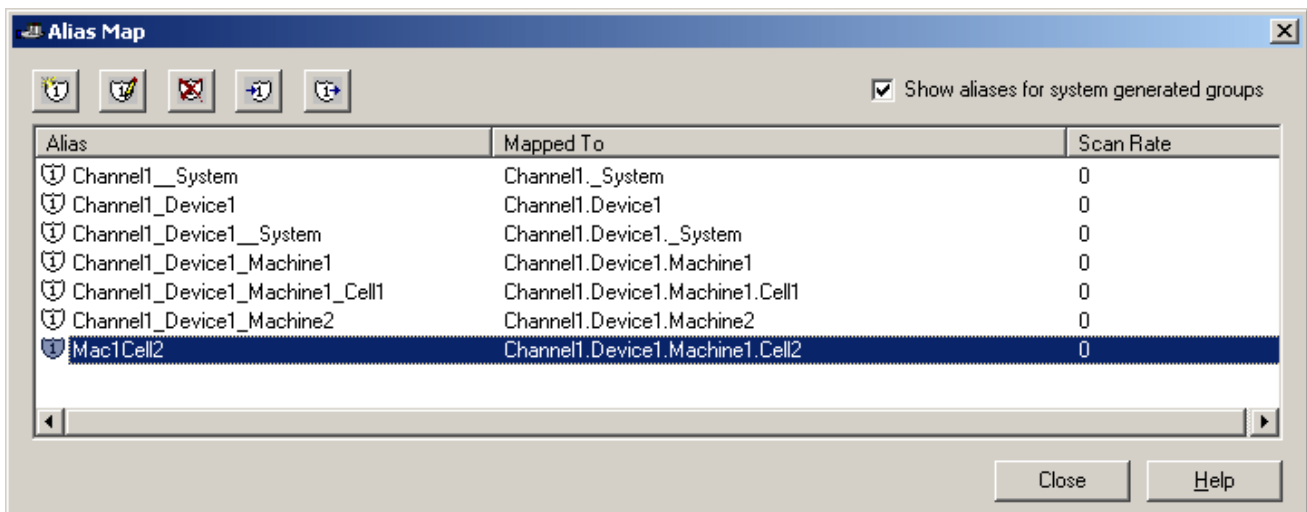


Note: Once a tag is selected, the page will return to the [Tag Properties](#) dialog.






Alias Map

The Alias Map provides both a mechanism for backwards compatibility with legacy server applications as well as a way to assign simple alias names to complex tag references (this is especially useful in client applications that limit the size of tag address paths). The latest version of the server will automatically create the alias map for you. You can add your own alias map entries to compliment those created by the server or even filter out the server created aliases so that you only see your own.

Alias map elements can be added, edited, deleted, exported and imported by clicking on the appropriate icon buttons in the alias map window. The [alias properties dialog](#) allows an alias to be added or edited.



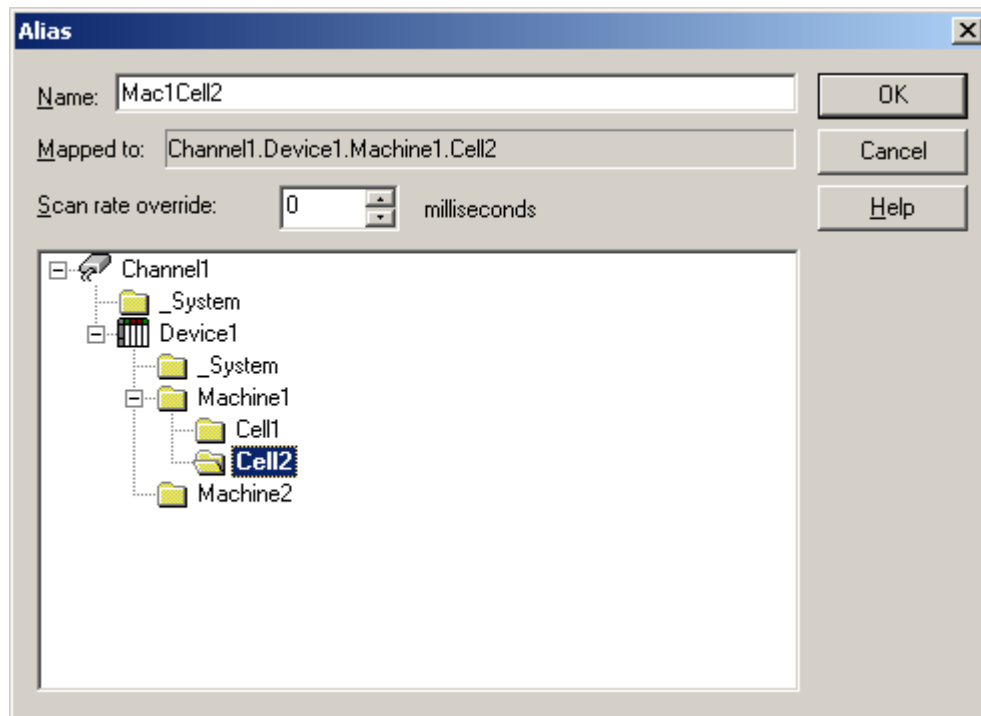
Note: The dialog above shows the various alias map entries generated for the server project.

- **Create** a new Alias by selecting the  button.
- **Edit** an existing alias by selecting the alias from the list and clicking the  button.
- **Delete** manually created aliases by clicking the  button.
- **Import** alias map as a .CSV file by clicking the  button.
- **Export** alias map as a .CSV file by clicking the  button.
- **Show aliases for system generated groups** check box allows the server created aliases to be shown or hidden.

See Also: [Alias Properties](#) and [Creating an Alias](#).

Alias Properties

The alias map allows a way to assign simple alias names to complex tag references that can be used in client applications. An alias is constructed by entering an alias name and then clicking on the desired device name or group name.



Alias Name: The alias name can be up to 256 characters long. The alias name entered must be unique in the alias map.

Alias Map Browse: The alias map will not allow tag items to be browsed from the alias table. The idea is to create a simple name that replaces the address that leads up to the tag so that it is easier to address items in a client application that does not allow you to browse for tags.

The **Scan rate override** selection can be used to specify an update rate that will be applied to all [DDE](#) | [FastDDE](#) / [SuiteLink](#) and most other non-OPC tags accessed using this alias map entry. This setting is equivalent to the Topic update rate found in many DDE only servers. The valid range is 0 to 99999990 milliseconds. The default is 0 milliseconds. When set to 0 milliseconds the server will observe the DDE scan rate set at the individual tag level using

the [Tag Properties](#) dialog.

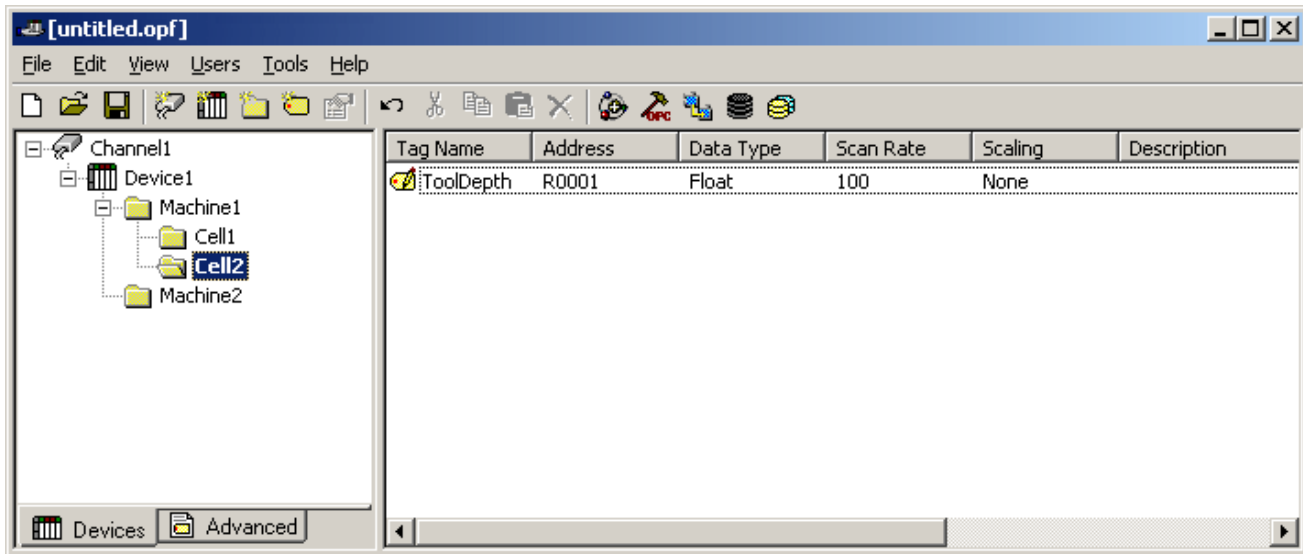
Once the desired path has been selected and the DDE scan rate is set, click **OK** to complete the alias. More alias map elements can be entered by returning to the [Alias Map](#) dialog.

See Also: [Alias Map](#) and [Creating an Alias](#).

Creating an Alias

Example of a Complex Tag Reference

The following figure is an example of a complex tag reference in the server.



In this example, if a DDE link to an application were needed for the tag "ToolDepth", the DDE link would need to be entered as follows:

= [<DDE service name>](#) |_ddedata!Channel1.Device1.Machine1.Cell2.ToolDepth

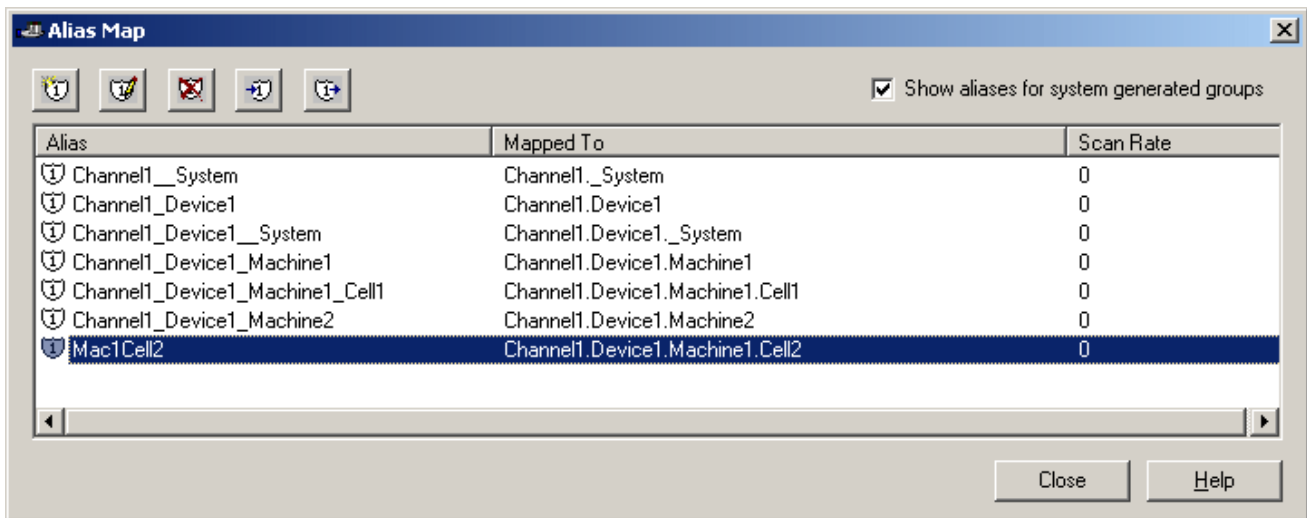
It's easy to see that the simple Application|Topic!ItemName format of a DDE link still exists, but the content has become a little more complex with the addition of optional tag groups and the required channel name as part of the topic. The alias map allows a shorter version of the reference to be used in DDE client applications.

Creating aliases for complex address paths

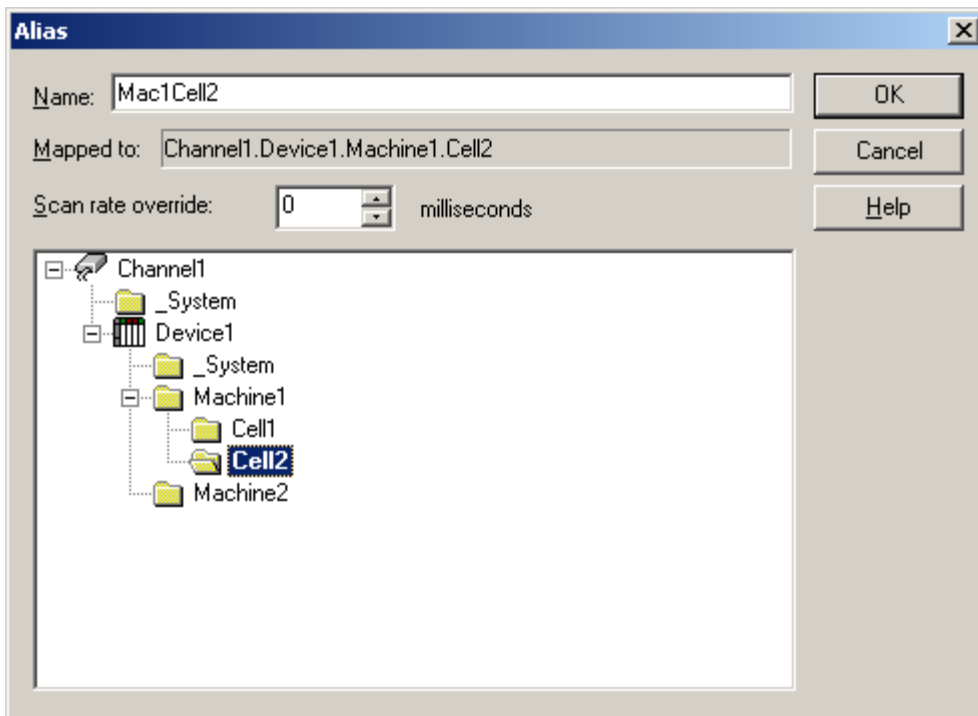
The following instructions demonstrate how to simplify complex tag address paths by creating aliases.

1. Click Edit and then select **Alias Map**.

2. Select the  button to create a new alias.



3. Clicking on the desired device name or group name in the tree browse where the tag you want to reference is located. The fully qualified path for the tag item will be shown in the grayed **Mapped to** field. Enter an alias name that will represent this complex tag reference. It is this alias name that now can be used in your client application instead of the fully qualified path to address the tag found in the server.



Here the complex topic and item name of "_ddedata!Channel1.Device1.Machine1.Cell2.ToolDepth" can be replaced by using the alias "Mac1Cell2". If this is applied to the above example, a DDE link in the application can now be entered as follows:

= <DDE service name>|Mac1Cell2!ToolDepth

Important: If Net DDE is enabled (using the **Tools|Options DDE** menu option), the alias map entries will be registered as DDE shares for use by remote applications. The names given to each alias map entry must not conflict with any existing DDE shares already defined on the server PC. For more information, refer to [NetDDE](#) for more).

Operational Note: If while using a DDE client application you receive the error message ""DDE client attempt to add a topic (?) failed. Refer to the alias map under the Edit menu for valid topics.", the topic shown at (?) in the error

message does not exist in your alias map. Edit the alias map to include this topic and restart your DDE client.

See Also: [Alias Map](#) and [Alias Properties](#).

Diagnostics Overview

On occasion, communications problems may occur that require more insight to its cause. It is in these situations that an advanced user can employ the various diagnostics views available to help in determining what the communications problem may be. Two diagnostic tools available in the server are the [OPC Diagnostics](#) and [Channel Diagnostics](#) windows. These views provide both server and driver level diagnostics for advanced users.

OPC Diagnostics

The OPC diagnostics window provides a real-time and historical view of OPC events that occur between any OPC client and the server.

An

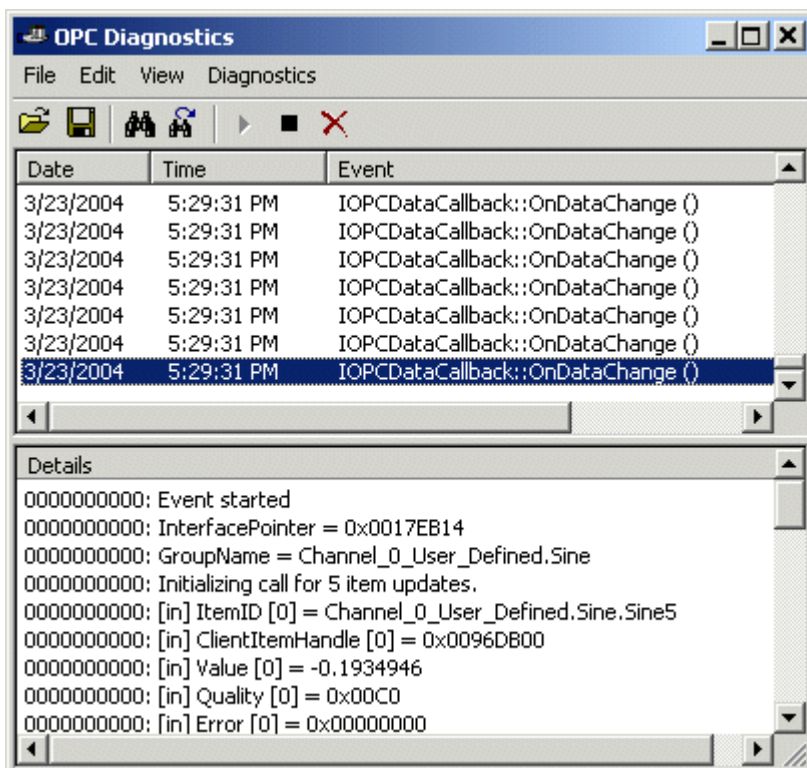
event is a method call that a client makes into the server, or a callback the server makes into a client.

This

window is separate from the main server configuration window, which can be used to hide the window even while diagnostics are being captured. In the event that there is an issue between a client and the server, the diagnostics window can be opened to display the events that have occurred. The operation of this view can be controlled through the menu, toolbar or by right-clicking in the event window.

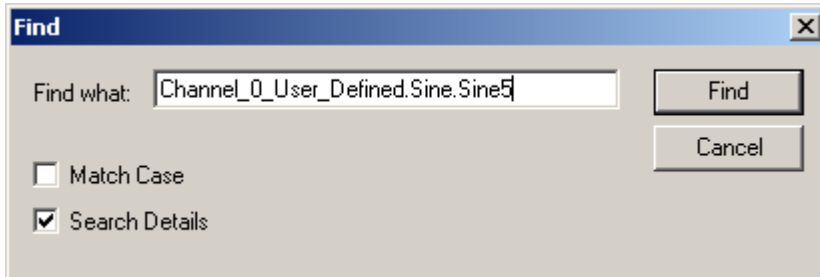
The events and details captured by this utility are specific to the OPC Data Access 1.0, 2.0 and 3.0 Custom Specifications. Please consult these specifications to determine the meaning of the data presented by this window. Copies of these specifications can be found on the OPC Foundation web site: www.opcfoundation.org.

Click on any area of the figure below to learn about the basic functionality of the OPC diagnostics window.



OPC Diagnostic Features

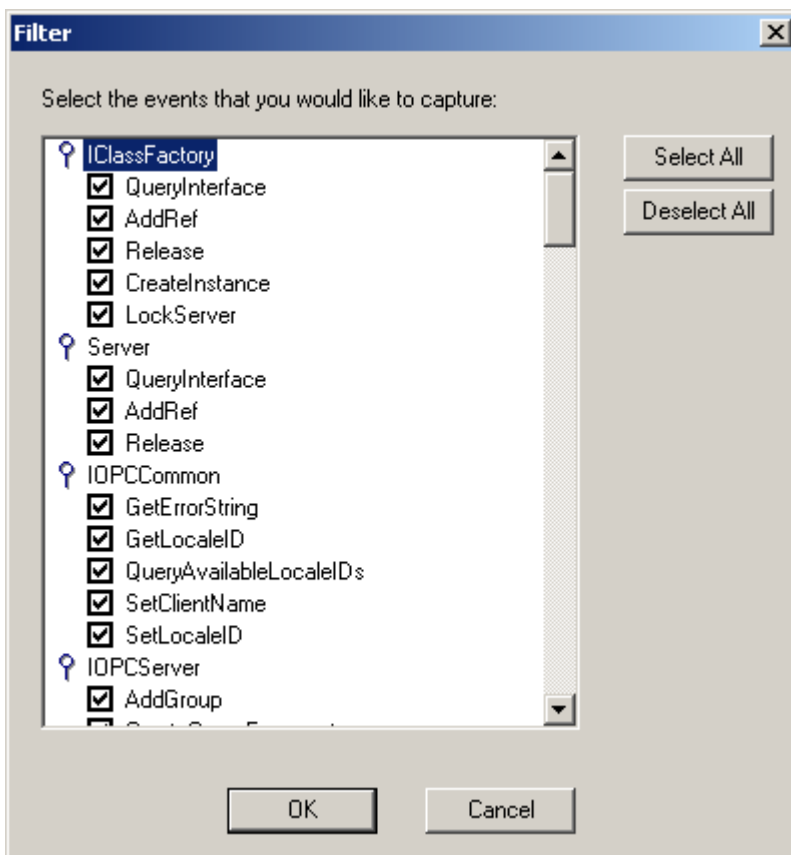
The **Find Dialog** can be used to search for specific text within the diagnostics view. This aids in the debugging of a particular issue by allowing you to search for key information transferred between the client and server (e.g., All actions on a particular Item ID or group name can easily be found using the search functionality).



Users can specify whether the search criteria should be case-sensitive by selecting the **Match Case** checkbox, or whether or not to include details in the search by selecting the **Search Details** checkbox.

When an event or detail with the specified text is found, the line containing the text is highlighted. Press **F3** to perform a **Find Next** operation which will look for the next occurrence of the specified text. When the last occurrence is found a message box indicating this condition will be posted. To change the search criteria at any time press **Ctrl+F** to bring up the **Find Dialog**.

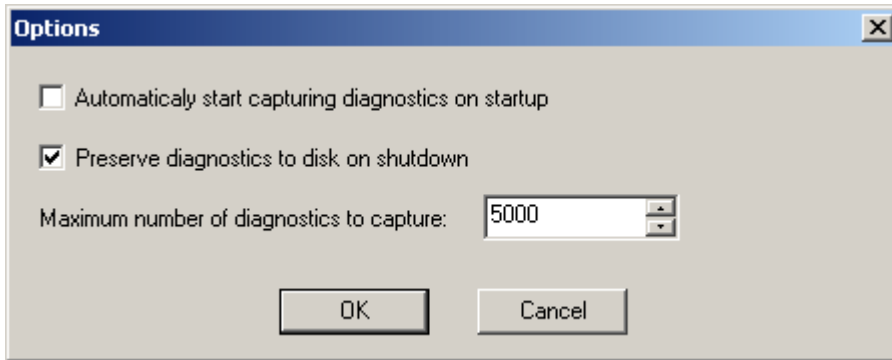
The **Filter Dialog** can be used to select which events you want to capture. This aids in debugging a particular issue by eliminating events that do not pertain to a particular issue (e.g., Most clients will make continuous GetStatus calls into the server to determine that the server is still available. Filtering this event will simplify the diagnostics data to examine).



Each method (e.g., GetErrorString) of every OPC Data Access 1.0, 2.0 and 3.0 interface (e.g., IOPCCommon) supported by the server is available as a filter. Users can select a method by clicking the checkbox to the left of the method name. Additionally, Users can select all methods of an interface by double-clicking the interface name. By default all methods for all interfaces are selected.

Note: Filter changes apply only to new events captured. Previous events that meet the filter criteria will not be removed from the view.

The **Options Dialog** is used to control certain functionality built into the OPC diagnostics feature.



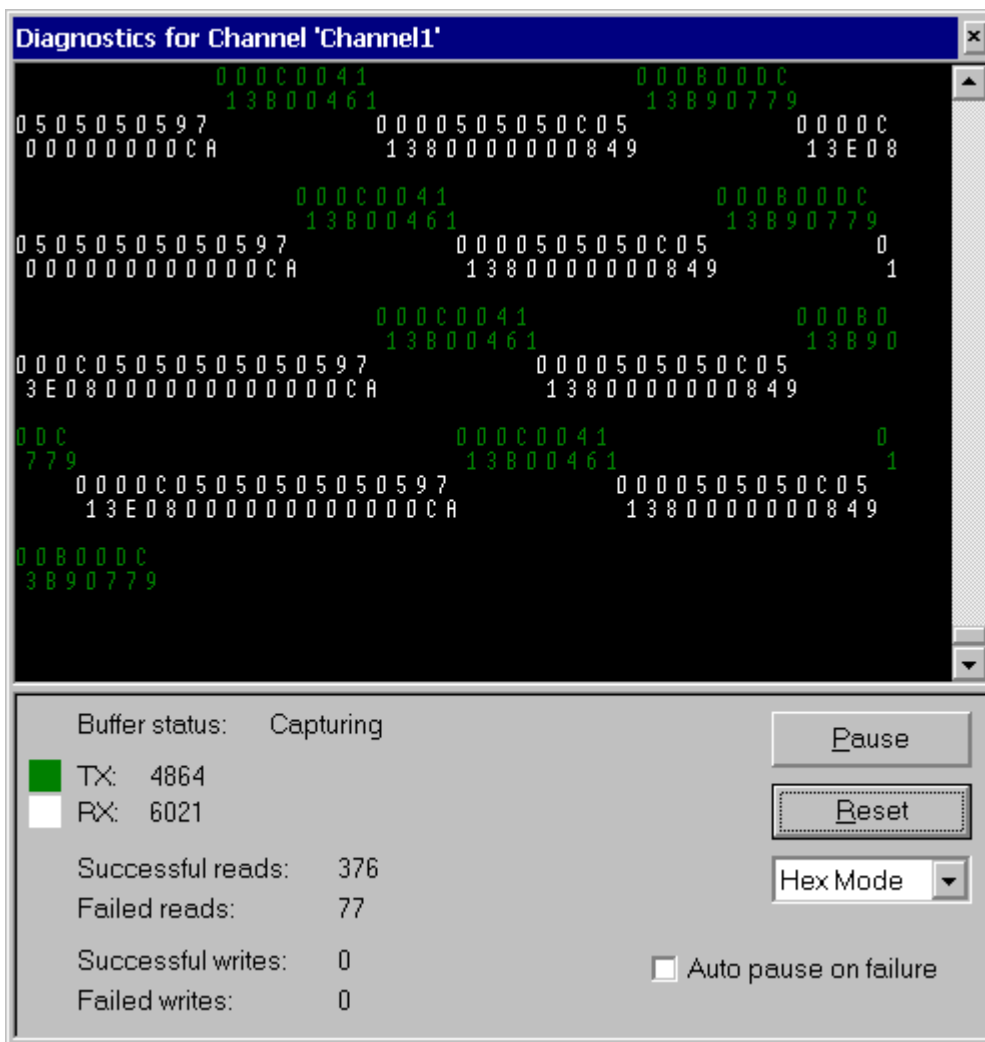
Users can decide whether or not OPC diagnostics information should be captured every time the server is started by selecting the **Automatically start capturing diagnostics on startup** checkbox. Remember that the performance of the server is affected by capturing diagnostic information, as it is an additional layer of processing that occurs between all client/server communications. The OPC diagnostic information is saved to the installation directory under the file name **autosave.opcdiag**. The file is only created on shutdown. This functionality is turned off by default.

Users can also decide whether or not to persist OPC diagnostic information to disk when the server is shutdown by selecting the **Preserve diagnostics to disk on shutdown**. This can be useful if you need to go back in time to determine where a particular issue may have occurred. Users can also create your own backup of diagnostics data at any time through the **File | Save As** menu option. By default this functionality is turned on.

Since diagnostic information has an affect on memory/file storage Users can decide the maximum number of OPC diagnostic events that should be captured at any specific time, by modifying the **Maximum number of diagnostics to capture**. This value only pertains to the number of events and does not include the number of details, as the detail count for a particular event could be potentially very high. The range for this setting is 1000 to 30000. The default value is 5000.

Channel Diagnostics

The channel diagnostics window provides a real-time view of the [diagnostic tags](#) for the selected channel and a real-time view of the actual device protocol. The protocol view window can be sized to suit specific application needs. To modify the operation of the diagnostic window, right-click in the protocol view of the window.



The diagnostic window can be a powerful tool in solving tough communications problems. The window operates in a modeless form that allows it to exist while other dialogs in the server are open. If a tough communications problem is occurring or even if the exact device communications settings are unknown, the diagnostic window can be used to quickly resolve these problems.

Diagnostic Controls

The **Pause** button will capture and freeze the protocol data in the diagnostics window, while the [diagnostic tags](#) will continue to be updated. The **Reset** button will clear only the contents of the protocol view and does not affect the diagnostic tags. The **Mode** selection can be used to display the protocol information in a pure Hex representation (Hex Mode) or in a mixture of ASCII and Hex representation (Mix Mode). Depending on the nature of the target device, the Mode selection may need to be changed to provide the best view of the protocol. The **Auto pause on failure** function causes the capture buffer of the protocol view to stop whenever either a read or write error occurs. By default, this selection is off and the protocol capture will run in continuous mode. The Auto pause feature is a great way to capture intermittent communications issues.

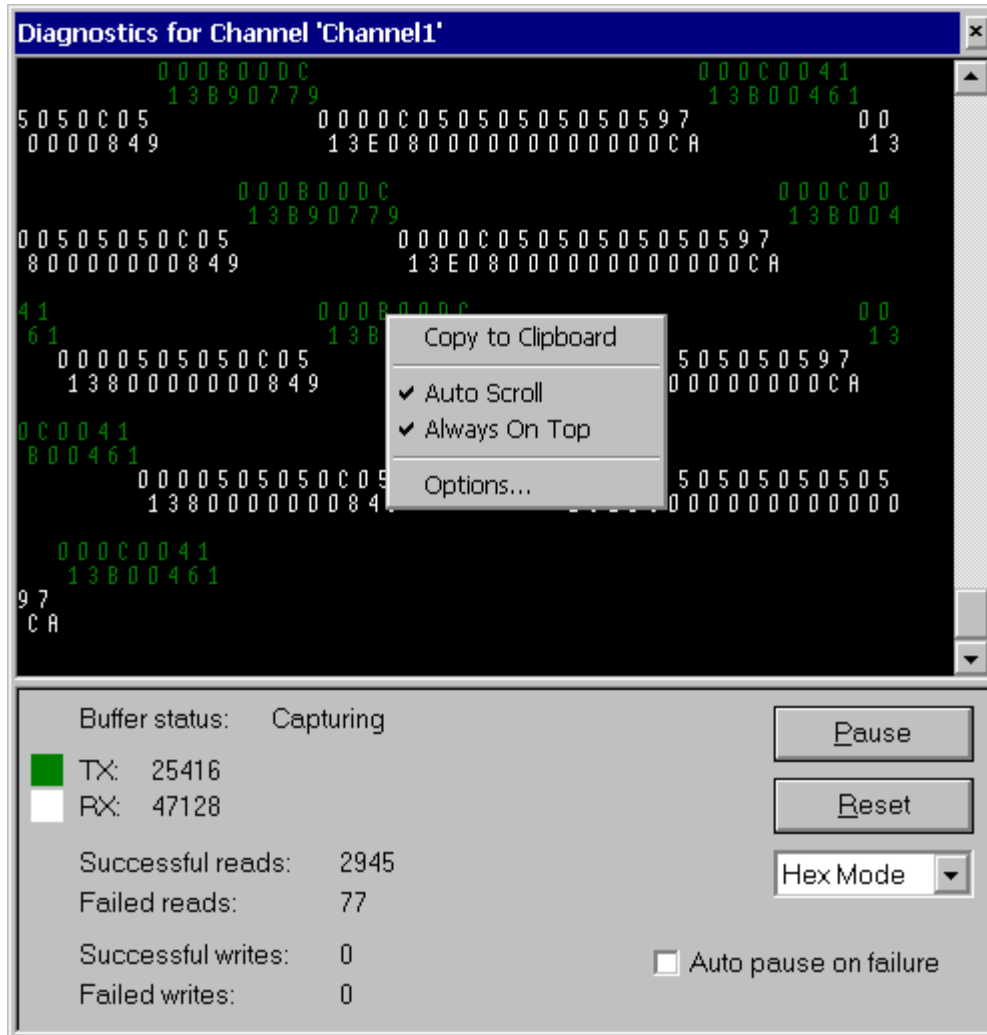
The first step to using the diagnostic window is to ensure that diagnostic functions have been enabled on the [channel properties](#) page. Once diagnostics have been enabled, the diagnostic window can be displayed by selecting it from a context menu on the channel or by the **View|Diagnostics** menu option. Once the diagnostic menu is displayed it will begin capturing the real-time protocol data. If communications are occurring properly, there will be a stream of communications messages between the server and the device. You can tell when this is occurring due to the two-color scheme of the protocol window and by current counts of the [diagnostic tags](#) displayed near the bottom of the diagnostic window.

If, for some reason, communications do not appear to be occurring normally, the parameters for the channel can be

easily accessed using the dialogs to configure these settings. The diagnostic window will remain displayed even when you display the channel properties for settings like the [communications parameters](#). This allows you to interactively change communications parameters while monitoring the effect in the protocol view. Remember to display the diagnostic window before accessing any parameter dialogs.

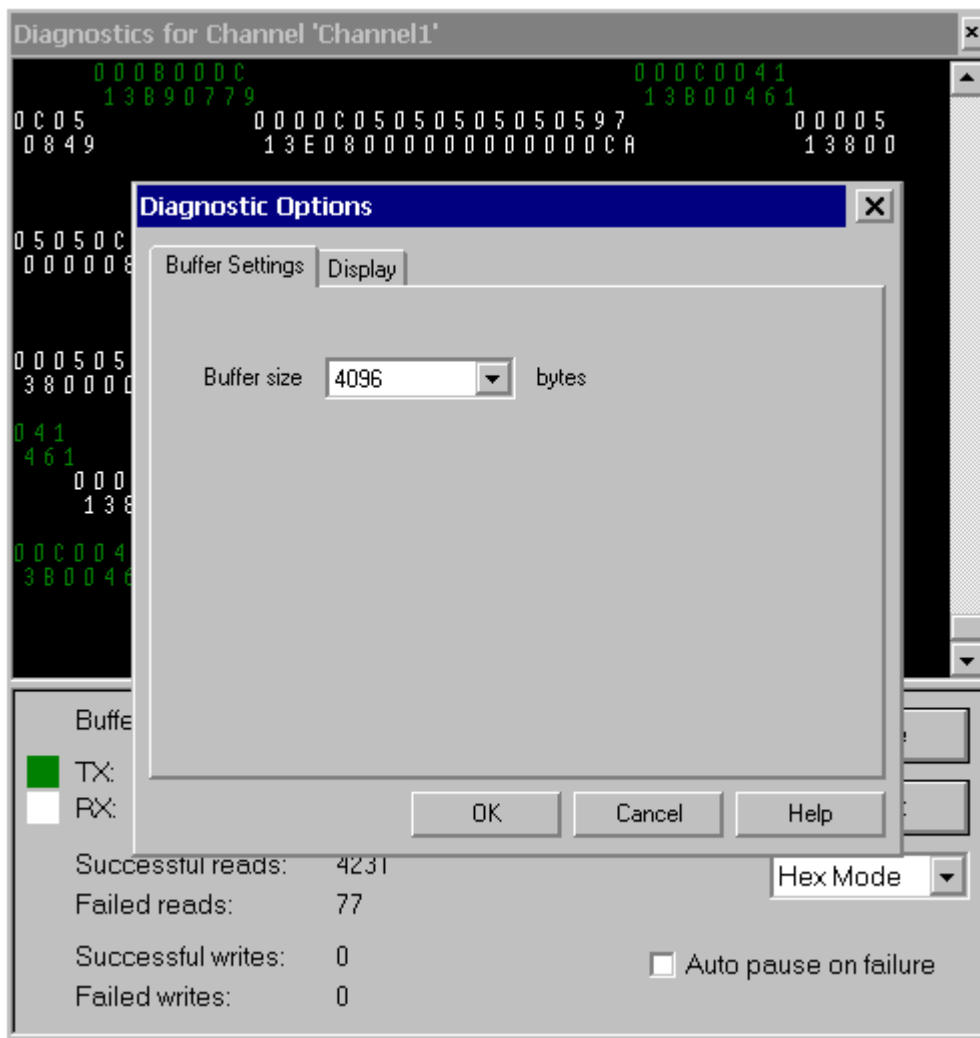
Once the diagnostic window is displayed, various communication and device settings (in your channel and device) can be modified to see the effect on the communications stream. If you are certain that you have your communication parameters set correctly and you are still having a communications issue, you can use the second benefit of the diagnostic window.

By right-clicking in the diagnostic window, the context menu will appear as shown below.

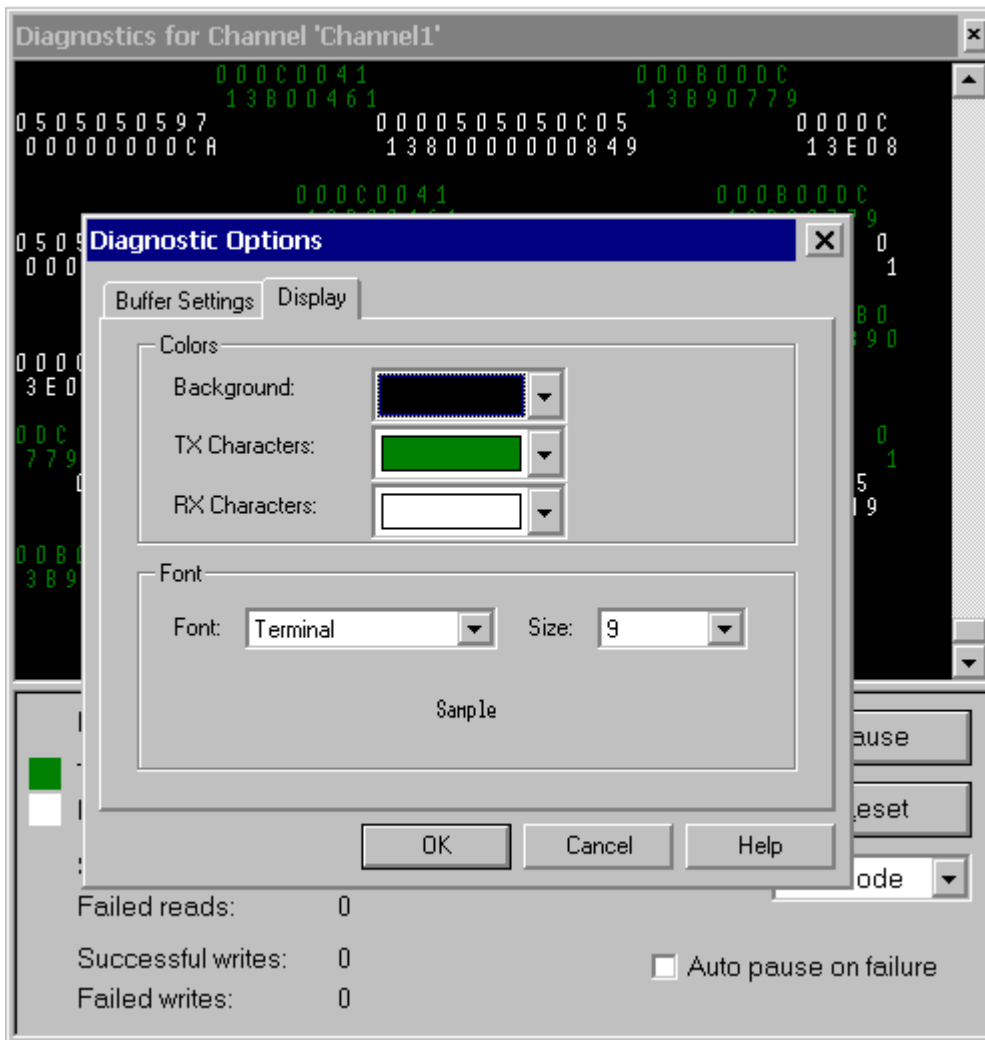


The selections available from this context menu can be used to tailor the operation of the diagnostic window and copy the contents of the protocol view to the Clipboard. The **Copy to Clipboard** selection is the most important feature that will help aid technical support in solving many of your communications issues, should they occur. Once the Copy to Clipboard selection is invoked, the current contents of the protocol capture buffer is formatted as text for easy "cut and paste" into either an email or fax message that can be sent to technical support for analysis. This can be very helpful when trying to diagnose a communications issue. The **Auto Scroll** selection simply ensures that the protocol view will automatically scroll to the next line as new protocol data is received. The **Always on Top** selection can be used to force the diagnostics window to remain on the top of all other application windows. Always on top is the default condition.

The **Options...** selection can be used to select a buffer size for the protocol capture buffer, set the font and size used by the protocol view, and also assign unique colors to both the transmit and receive data streams.



The **Buffer Size** selection can be used to select a capture buffer of 1024, 2048, 4096, 8192, 16384, 32768, 65536 bytes. Depending on the speed of the device, a larger buffer size may be needed to effectively capture protocol related issues.



The **Display** settings can be used to tailor the appearance of the protocol view. The **Background**, **TX Characters**, and **RX Characters** color selections allow you to select a color from a palette of 16 available colors with each selection required to be unique. The font used by the protocol view can also be selected from this dialog as well as the size of that font. When selecting a font to be used by the protocol view, keep in mind that the speed of update can be affected if a font size is set to large. Additionally, larger font sizes will prevent you from being able to see a usable amount of the protocol.

Designing a Project

There are a couple steps required to get a project up and running. To build a project, use the server's user interface to select the communication driver, configure that driver's parameters and add user-defined tags to the project. To begin, review the OPC server system requirements. For more information, refer to [System Requirements](#).

The following steps will help guide users through the process of building a project. Throughout this example, a number of additional help links will be presented that will offer relevant pieces of information. With that in mind, use the "Back" button on the help system's toolbar to return to the example.

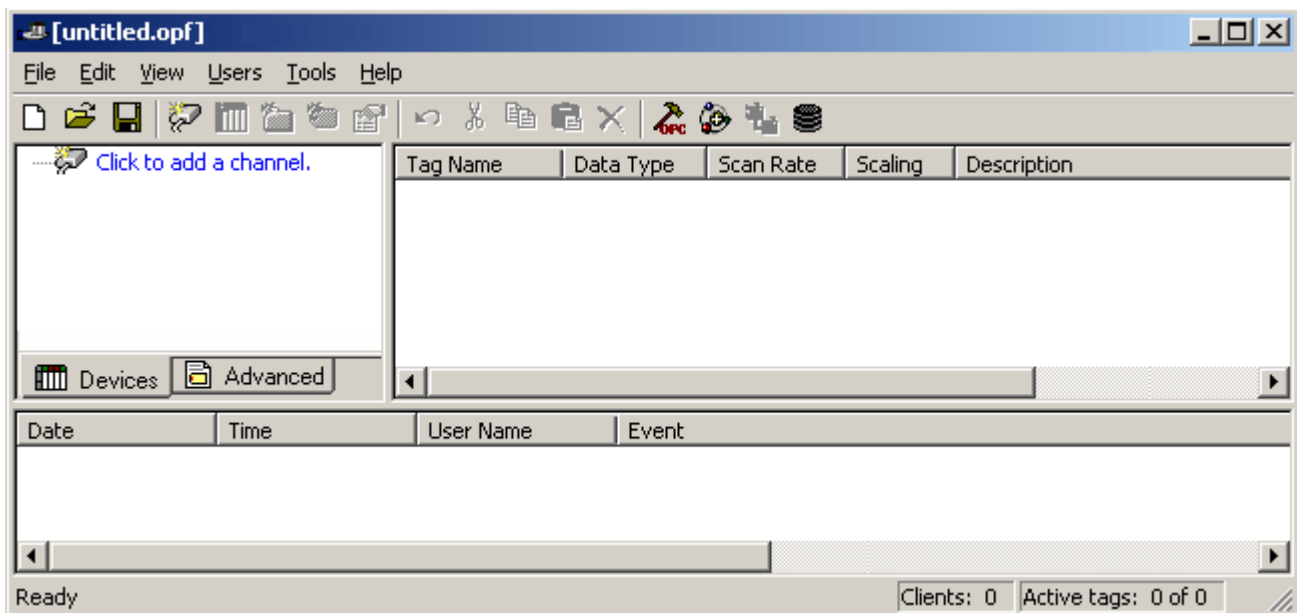
1. [Running the Server](#)
2. [Starting a new Project](#)
3. [Adding and Configuring a Channel](#)
4. [Adding and Configuring a Device](#)
5. [Adding User Defined Tags to the Project](#)
6. [Adding Tag Scaling](#)
7. [Saving and Testing the Project](#)

Note: The examples shown here will use the Simulator driver supplied with the server. The Simulator driver is a memory-based driver that provides both static and changing data for demonstration purposes. The Simulator driver does not have the broad range of configuration options you would find in our normal communication drivers. With that in mind, some examples may require additional screen shots to demonstrate specific product features that are present when using one of the normal communication drivers. In these cases, screen shots will be provided for future reference when configuring a project with one of our standard communication drivers.

Designing a Project (Running the Server)

The server, like any OPC server, can be started a number of ways. One of the benefits of OPC technology is that the OPC client can automatically invoke the server when it attempts to connect and collect data. In order for this automatic mode of operation to occur, a project must first be created and configured. Then, the server will automatically select the most recently used project when it is invoked by the OPC client.

Initially however, users must manually invoke the server by either double-clicking the desktop icon or by selecting the server from the Windows Start menu. Depending on any changes that have been made to server's appearance, the interface should appear as shown below. For more information on the various user interface elements, refer to [Basic Server Components](#).



Designing a Project (Starting a New Project)

The server must be configured to determine the content of what the server will provide while it is operating. For a server project, that means defining [channels](#), [devices](#), optional [tag groups](#) and [tags](#). These factors exist in the context of a project file. As with many applications, a number of project files can be defined and then save and loaded as needed.

While most of the configuration done within the server is contained within a project file, there are also a number of configuration options that are global to the server and are applied to all projects. These global options are configured by using the dialog invoked by clicking **Tools|Options**. The global options include [General](#) options, [Event Logger](#) options, [OPC](#) options and [Compliance](#) options. All global options are stored in a windows INI file called "**Servermain.INI**," which is located in the server's root directory. While it is normal practice to store global options like those listed here in the Windows registry, INI files can be used to easily move the global settings from one machine to another.

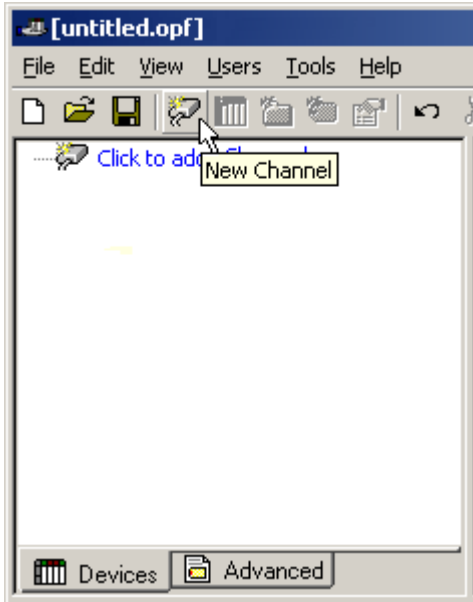
See Also: [Adding a Channel](#)

Designing a Project (Adding and Configuring a Channel)

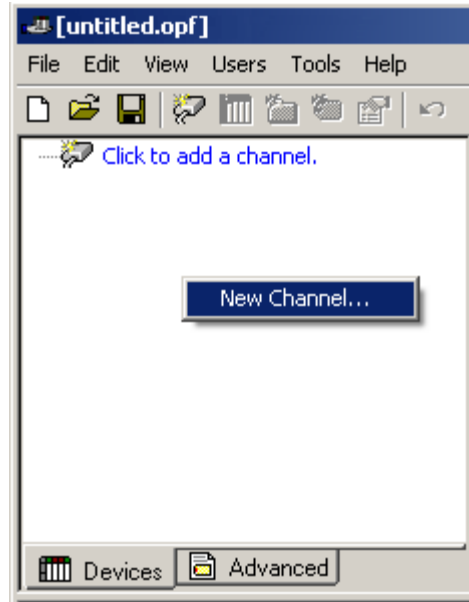
When starting a new project, the first step is to determine which communication driver(s) the application requires. A

communication driver in the server is referred to as a **channel**. The number of channels within a single project that can be defined depends on the the driver or drivers currently installed.

To add a new channel to the project, click **Edit | New Channel**. Alternatively, use the Toolbar Add Channel or context menu. The following figures demonstrate two methods to add new channels to the project.



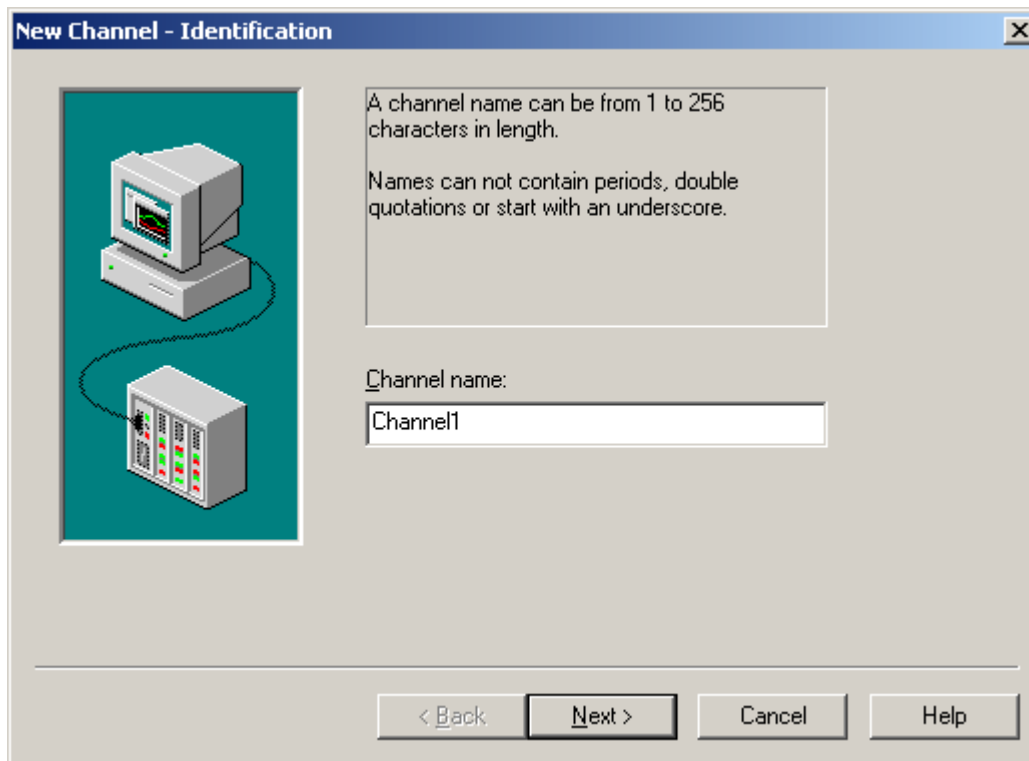
New Channel by Toolbar | Add Channel



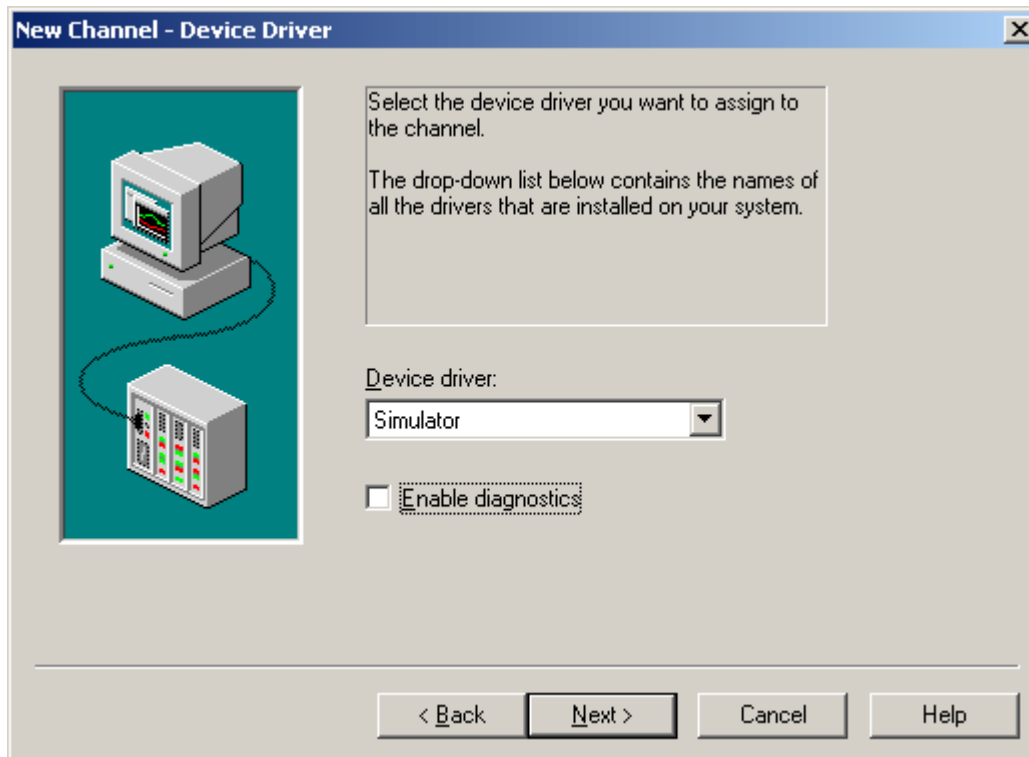
New Channel by Context Menu

Clicking the new channel invoked the **Channel Wizard**, which is used to name the channel and select a communications driver. For this example, the Simulator driver is used.

For simplicity, leave the channel name and type it as "Channel1," as shown below.



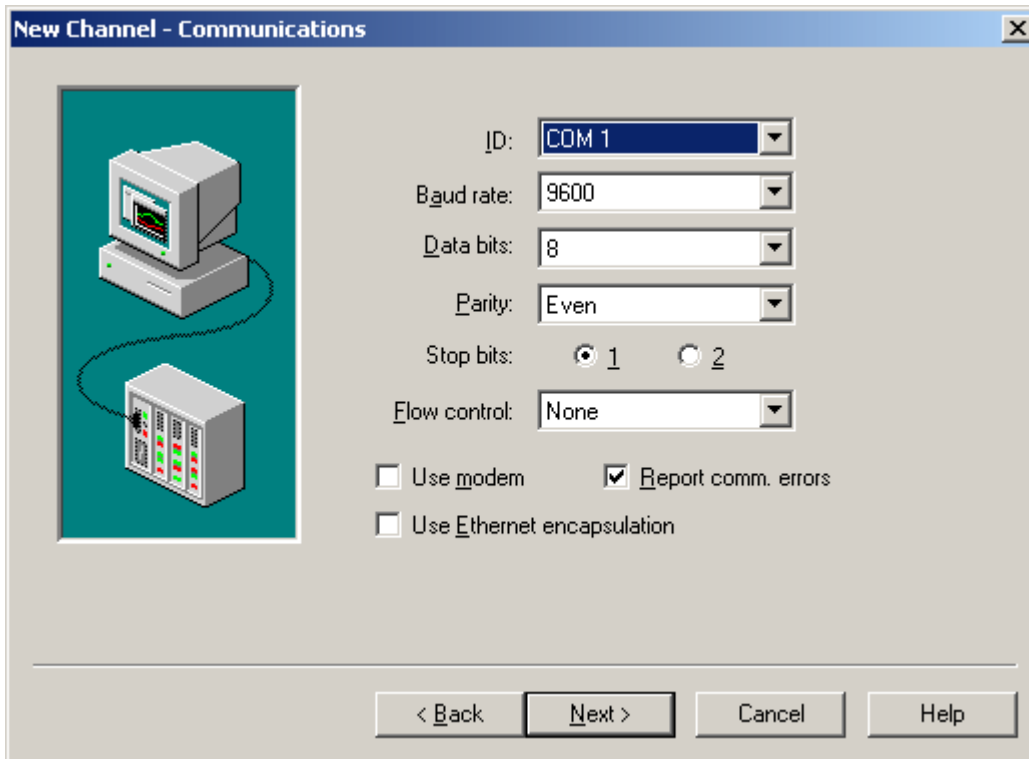
Click the next button to proceed to the next dialog, which is used to select the [communications driver](#) that will be applied to this channel. As in this example, choose the Simulator driver from the driver selection drop-down menu.



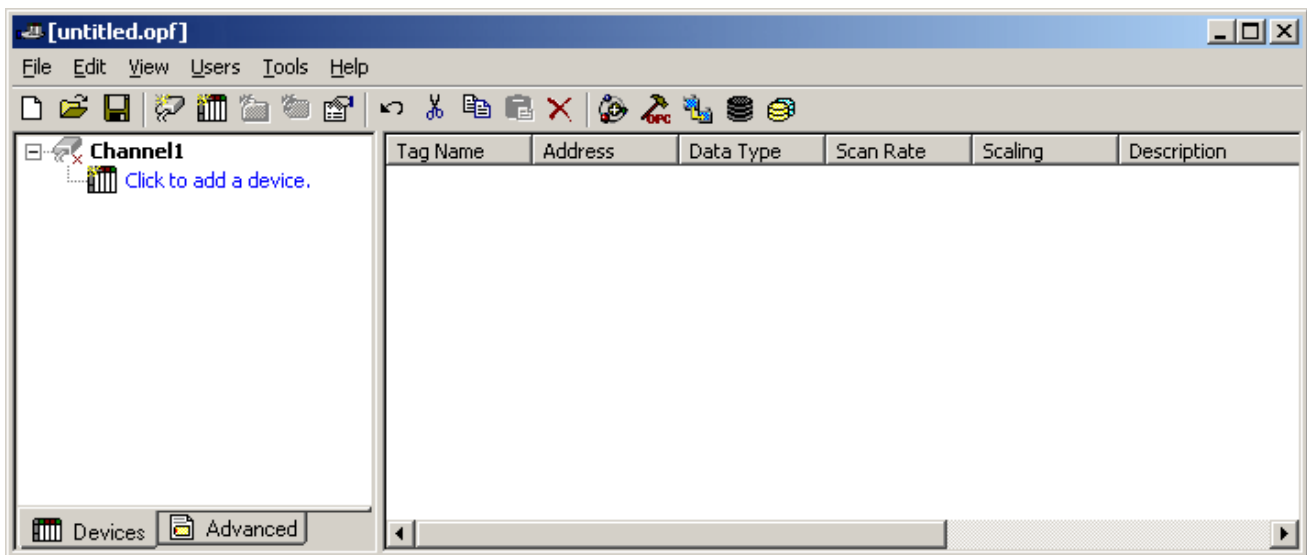
For the Simulator driver the **Next** button will simply display a [channel summary](#) page. To complete the addition of the channel, click **Next** to display the channel summary. After reviewing the information to make sure it is correct, click

Finish.

In normal applications there will be additional dialog pages that allow the configuration of parameters such as communications port, baud rate and parity. While not used by the Simulator driver, the following figure shows the [communications dialog](#) that is common to all communications drivers that use the serial port.



With the Simulator channel now added to the server, the server will appear as shown below.



Its important to note here that the channel is shown using the channel name you gave it but is also has a small red "x" below the channel icon. This red "x" denotes that the channel does not contain a valid configuration. The channel is not valid because a [device](#) has not been added to the channel. Go to [Adding a Device](#).

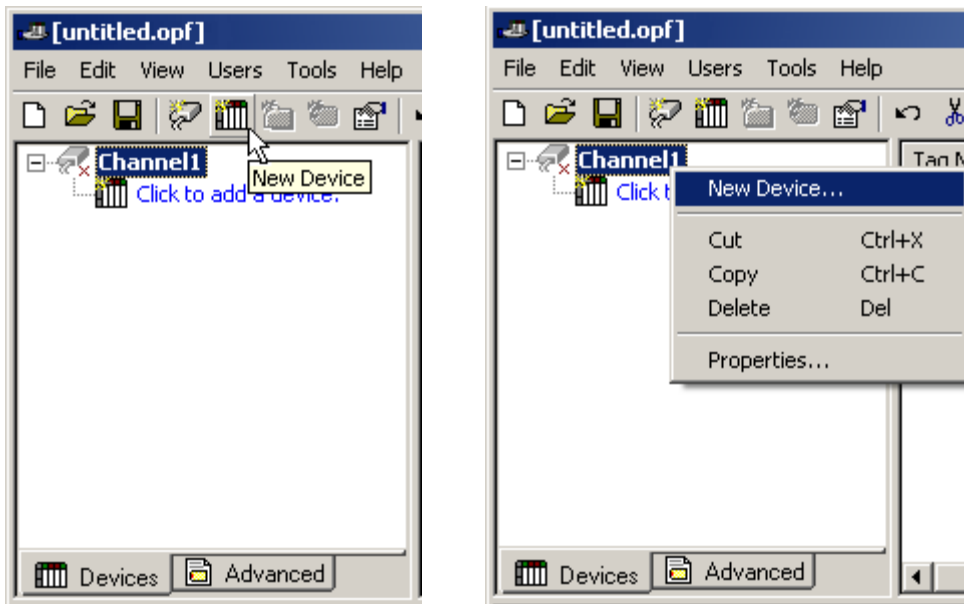
Note: The server supports the use of multiple channels. As channels are added to the project, users can specify either

the same communications driver or different communications drivers. Most communication drivers offered support operation on up to 16 communications ports or Ethernet network connections simultaneously. By defining multiple channels, users can improve the overall performance of the application. In the case of either a serial driver or Ethernet driver using multiple channels can be used to spread large communications loads across the multiple channels. A good example of this would be a serial driver that is being used to communicate with eight devices on the serial line. Normally the communications driver used in this application would be responsible for gathering data from all eight devices in a round robin fashion. If this same application is reconfigured to use multiple channels assigned to multiple communications ports, the device load can be divided across the channels. The end result is a reduced workload on each channel and dramatic improvements in the responsiveness of the application. The need to use multiple channels is dependent solely on the needs of the application. In either case there is no additional cost involved to use a licensed driver on multiple communications or Ethernet ports.

Designing a Project (Adding and Configuring a Device)

Once a channel has been defined in a project, a [device](#) must be added to the channel. In most cases a device refers to the identification of a physical node or station on a communications link. A device can also be viewed solely as a means of framing the definition of a connection to a specific point of interest in the application. In this respect a device would still be used to as a term when describing the connection to a database object.

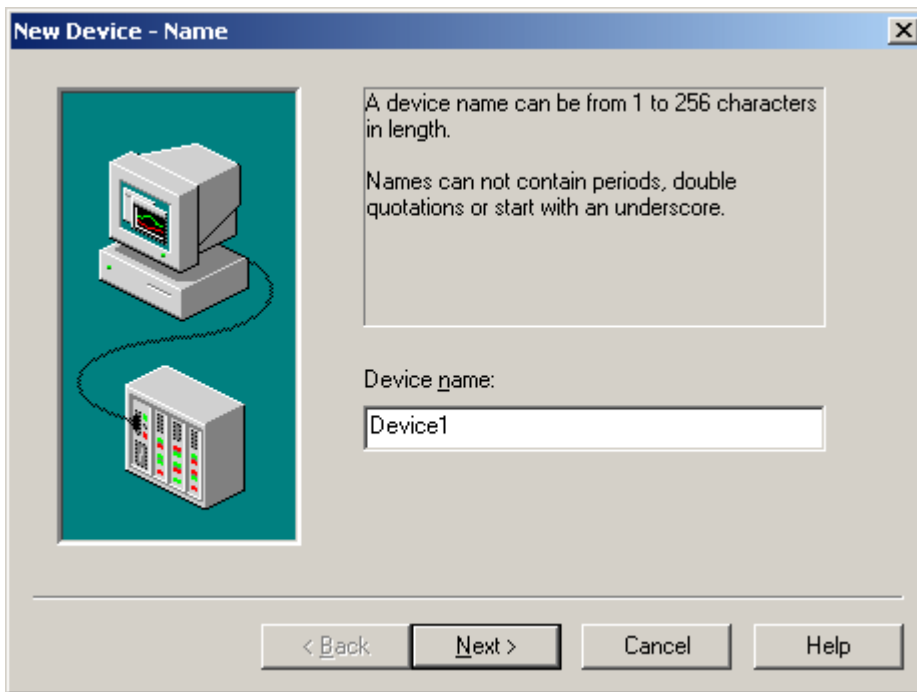
For the purpose of this example, a device refers to a specific device on a network. In the case of the Simulator driver, it supports multiple device nodes allowing you to simulate networked devices. To add a device to a channel, select the desired channel is selected and then click **Edit | Add Device**. Alternatively, use the Toolbar Add Device or the context menu. The following figures demonstrate two methods to add new devices to the project.



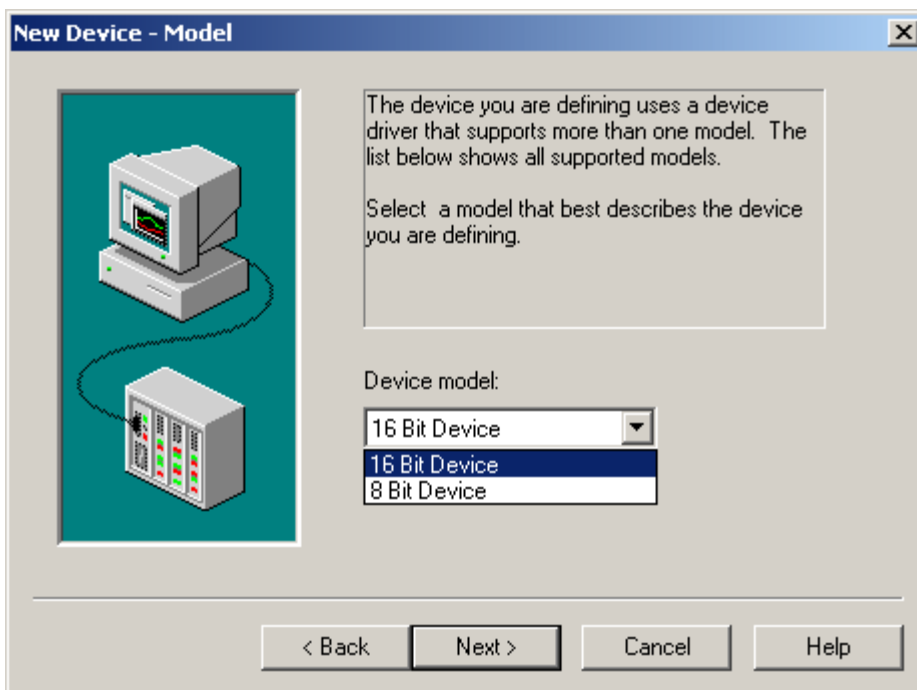
New Device by Toolbar | New Device New Device by Context Menu

Clicking the new device button will invoke the [Device Wizard](#). As shown below, the Device Wizard is used to name the device and set node ID of the device. Depending on the driver being used in the application, the number of dialog pages users encounter when configuring a device will vary. In the case of the Simulator, it is used to select either a 16 bit or 8 bit register size for the device it is simulating.

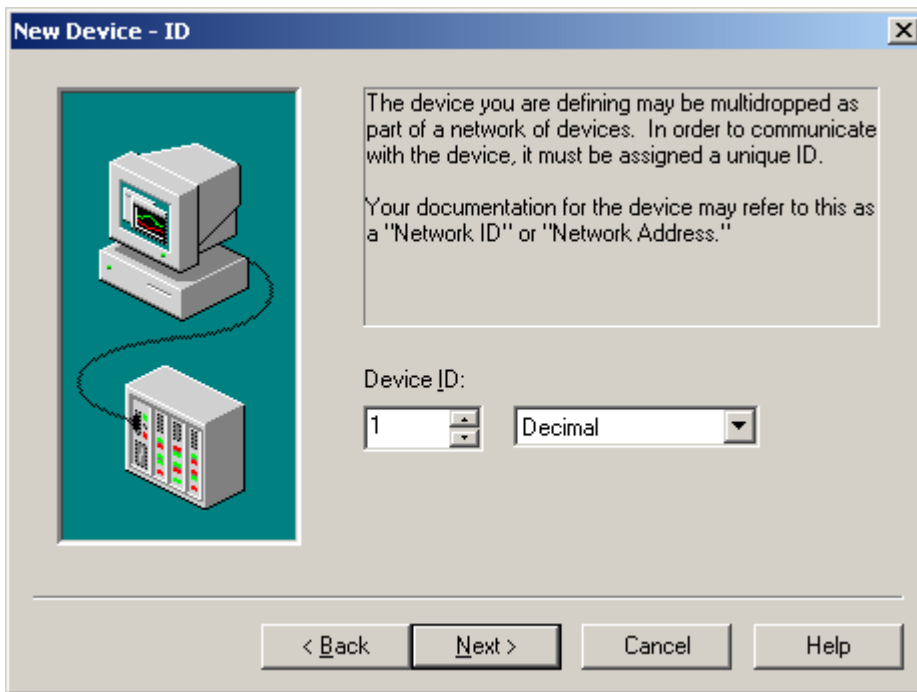
For simplicity, leave the device name as **Device1** as it is shown below.



To proceed to the next configuration wizard, click Next. The following dialog is used to select the register size for this device. In this example, choose the 16-bit device size from the drop down.

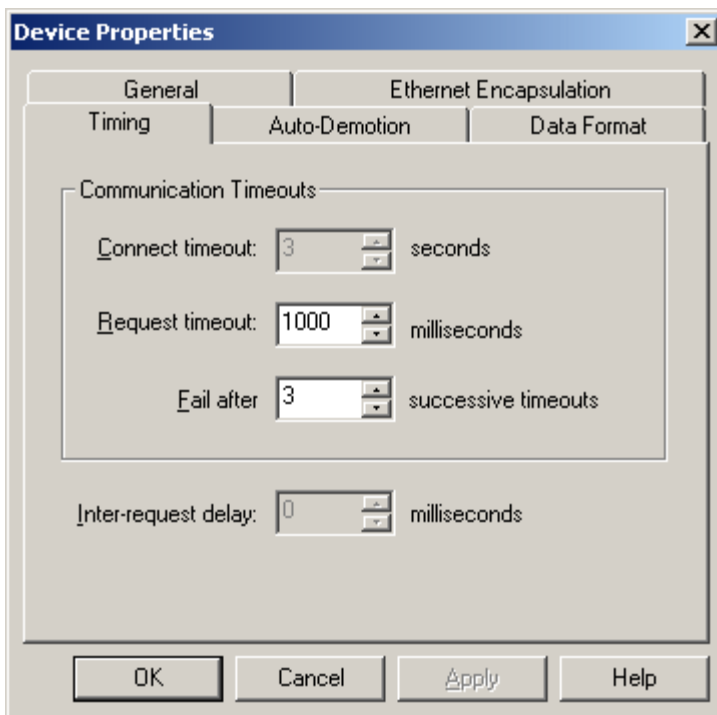


After clicking the next button, users will be presented with the [Device ID](#) dialog as shown below. The Device ID is used to enter the unique identifier that is required by the actual communications protocol. In the case of the Simulator driver, the Device ID is a numeric value. The format and style of the Device ID entered here is a factor of the communications driver being used.

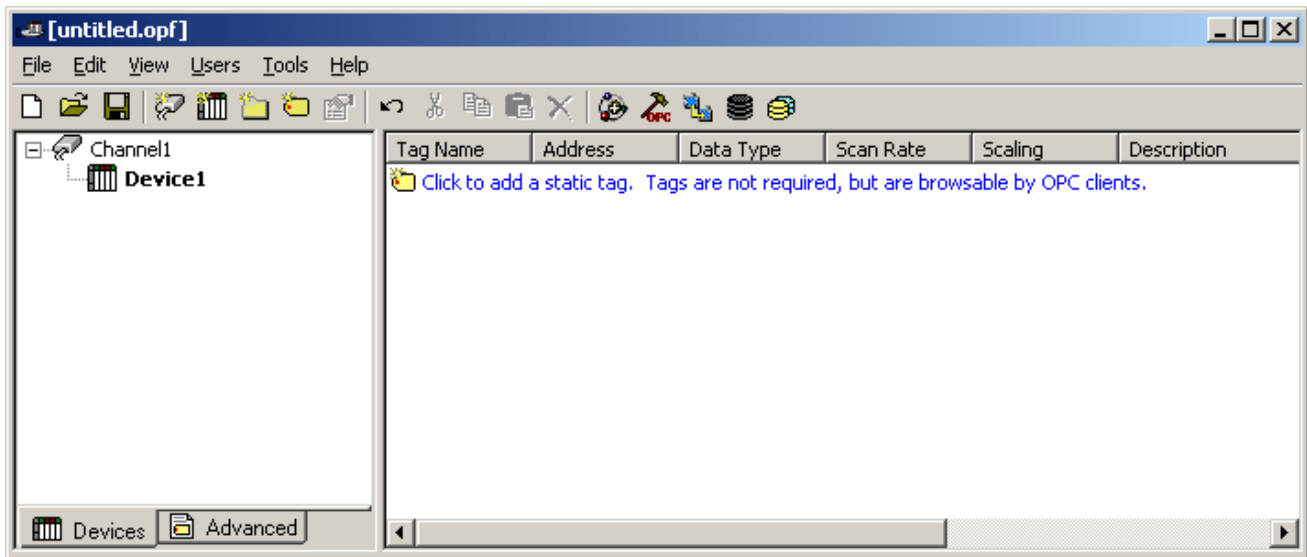


For the Simulator driver the **Next** button will simply display a [device summary](#) page. To complete the addition of the channel, click **Next** to display the channel summary. After reviewing the information to make sure it is correct, click **Finish**.

In normal applications, there will be additional dialog pages that allow the configuration of parameters such as device timeout, and retry count. While not used by the Simulator driver, the following figure shows the [timeout dialog](#) that is common to all communications drivers.



With a device now added to the Simulator channel, the server will appear as shown below.



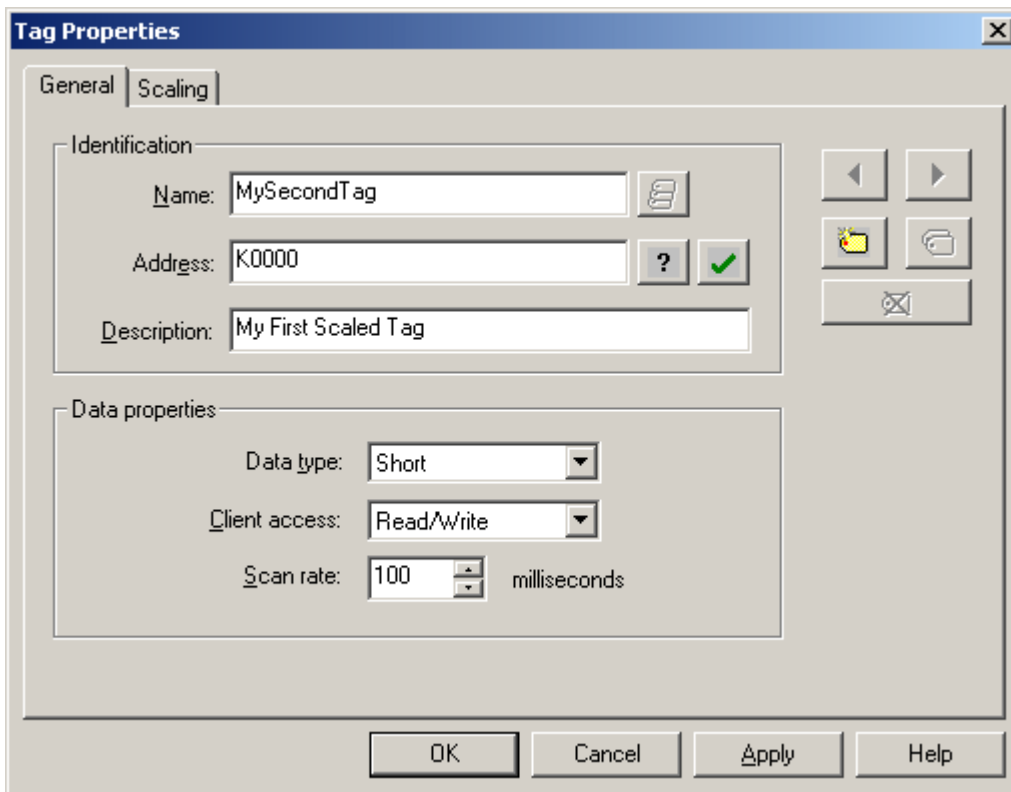
With a channel and device added to the project the server is ready to start providing data to OPC clients. With the server's online full time mode of operation, the server can start providing OPC data immediately. At this point, however, the configuration developed thus far could be lost since the project has not yet been saved.

Designing a Project (Adding Tags to the Project)

User Defined Tags

There are two ways to get data from a device to the client application using the server. The first method and most common method requires that users define a set of tags in the server project and then use the name assigned to each tag as the item of each link between the client and the server. The primary benefit to this method is that all user-defined tags are available for browsing within OPC clients. Additionally, user defined tags also support [scaling](#). For more information, refer to [Adding Tag Scaling](#).

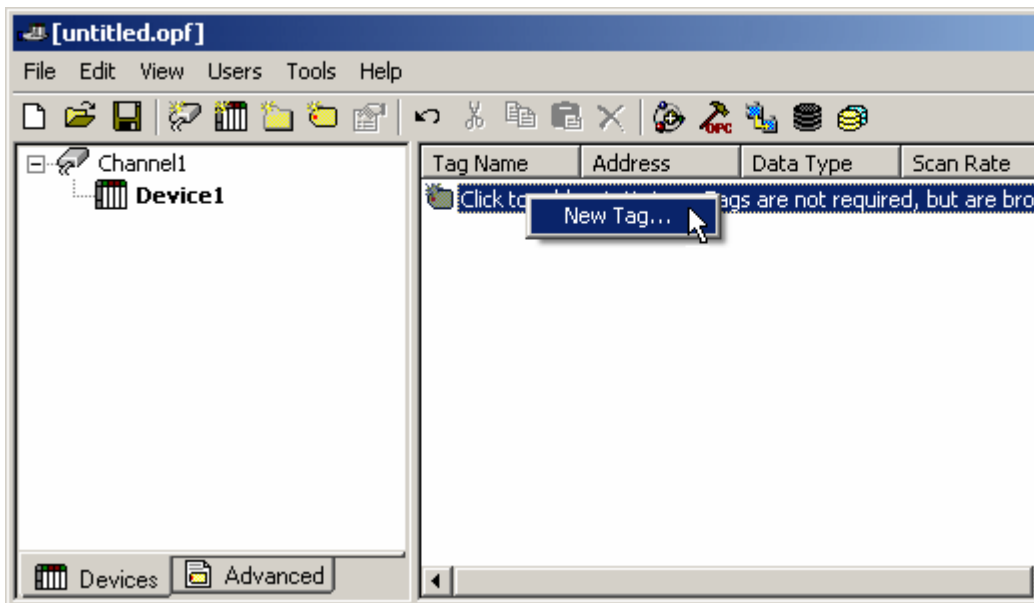
To add a tag to the project, first select a device name from the Channel/Device tree view within the server. Using the project created as part of this example that view should appear as shown below.



In the view shown above, **Device1** is currently selected. Next, use the Edit | Add Tag, Toolbar Add Tag, or the context menu to add new tags to the project.



Add Tag by toolbar Add Tag



Add Tag by context menu (right-click)

After clicking the new tag button, the **Tag Properties** dialog will be invoked. As shown below, the Tag Properties dialog is used to name the tag, specify a device specific address, select a data type and set the access method of the tag.

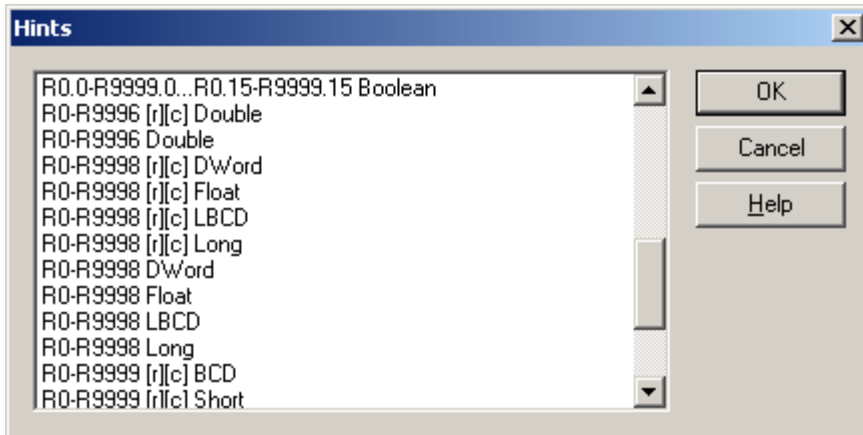
The screenshot shows the 'Tag Properties' dialog box with the 'General' tab selected. The 'Identification' section contains three text boxes: 'Name' (empty), 'Address' (empty), and 'Description' (empty). To the right of these boxes are several icons: a left arrow, a right arrow, a yellow lightbulb, a document icon, and a close icon. The 'Data properties' section contains three controls: 'Data type' (dropdown menu with 'Default' selected), 'Client access' (dropdown menu with 'Read/Write' selected), and 'Scan rate' (spin box with '100' and 'milliseconds' label). At the bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

Complete details for the Tag Properties dialog can be found on the [Tag Properties](#) page. Edit the controls to match the following content.

The screenshot shows the 'Tag Properties' dialog box with the 'General' tab selected. The 'Identification' section contains three text boxes: 'Name' (filled with 'MyFirstTag'), 'Address' (filled with 'R0000'), and 'Description' (filled with 'My First Simulator Tag'). To the right of these boxes are several icons: a left arrow, a right arrow, a yellow lightbulb, a document icon, and a close icon. The 'Data properties' section contains three controls: 'Data type' (dropdown menu with 'Word' selected), 'Client access' (dropdown menu with 'Read/Write' selected), and 'Scan rate' (spin box with '100' and 'milliseconds' label). At the bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

As shown in the tag dialog above the tag name is **MyFirstTag**, the address is **R000**, the description which is optional is **My First Simulator Tag**, the data type is **Word**, client access is **Read/Write** and scan rate is **100** milliseconds (which does not apply for OPC tags). There are three fields here whose content is specifically tied to the communications driver being used, the address field, data type field, and client access field.

The communications driver that was chosen when configuring the [channel](#) of this [device](#) is governed by the content of these fields. In the case of the Simulator driver, **R000** is a valid address. The address **R000** supports a data type of **Word** and client access rights of **Read/Write**. For help on addressing and data types, click the question mark icon in the Tag Properties dialog. This will invoke the hints page for the currently selected driver. For the Simulator, the hints dialog appears as follows.



The hints dialog lists all of the available addresses and their data types for a given communications driver. Once users are familiar with a particular communications driver, the hints may be all that's needed to remind them of an address. If, however, users need additional detail, the help button shown on the Hints dialog will invoke the driver specific help and take users directly to the page that covers data addressing for the device that is currently selected. Between the hints and the driver specific help data, finding the correct address and data type is a painless process.

If the tag information has been entered as shown above, commit this tag to the server by pressing the **Apply** button on the Tag Properties dialog. The tag should then appear in the [tag view area](#) of the server. If all users needed to do was add a single tag, they would be done at this point. For this example, however, a second tag must be added for use in the next step where scaling is applied to the tag.

If **Apply** has already been pressed, select the New icon on the Tag Properties dialog. This will cause all fields of the dialog to clear and return to their default state. Enter the following data:

Tag name: MySecondTag
Address: K000
Description: My First Scaled Tag
Data Type: Short
Client Access: Read/Write
Scan rate: (Doesn't Apply)

Once users have entered this information into the Tag Properties dialog, press the **Apply** button to commit this second tag. Once the tag is committed, refer to [Adding Tag Scaling](#) for more information.

When entering tag information, users can and will be presented with the occasional error message. The error messages presented will either be generated by the server or by the selected driver. The server will generate error messages when users attempt to add a tag using the same tag name as an existing tag. The communications driver will generate errors for three possible reasons. The first case would be any errors in the format or content of the address field that has been entered. This includes errors in the address range of a particular device specific data item that may have been entered. The second error generated may occur when the data type users have selected is not available for the address that has been entered. The third possible error may occur if the client access level users have chosen is not available for the address that has been entered. In each of these cases the error popup will be specific in describing the error. The error popup will also give users the option of directly invoking the driver specific help to aid users in remedying the problem.

The second method of defining tags is called [Dynamic Tag](#) addressing. Dynamic tags allow users to define tags solely

in the client application. Instead of creating a tag item in the client that addresses another tag item that has been created in the server, users need only to create a tag item in the client that directly accesses the device address.

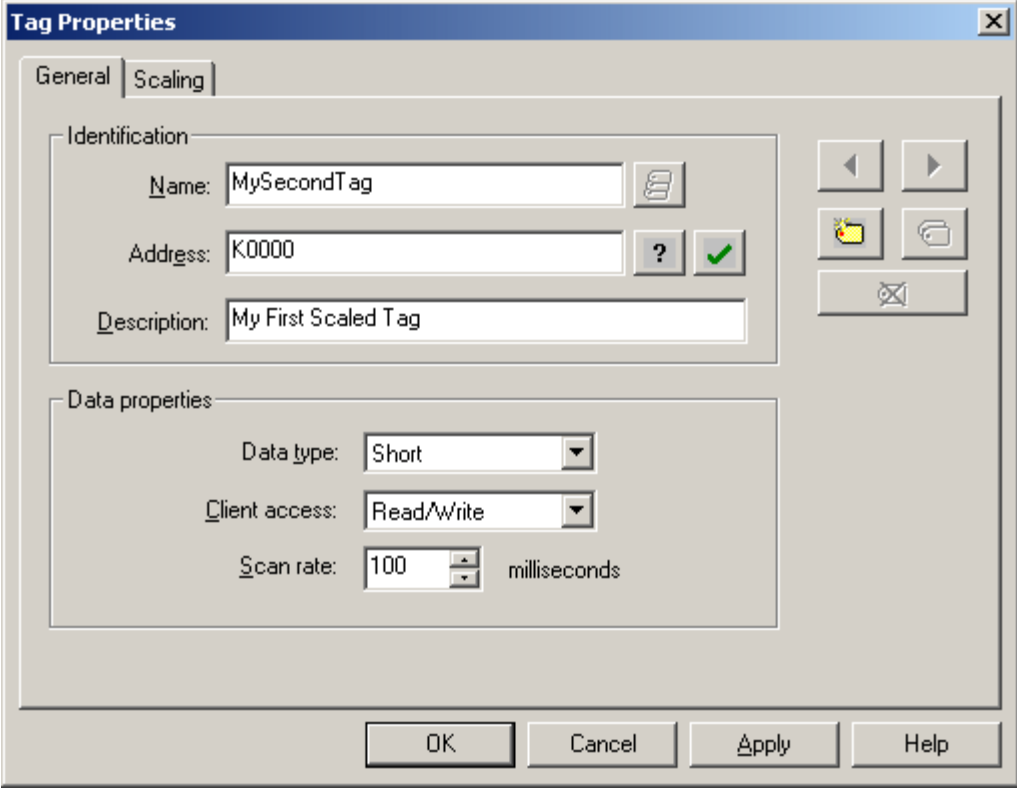
Note: The server creates a special Boolean tag for every device in a project that can be used by a client to determine whether that device is functioning properly. To use this tag users would specify the item in the link as Error. The value of this tag is zero if the device is communicating properly otherwise it is one.

Note: If users use a device address as the item of an link such that the address matches the name of a user-defined tag in the server, the link will reference the address pointed to by the user defined tag.

Designing a Project (Adding Tag Scaling)

The server now supports tag scaling. Scaling allows raw data from the device to be scaled to a more appropriate range for the application. There are two types of scaling; Linear, and Square Root. For more information, refer to [tag scaling](#).

When defining a new tag in the server, users can apply scaling to the tag. Using the second tag developed in this example the Tag Properties dialog should contain the following content:



The screenshot shows the 'Tag Properties' dialog box with the 'Scaling' tab selected. The dialog is divided into two main sections: 'Identification' and 'Data properties'. In the 'Identification' section, the 'Name' field contains 'MySecondTag', the 'Address' field contains 'K0000', and the 'Description' field contains 'My First Scaled Tag'. In the 'Data properties' section, the 'Data type' is set to 'Short', 'Client access' is set to 'Read/Write', and 'Scan rate' is set to '100' milliseconds. The dialog also features a set of navigation buttons on the right and standard 'OK', 'Cancel', 'Apply', and 'Help' buttons at the bottom.

To add scaling to this tag, first click on the scaling tab and then edit the content of the dialog to match the following.

The screenshot shows the 'Tag Properties' dialog box with the 'Scaling' tab selected. At the top, there are three radio buttons: 'None', 'Linear' (which is selected), and 'Square root'. Below this, there are two main sections: 'Raw Value Range' and 'Scaled Value Range'.
 In the 'Raw Value Range' section, the 'Data type' is set to 'Word'. The 'High' field contains the value '300' and the 'Low' field contains '-300'.
 In the 'Scaled Value Range' section, the 'Data type' is set to 'Double'. The 'High' field contains '400000' and the 'Low' field contains '-10000'. Both the 'High' and 'Low' fields have a checked 'Clamp' checkbox. The 'Units' field contains the text 'Position'. There is an unchecked checkbox for 'Negate scaled value'.
 At the bottom of the dialog, there are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

There are two types of scaling in the server. For this example "linear" scaling has been selected. The raw value range can be used to specify what you expect to receive as a data range from the actual device. The scaled data type can be used to specify how the resulting scaled value is presented to the OPC client application. The scaled value range can be used to specify the desired range in engineering units for the resulting value. By applying the high and low clamps you can ensure that your output will always stay within the limits you have set. This may not always be the case. If your raw data exceeds the range set by the raw value High and Low, it would force the scaled value beyond the range you have entered for the scaled value. The clamps prevent this from occurring. The Units field can be used to provide a string to the OPC client that describes the format or unit for the resulting engineering value. Use of the Units field requires an OPC client that can access the Data Access 2.0 Tag Properties data. If the client does not support these features, there is no need to configure the Units field. For more information, refer to [Scaling Properties](#).

If data has been entered as shown above, commit the scaling changes by pressing either the **OK** or **Apply** button.

Note: With the addition of online full time operation, technically users can begin using this project in an OPC client at this moment. For the sake of project preservation, proceed to [Saving and Testing the Project](#).

Designing a Project (Saving and Testing the Project)

After following the steps in this example, the project should now be configured with two user-defined tags. With the server's online full time operation, these tags can be accessed from an OPC client immediately even though the project has not been saved to disk. Like any Windows application, the project is saved using the **File|Save** menu selection. Once the file save dialog is displayed, give the project the name **Example1**. Once the project is saved, the server's operation can be tested using the OPC Quick Client.

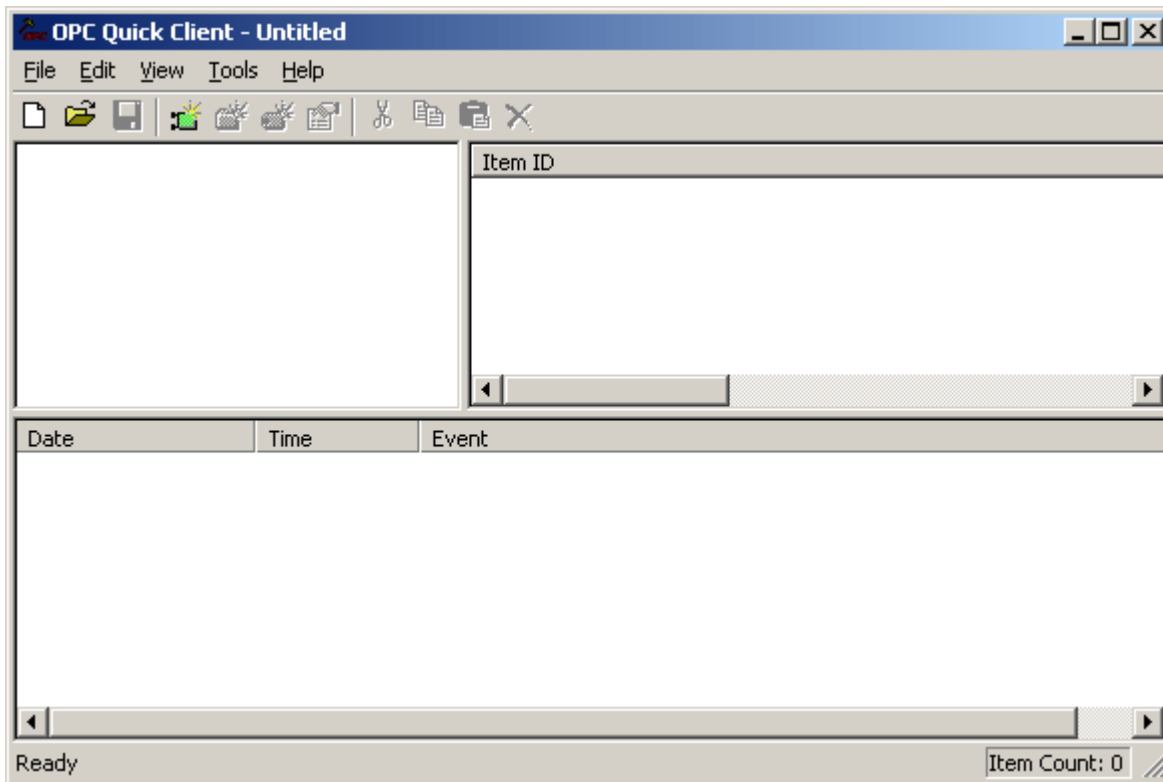
Note: OPC technology allows an OPC client application to automatically invoke an OPC server when the client needs data. The OPC server, however, needs to know what project to run when invoked in this fashion. The server will load the most recent project you have loaded or configured. To determine what project the server will load; simply look at the first project file listed in the "Most Recently Used" file list found on the file menu.

Testing Your Project

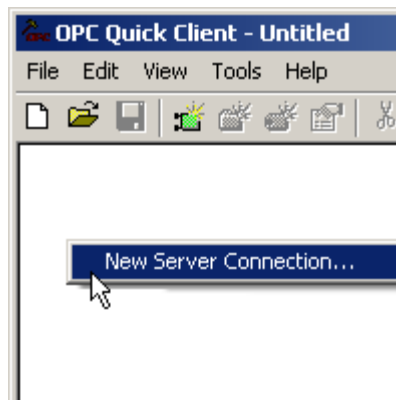
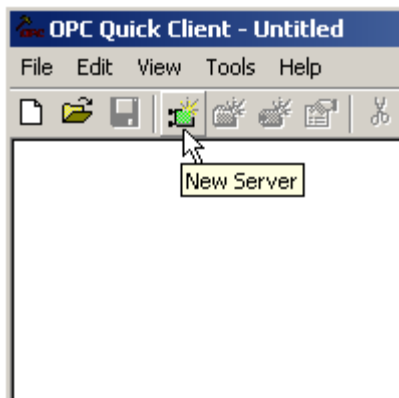
Included with the server is a full-featured OPC Quick client. The quick client supports all of the operations available in any OPC client application. The Quick Client can be used to access all of the data available in the server application. The quick client can be used to read and write data, perform structured test suites and test server performance. The

quick client also provides detailed feedback regarding any OPC errors returned by the server.

Find the OPC Quick Client program; it should be located in the same program group as the server. Run the OPC Quick Client to invoke the dialog shown below.

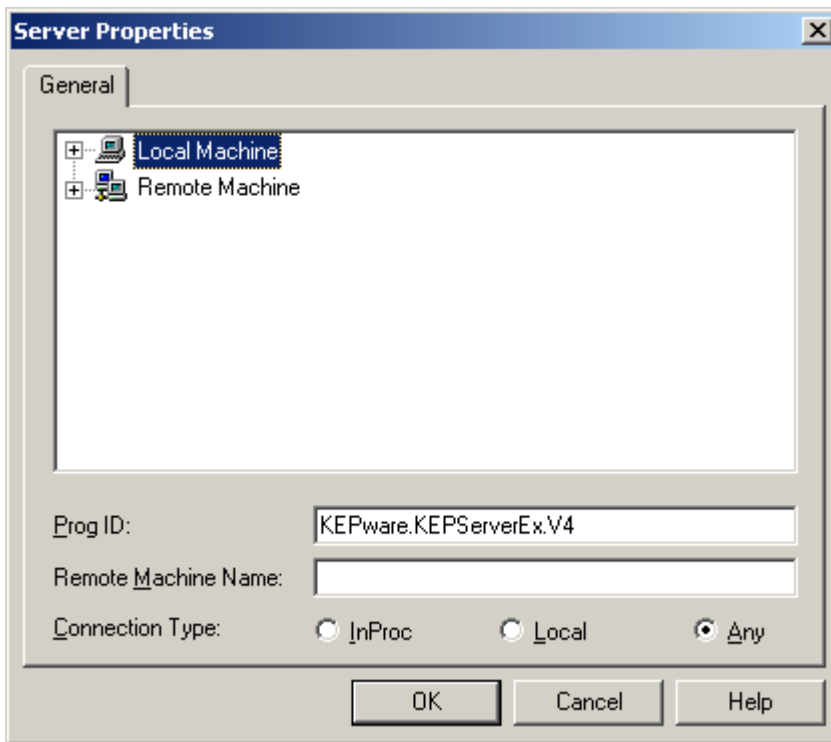


The first step in accessing any OPC server is to establish a connection. In the OPC Quick Client this is done by using the **Edit|New Server Connection**, the Toolbar New Server or the context menu. The following figures demonstrate two methods to establish a connection to an OPC server.



Connect to Server by Toolbar New Server. Connect to Server by Context.

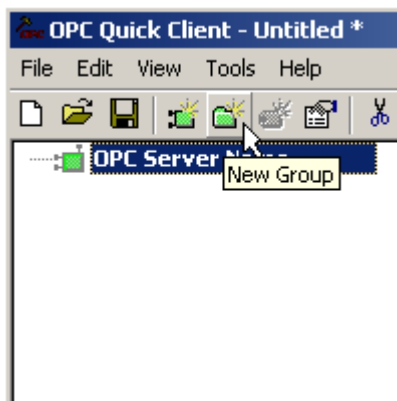
Clicking on the **New Server** button will invoke the OPC server selection dialog. The server selection dialog is used to make connections with an OPC either locally or remotely via DCOM. By default, the server selection dialog is configured with the Prog ID of the server. The name listed in the server connection dialog is called the **Prog ID** of the server. OPC clients use this name to reference a specific OPC server. The server connection dialog should appear as shown below.



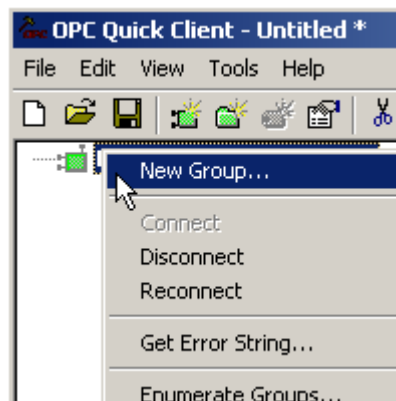
When an OPC server is selected and a connection is made, two things may happen. If the server is running, the OPC Quick Client will simply make a connection to the server. If the server is not already running, the server will startup automatically. It is this automatic startup described above that requires the server to know what project to load. "**Example1**" should be the running project if the example project was saved and the server has automatically started.

Once a connection has been made to the server a group needs to be added to the connection. Groups act as a container for any tags that will be accessed from the server. All OPC clients use groups to access OPC server data. Aside from providing a container for lists of tags, OPC groups also provide control over how those tags are updated. There are a number of properties attached to a group that allow the OPC client to determine how often data should be read from the tags, whether the tags are active or inactive, does a dead band apply, etc. These properties give the OPC client a great deal of control over how the OPC server operates.

To add a group to the connection, first select the server connection. Then, use either the **Edit | New Group**, the Toolbar New Server or the context menu. The following figures demonstrate two methods to add a group to an OPC server connection.



Adding a Group by Toolbar.



Adding a Group by Context Menu.

Pressing the New Group button will invoke the group properties dialog, which is used to configure how the group will interact with the OPC server. For the purposes of this example, edit the controls of the group properties dialog to match

the following dialog.

The screenshot shows a dialog box titled "Group Properties [Item Count: 2]". It has two tabs: "General" and "Interfaces". The "General" tab is selected. The dialog contains the following fields and controls:

- Name: ExampleGroup
- Update Rate (ms.): 100
- Time Bias (min.): 0
- Percent Deadband: 0
- Language ID: 1033
- Update Notification: OPC 2.0 (dropdown menu)
- Active State: Active State
- Keep Alive Rate (ms.): 0

At the bottom of the dialog are four buttons: OK, Cancel, Apply, and Help.

The key properties of the group at this point are the Update Rate, Dead band, and Active State. These three parameters control when and if data will be returned for the tags in this group. The Update Rate determines how often data will be scanned from the actual device and as a result of that scan how often data will be returned to the OPC client. The Percent Dead band is used to eliminate or reduce noise content in your data by only detecting changes when they exceed the percentage change that has been requested. Keep in mind that the percent change is a factor of the data type of a given tag. The Active State is used to turn all of the tags in this group either on or off. The group name is used for reference from the client and can actually be left blank. For more detail see the OPC Quick Client help. Press **OK** when finished in order to commit the group.

With a group added to the server, the OPC server tags can now be accessed. To do so, first add them to the group. Before OPC, accessing data within a communications product required users to remember the names of the available tags and re-enter them in the client application. One of the goals of the OPC Foundation was to make data access not only universally available, but also to make that access easier. To that end, the OPC Data Access specifications defines a tag browsing interface that allows an OPC client to directly access and display the available tags in an OPC server. By allowing the OPC client application to browse the tag space of the OPC server, a user can simply click on the desired tags to automatically add them to a group.

To add tags to a group, the group must first be selected. Then, use **Edit|New Item**, the Toolbar New Server or the context menu. After the **New Item** button has been invoked, the Add Items dialog should appear as shown below.

Add Items

Item Properties

Access Path:

Item ID:

Data Type: Native

Active

OK Cancel Help

Browsing

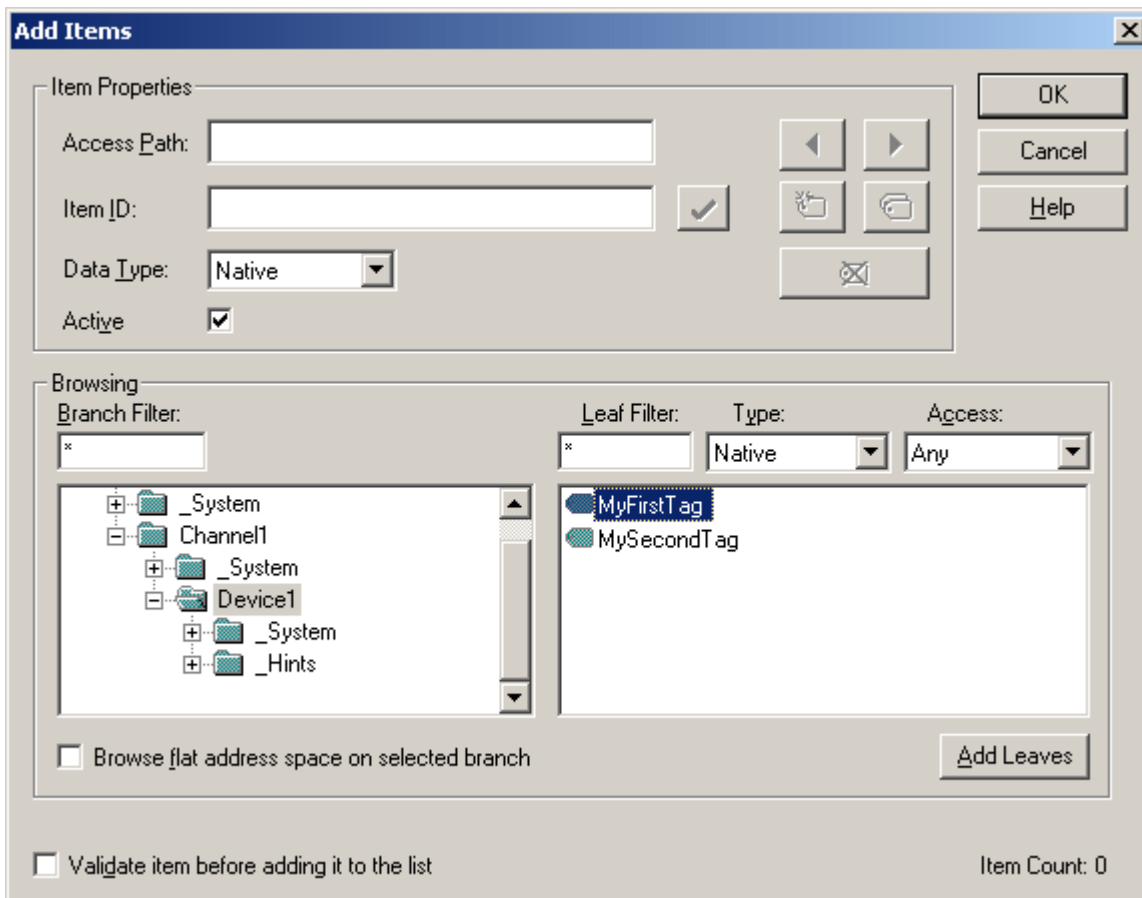
Branch Filter: Leaf Filter: Type: Native Access: Any

OPC Server Name

Browse flat address space on selected branch

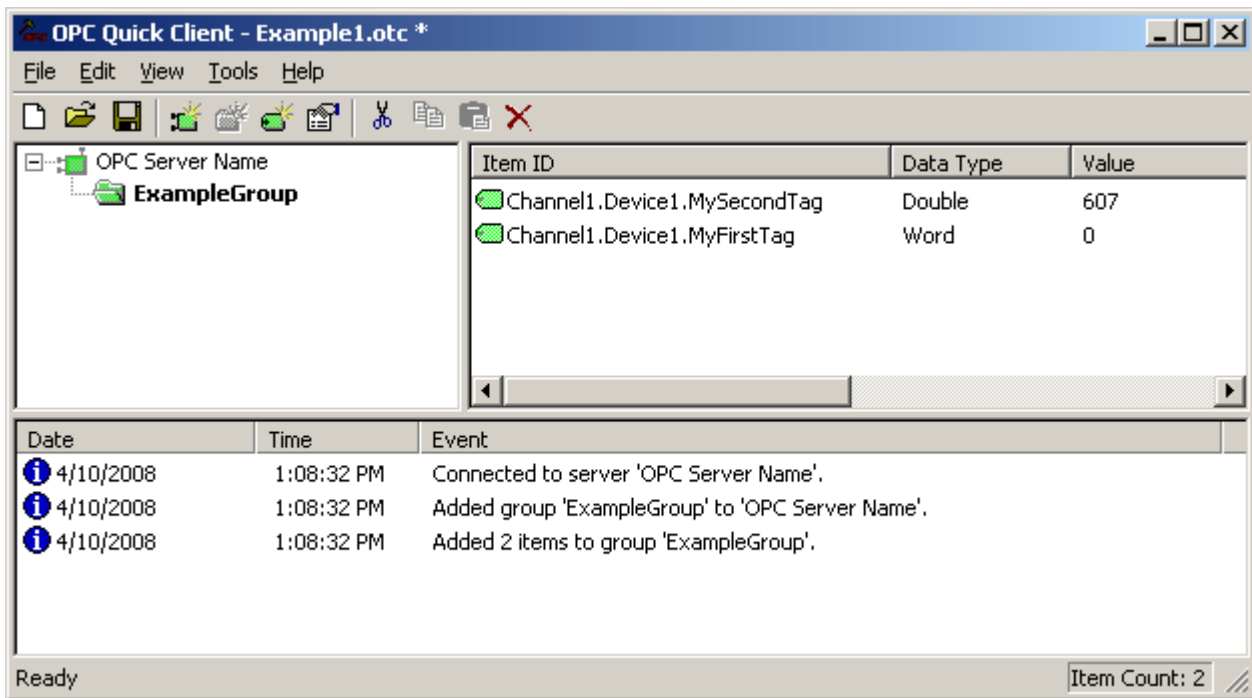
Validate item before adding it to the list Item Count: 0

The Add Items dialog is used to enter an Item ID, Data Type, and Active State for an OPC item. These fields can be used to manually enter an OPC item/tag. The Add Items dialog also provides an explorer like tree view in the Browsing section of the dialog. This explorer can be used as an interface by browsing an OPC server to find tags configured at the server. Using the "**Example1**" project created as part of this example, users can access the tags you defined by expanding the branches of the view. The newly defined tags should be visible in the branches of the tree view.



The tags to can be added to the OPC group by simply double-clicking on the tag name. As tags are added to the group, the "Item Count" at the bottom of the Add Items dialog will increase to indicate the number of item being added. If both "MyFirstTag" and "MySecondTag" were clicked, the item count should be 2. If this is the case, press **OK**.

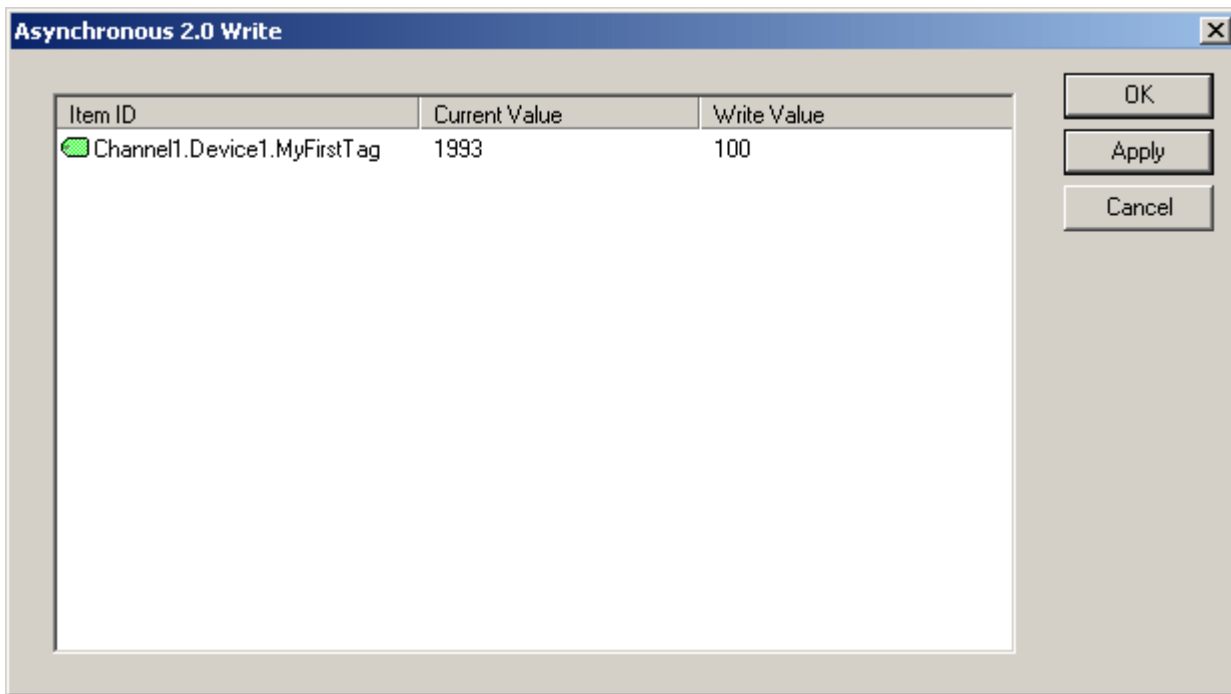
Data should now be able to be accessed from the server using the two newly defined tags. The OPC Quick Client should appear as shown below.



"**MyFirstTag**" should contain a changing value. The second tag should be zero at this point. To finish, test the reading of an OPC item. If changes are necessary, however, use one of the write methods to send new data to the OPC item.

There are two mechanisms for writing data to an OPC server: Synchronous Writes and Asynchronous Write. Synchronous Writes performs a write operation on the OPC server and waits for it to complete. Asynchronous Writes performs a write on the OPC server but it doesn't wait for the write to complete. Either method will work: their use is more of a factor in OPC client application design.

To write to an OPC item, first select the item and then display its context menu by right-clicking on one of the tags. Once the context menu is displayed, select either Synchronous or Asynchronous writes. In both cases, the write dialog will be presented. In this example, right-click on "**MyFirstTag**" and select Asynchronous Write.



With the write dialog displayed, the values continue to update. Users can enter a new value in for this item by clicking in the **Write Value** column for and then entering a different value. Once a new value is entered, the **Apply** button becomes active. Click Apply to write the data. Clicking Apply instead of **OK** allows users to continue writing new values: OK closes the dialog after writing the new value. Thus, when finished, press OK. If no new data has been entered, clicking OK will not send data to the server.

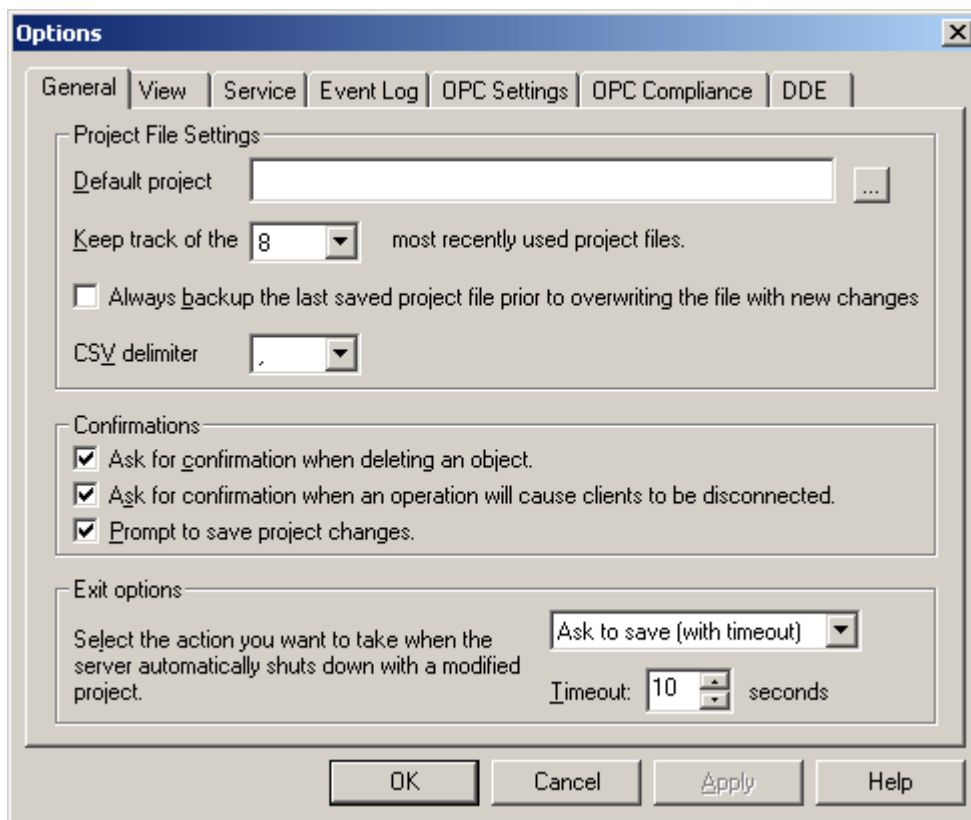
Conclusion

At this point, all of the basic steps involved in building and testing an OPC project have been discussed. At this point, users can continue to test various features of both the server and the OPC Quick Client. For more detailed information on the OPC Quick Client, refer to the driver's associated help file.

Note: If using Visual Basic, remember to examine the supplied example projects. These two projects provide both a simple and complex example of how OPC technology can be used directly in Visual Basic applications.

General Options

The server's online full time mode of operation places some special requirements on the server as well as the operator. Just about any parameter can be changed in the server at any time. This is a very powerful feature that can also cause a lot of problems for OPC applications. To prevent an operator from making a change that might disrupt the ability of the server to provide data to client applications, warning pop-ups are presented whenever a change will affect active connections. The General Option dialog is used to tailor the presentation of these warning pop-ups. Additionally, as changes are made to a project, the saving of these changes also requires possible operator interaction. The operator interaction required when a project requires saving can also be modified.



The **Default Project** setting is used to configure a specific project that will be loaded when the OPC server is loaded manually by the user, loaded automatically by an OPC client or invoked as an **NT service**. By specifying a Default project, users ensure that the server will always load the correct application regardless of any changes or modifications made to other projects. Once this is set, the settings will need to be modified in order to load a new project. This selection is undefined by default, thus allowing the server to operate in its standard mode. When no Default project is selected, then the most recent project (i.e., the last saved project) will be loaded.

Keep track of the ___ most recently used project files: This setting is used to specify how many project files are presented on the MRU (most recently used) list of projects found on the File menu. The valid range is 1 to 16.

Check **Always backup the last saved project prior to overwriting the file with new changes** in order to have the system automatically make a backup copy of the last saved project (*.opf) before overwriting that file with the new project file. The backup file will be named *projectname.opf.bak* and stored in the \Project Backups folder.

Use the **CSV delimiter** setting to choose the comma-separated variable that the server will use during the import and export of tag data in a CSV (comma separated variable) file. Choose either a comma (default) or a semicolon. **See Also: [CSV Import and Export](#).**

The **Confirmations** settings are used to specify the conditions that will force the server to present warning pop-ups to an operator. When the **Deleting an object** selection is enabled, all delete operations will cause a warning popup to be displayed to the operator requiring confirmation before the delete operation is completed. When the **Disconnect** selection is enabled, all operations that would cause client applications to be disconnected from the server will cause a warning popup to be displayed requiring confirmation before the disconnect sequence is initiated. If the **Prompt to save** selection is enabled, a popup will be displayed if the server is being shut down with outstanding changes in the project.

OPC Note: The server supports the Data Access 2.0 server shutdown event. If an operator action will cause connections to OPC clients that are using the 2.0 Data Access specifications to be lost, the server will issue a server shut down event to these clients. If the clients are designed to utilize this event they can properly disconnect their OPC resources. OPC clients that do not support the server shutdown event may have issues once the server shuts down. When an OPC server is invoked from a client application the server automatically loads itself and begins running the last project used. Normally when the OPC client is finished using the OPC server and disconnects, the OPC server would

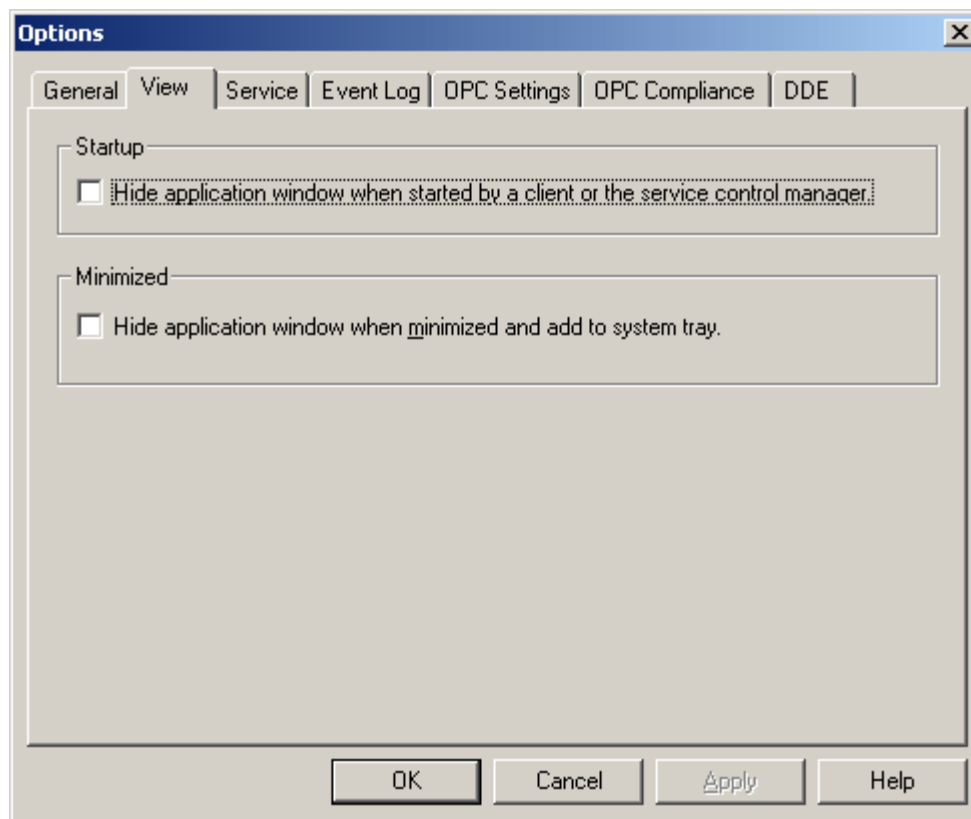
then exit. Due to the server's online full time mode of operation the operator may have made changes to the project while it was invoked by a client application. If the server were to shut down without warning, those changes could be lost. To prevent this from happening the server has provided exit options. The exit options allow the exit strategy of the server to be tailored to the application. **See Also:** [OPC Settings](#).

There are four exit strategies: **Ask to save (with timeout)**, **Auto-save (if possible)**, **Discard changes** and **Keep running**. The **Ask to save (with timeout)** method will present a save project dialog to the operator. If the operator is present they can decide to save the changes. If the operator is not present, the option to save the project changes will expire after the time specified by the **Timeout** parameter, which has a range of 3 to 30 seconds. If the operator does not respond within the timeout period the project changes will be lost and the server will complete its exit. The **Auto-save (if possible)** method will automatically save any changes made to the project if a project file has already been established. With this method the operator will not be prompted to save changes. The **Discard changes** method simply allows the server to exit losing any changes that have been made to the project. The **Keep running** method simply prevents the server from automatically shutting down if changes have been made to the project.

View Options

The View Options menu is used to configure when the server's main window is displayed to the user and whether or not the server appears on the Window's task bar. The server's main window is usually displayed to the user whenever the server is either loaded manually or automatically invoked by an OPC client application. This mode of operation is the default condition and is selected by leaving the **Startup** and **Minimized** options unchecked. This is also how the server has operated historically. These two options can also be used to configure how the server behaves when launched either manually or automatically.

The first option, **Startup**, configures how the server will respond when being launched automatically by an OPC client or through the NT service manager when configured to run as a service. If this selection is checked, the server will not appear on the normal Window's task bar but instead will appear on the System tray as a Server Icon. Once placed on the System tray, users can still interact with the server by clicking on the icon.



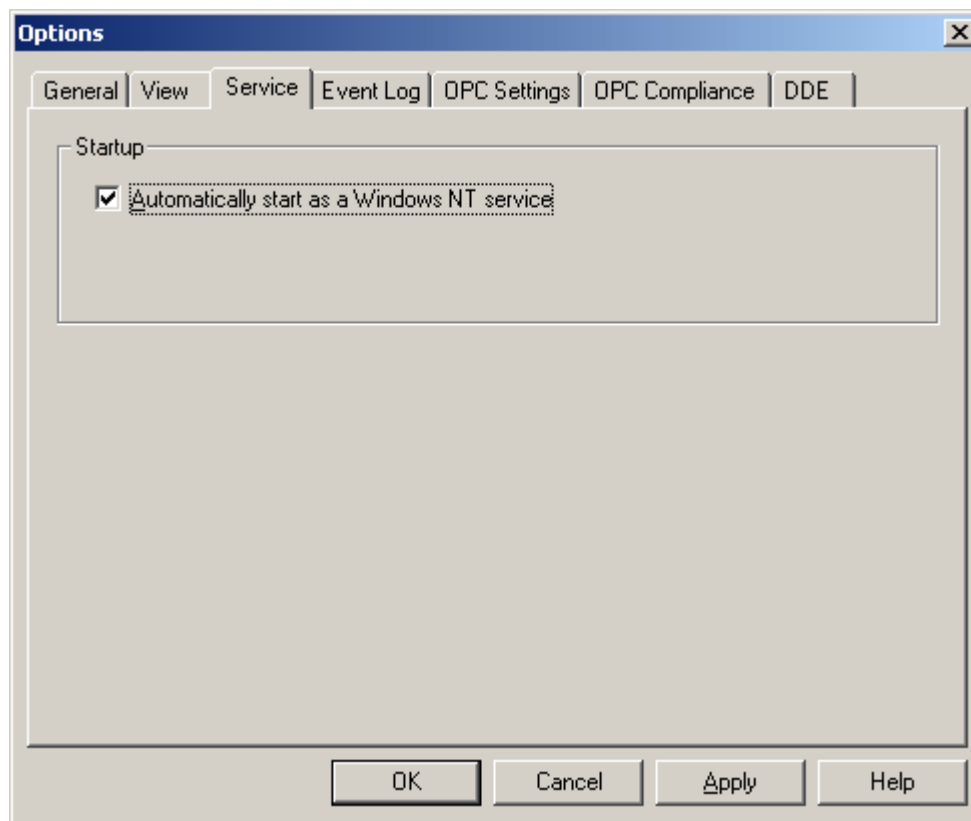
The second option, **Minimized**, is used to configure how the server will respond when the main window is minimized. If this selection is left unchecked the server application will continue to appear on the Window's task bar when minimized. If this selection is checked, the server will only be displayed as an icon on the System tray when it is minimized. This

can help remove clutter from the Window's environment and help in preventing the user from unnecessarily interacting with the server project.

Service Options

The server supports running as a service under Windows NT/2000/XP/Server2003. The ability to run as an NT service is crucial for many applications where the server must provide data to OPCclients via DCOM. For these applications, the loss of a DCOMconnection cannot be tolerated. Normally an OPCserver that only supports stand alone program operation is forced to shut down when its host machine experiences a user login or logout. While running as a service, the server can continue to supply OPCdata across user loginsessions. As a service, the server can be configured to run with no visible presence. It can also be configured to interact with the desktop, which is used to make changes to your server project.

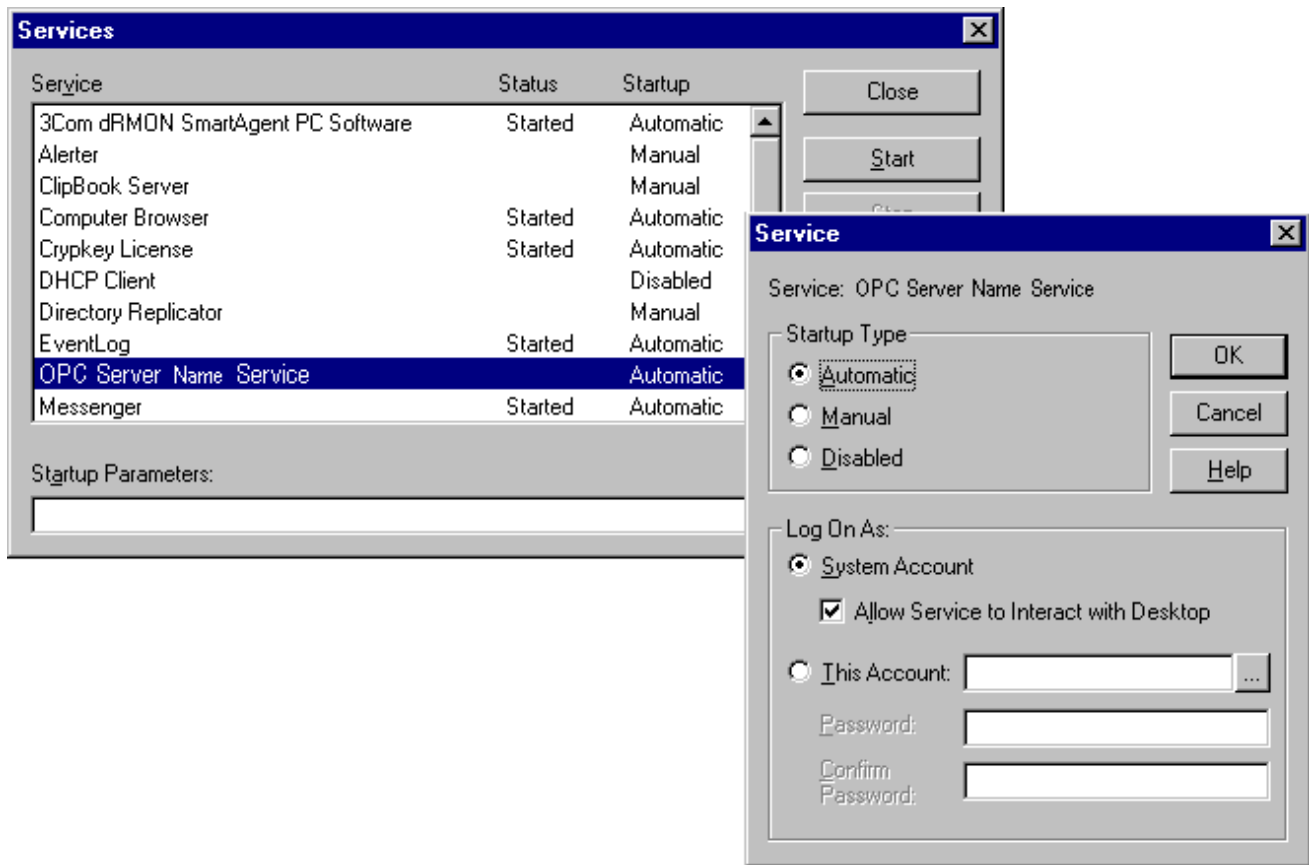
To run the server as a service, click **Service | Tools|Options**.



The **Automatically start as a Windows NT service** option is used to configure the server for operation as a service under Windows NT/2000/XP/Server2003. By default this option is disabled (unchecked). When the service option is enabled, users will be presented with a message dialog warning that this change will take effect on the next server run. To make the server become a service immediately, exit the application and then either restart it from the Windows NT Service Manager found on the Control Panel or allow the OPCclient to invoke the server when it connects.

Note: The Service option of the server can only be accessed when you are logged into Windows NT/2000/XP/Server2003 as an administrator. If logged in as a normal user, the **Run as service** option will be disabled (grayed out).

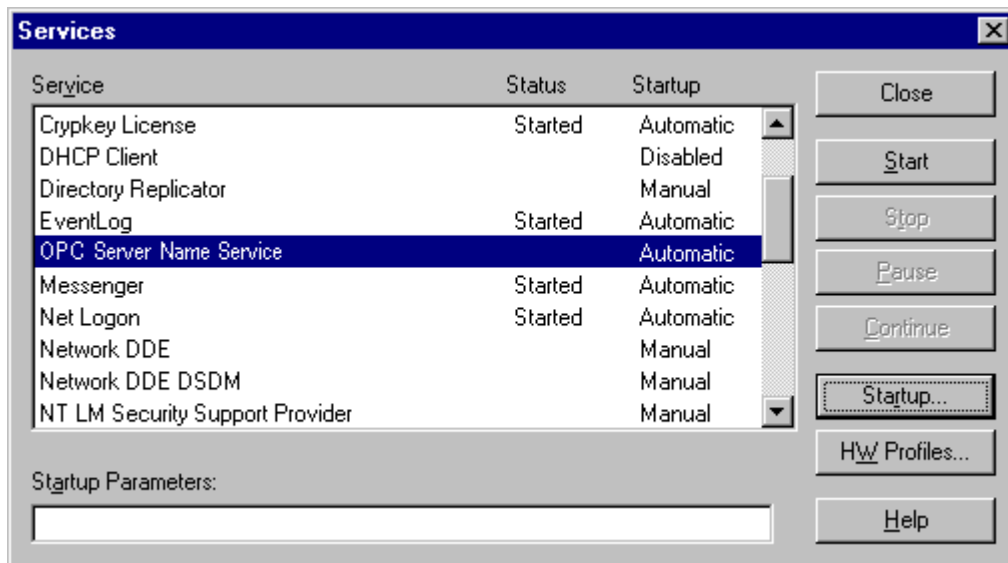
Once the service option has been configured, the NT service options may need to be modified in order to setup the server for the specific application. The image below shows the NT Service Manager's properties dialog for the server with its default selections.



By default the server is configured for automatic startup and interaction with the desktop. Interaction with the Desktop means that you will still see the server on your task bar and that you will be able to make changes to your server project even though it is running as a service. However, you will not be able to shut the server down from any of the normal menu exits or close functions. Allowing the server to interact with the desktop has no effect on its ability to supply OPC data across user logins. Once you have your server project completely configured you may want to consider disabling the server's interaction with the desktop. This will prevent users from seeing the server on their task bar and further reduce unauthorized access to the server. To prevent the server from interacting with the desktop simply uncheck the "Allow Service to Interact with Desktop" check box. Once you disable desktop interaction you will need to restart the server to remove it from your system taskbar.

Note: We strongly recommend that users continue to allow the server to **Interact with Desktop** while it is running in a demo or evaluation mode. This will allow users to see any error messages that the server may generate such as **Demo period has expired**. If this occurs, stop the service using the Service manager.

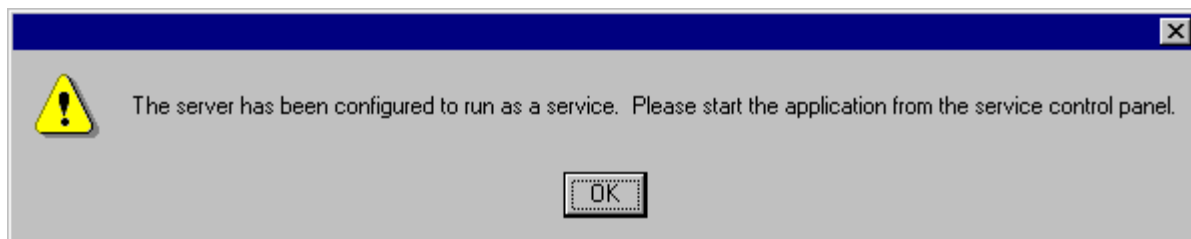
Once the server is configured to run as a service, can use the NT Service manager to manually start and stop the server. The Service Manager menu is used to select the server by name and either start or stop the service.



Returning the Server to Normal Program Mode

To return the server to normal stand alone program operation, users must be able to interact with their server project. This requires that the server be able to interact with the desktop. If the server is not configured to interact with the desktop, use the NT Services Manager to enable desktop interaction then stop the server service. Once the server has stopped, restart it using the Start button as shown above. The server should now be visible on the system taskbar. Changes can then be made to the server project. To return to normal program mode uncheck the **Automatically start as a Windows NT service** checkbox in the Service Option dialog. Next, stop the server using the Stop button of the Service manager. The server should now be able to be run from the normal desktop icon or Start menu selection.

If users attempt to run the from the desktop icon or start menu while it is configured to run as a service, the following message box will be displayed.



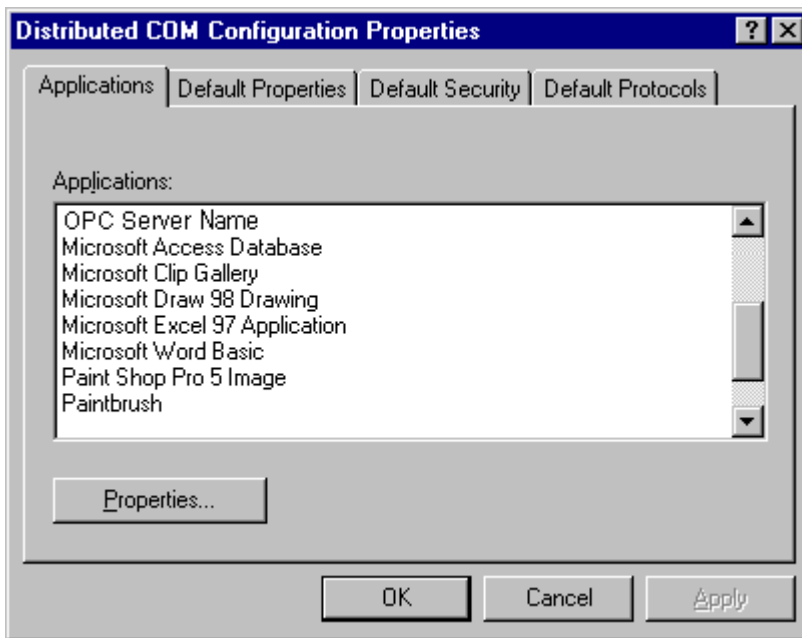
If the server is required to run as a normal program, review the steps in **Returning the Server to Normal Program Mode** above.

Possible issues when running as a service

Under most circumstances, users should be able to seamlessly switch between **Service mode** and **Stand Alone mode**. There is the possibility of a DCOM (Distributed Component Object Model) security issue that may prevent the OPC client from being able to connect to the server when it is running as a service.

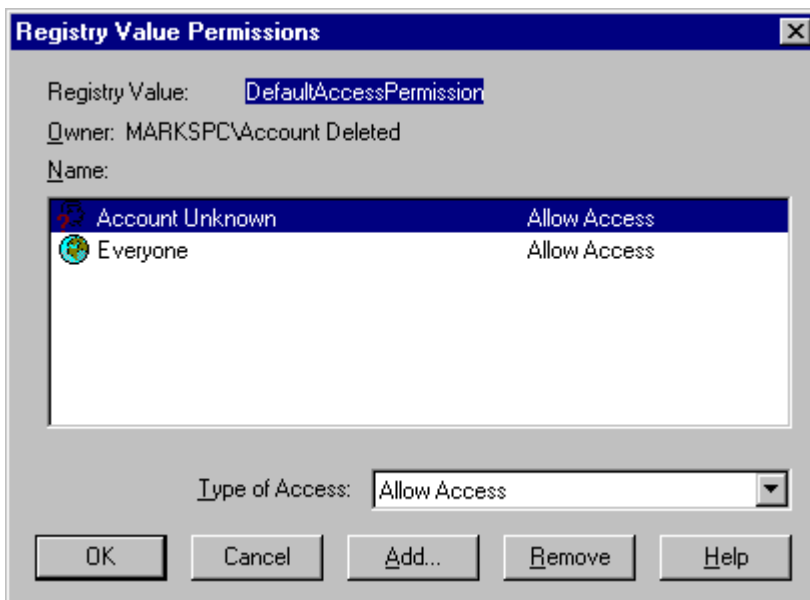
If you experience an issue connecting your OPC client to the server when it is running as service, we suggest that you first take the server out of service mode and confirm your OPC connection in Stand Alone mode. Once the OPC client connection has been confirmed under normal circumstances, it is best to check the DCOM settings.

1. Place the server back into Service mode.
2. Run DCOMConfiguration. This can be done from the windows start menu using the Run command. Type "DCOMCNFG."
3. Once DCOMCNFG is running, the following dialog will be invoked.



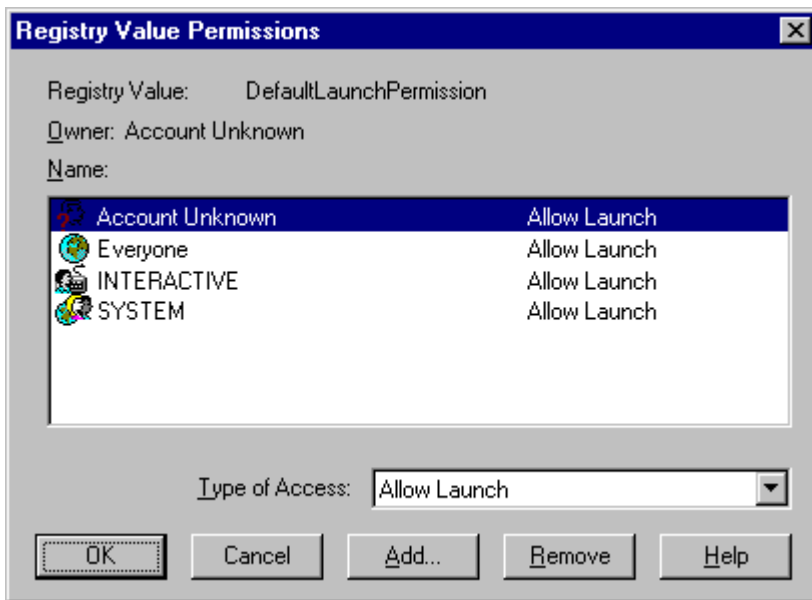
4. Select the **Default Security** tab. There will be three security groupings: default **Access**, **Launch** and **Configuration**.

5. In the default **Access** group, click **Edit Default**.



6. Make sure that the **Everyone** name is a member of this group. If not, click **Add** and select **Everyone** from the list. Set the type of access to **Allow Access**. Press **OK**.

7. In **Default Launch**, click **Edit Default**.



8. Make that both the **Everyone** name and **SYSTEM** name are members of the group. If not, click **Add** and select **Everyone** and **SYSTEM** from the list. Set the type of access to **Allow Launch**. Press **OK**.

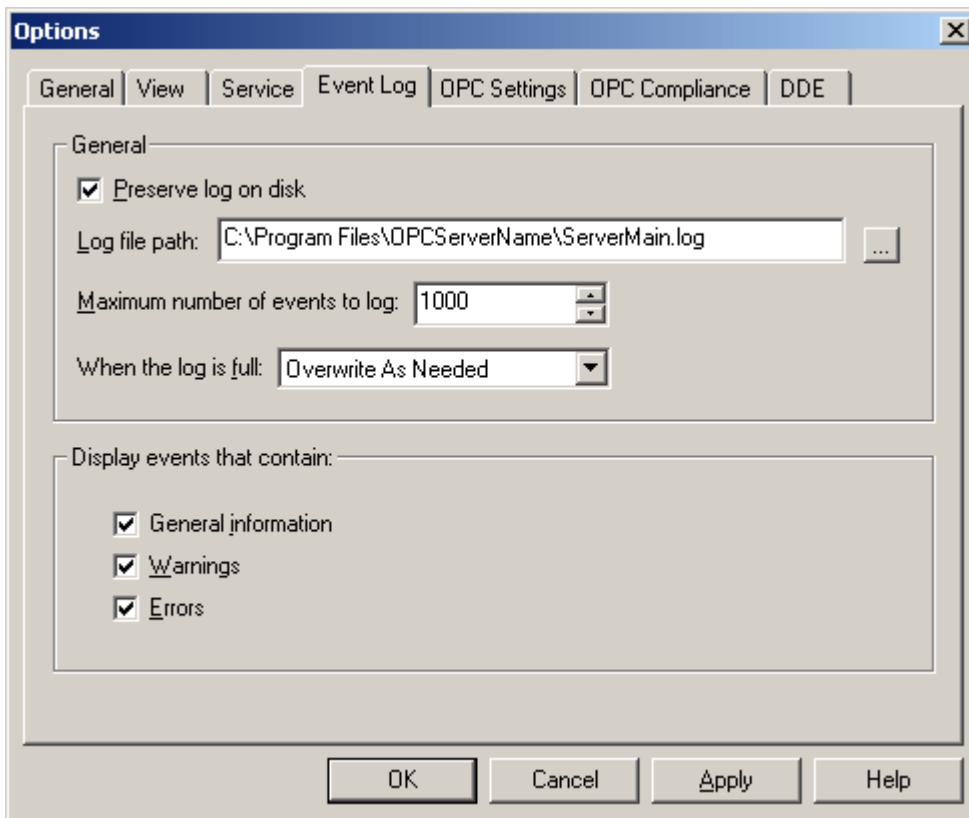
9. Stop and restart the Server as a service through the **Service Control Manager**.

10. Attempt to reconnect to the server using the OPC client.

Note: If still unable to connect at this point, reboot the PC. In most cases, DCOM settings take effect immediately, but sometimes the PC must be restarted.

Event Log Options

The server supports a persistent event logger. Events that occur within the server can be stored to disk. The data contained in the event log is dependent upon the options selected here. There are a number of options that allow Event Log operation to be tailored to the needs of the application.



The **Preserve log on disk** enables the use of a disk based log file. When enabled all events in the server will be maintained on disk from one run to the next. If disabled, the event logging system of the server will simply be done in memory and no disk log will be generated. When the preserve log file option is disabled, the event log contents will be empty each time the server is run.

The **Maximum number of events** parameter determines the number of records the log system will hold before the log full action comes into effect. The valid range is 100 to 30000 records. The default value is 1000 records. If you attempt to change this parameter to a value that is less than the current number of records in the log you will be given a warning that log file truncation will occur.

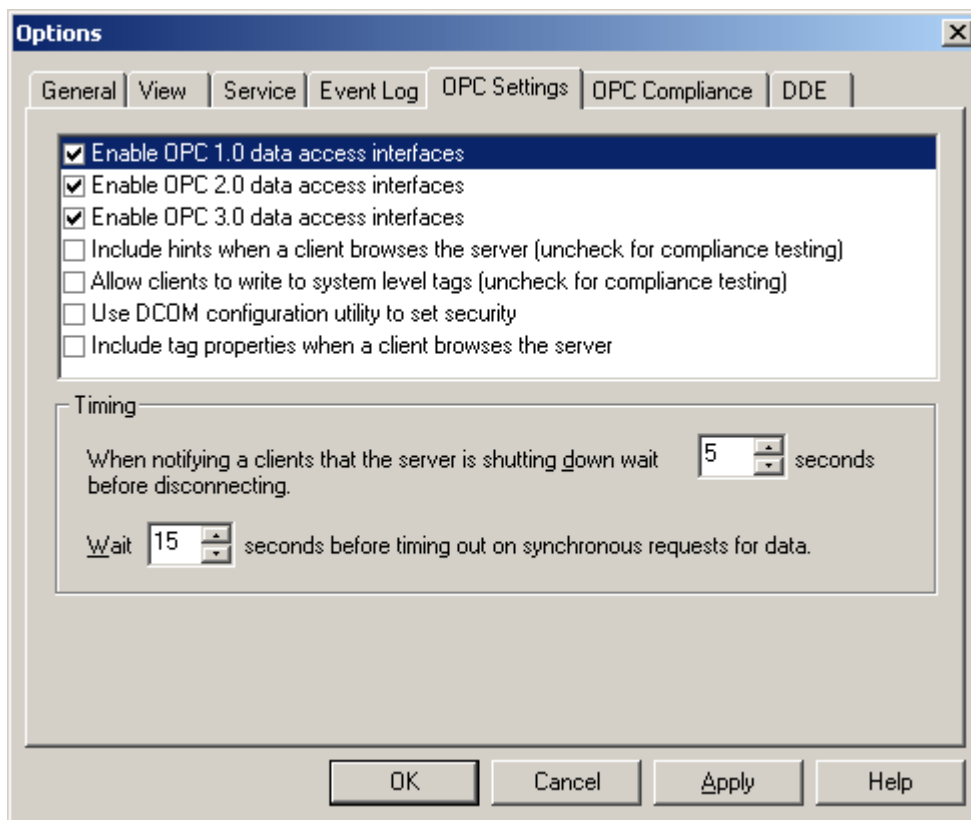
The **Log Full** parameter determines the course of action that should be taken when the event log reaches the maximum number of events. There are two possible options available, **Overwrite As Needed** and **Halt (Requires Manual Reset)**. When set to "Overwrite As Needed," the event log acts as a circular buffer with each new event replacing the oldest event once the log is full. When set to "Halt, (Requires Manual Reset)," the event log will stop accepting new events and maintain the current list of events until an operator manually resets the event log. This can be useful when trying to catch elusive error conditions.

The contents of the event log can be tailored to meet the reporting requirements of the application. Currently there are three types of messages that can be placed into the event log; **General** messages such as server startup and shutdown messages, **Warnings** such as device not responding and **Errors** such as the rejection of bad OPC item request.

The event log system would be useless if there was no mechanism to protect the contents of the log. If operators could change these parameters or reset the log, the purpose would be lost. To prevent these actions from occurring, use the [User Manager](#) to limit what functions an operator can access.

OPC Options

The server supports the OPC Foundation's Data Access Specifications for 1.0, 2.0 and 3.0 simultaneously. While this provides the up most level of compatibility there may be times when forcing the server to use one method over another may be necessary. The OPC Options dialog is used to make these selections.



The **Enable OPC 1.0** selection, when enabled, allows the server to accept OPC client connections from OPC clients that support the 1.0 specification. By default, 1.0 operation is enabled.

The **Enable OPC 2.0** selection, when enabled, allows the server to accept OPC client connections from OPC clients that support the 2.0 specification. By default, 2.0 operation is enabled.

The **Enable OPC 3.0** selection, when enabled, allows the server to accept OPC client connections from OPC clients that support the 3.0 specification. By default, 3.0 operation is enabled.

The **Include hints** selection, when enabled, allows OPC client applications to browse the address formatting hints available from each communications driver. The hints provide a visual quick reference on how a particular device's data can be addressed. This can be useful when entering **dynamic tags** from the OPC client. The hint items are however not valid OPC tags. Some OPC client applications may try to add the hint tags to their tag database. When the OPC client attempts to add a hint item it will receive an error from the server. For most clients, this is not a problem, for some however it can cause them to stop adding tags automatically or report errors. To prevent this from happening this option is used to turn the hints either On or Off. By default, hints are enabled (On).

Note: If you are running the OPC Compliancy Test Tool this option should be turned (Off).

The **Allow clients to write to system level tags** option controls write access to system tags like the **Enable** tag on a given device. In some cases you may not want a client application to have the ability to turn a device on or off in your project. This setting applies to all system level tags. Special **Note:** If running the OPC Compliancy Test Tool, this option should be turned (Off). The default condition is (On).

The **Use DCOM configuration utility to set security** options control how the server will respond to DCOM security request. When enabled you can use the DCOM configuration utility to establish strong security on who or what can access the server. When this option is disabled the server will reduce the level of DCOM security making it much easier for applications to connect to the server. If experiencing DCOM security issues, try disabling this option. For new applications this option defaults to the (Off/disabled) state.

The **Include Tag Properties** selection, when enabled, allows OPC client applications to browse the Tag Properties available for each tag in the address space. By default this setting is disabled. Changing these options can be useful

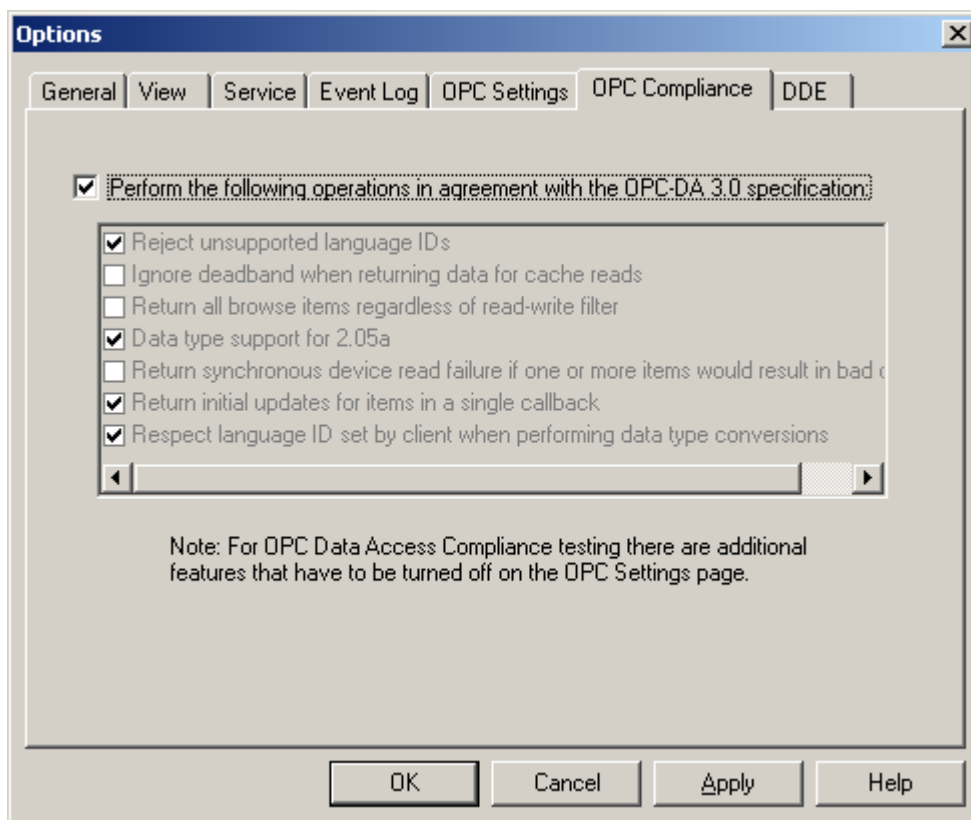
when the OPC client application may have problems with a particular version of the specification.

The **Shut down wait** timeout is used to configure how long the server will wait for an OPC client to return from the server shut down event. If the client application does not return within the timeout period the server will complete its shutdown and exit. For this setting; the valid range is 1 to 30 seconds, the default is 5 seconds.

The **Wait for synchronous request** parameter is used to configure how long the server will wait for a synchronous read or write operation to complete. If a synchronous operation is in progress and the timeout is exceeded, the server will force the operation to complete with a failure to the OPC client. This prevents OPC clients from appearing to become locked up when using synchronous operations. The valid range is 5 to 60 seconds, with 15 seconds as a default.

OPC Compliancy Options

The server has been designed to provide the highest level of compatibility with the OPC Foundation's specifications. In testing however it has been found that being fully compatible with the specification and working with all OPC client applications is a different matter. The OPC Compliancy option allows you to tailor the operation of the server to better meet the needs of rare OPC clients. Normally these options will not need to be adjusted for a majority of the OPC client applications you will encounter. The OPC compliancy dialog appears as follows:



The **Perform the following operations** selection is the master enabling switch for the options present in the list box. When enabled, the server will set all options to conform to OPC compliancy. This setting is not enabled by default.

The **Reject unsupported language IDs** selection will, when enabled, only allow language IDs that are natively supported by the server. If your OPC client application attempts to add an OPC group to the server and receives a general failure, it is possible the client has given the server a language ID that is not natively supported. If this occurs the server will reject the group addition. To resolve this particular issue you can disable this compliant feature to force the server to accept any language ID.

The **Ignore dead-band when returning data for cache needs** selection will, when enabled, allow the server to ignore the dead-band setting on OPC groups added to the server. Some OPC clients have had problems passing the correct value for dead-band. This problem may manifest itself in your OPC client as having good data but the data does

not appear to be updating frequently or at all. Like the language ID issue this condition is rare, as such this selection should normally be left in its default state of disabled.

The **Return all browse items regardless of read-write filter** selection will, when enabled, cause the server to return all tags to an OPC client application when a browse request is made regardless of the access filter applied to the OPC clients tag browser.

The **Data type support for 2.05a** selection will, when enabled, cause the server to adhere to the data type requirements and expected behaviors for data type coercion that were added to the 2.05a specification.

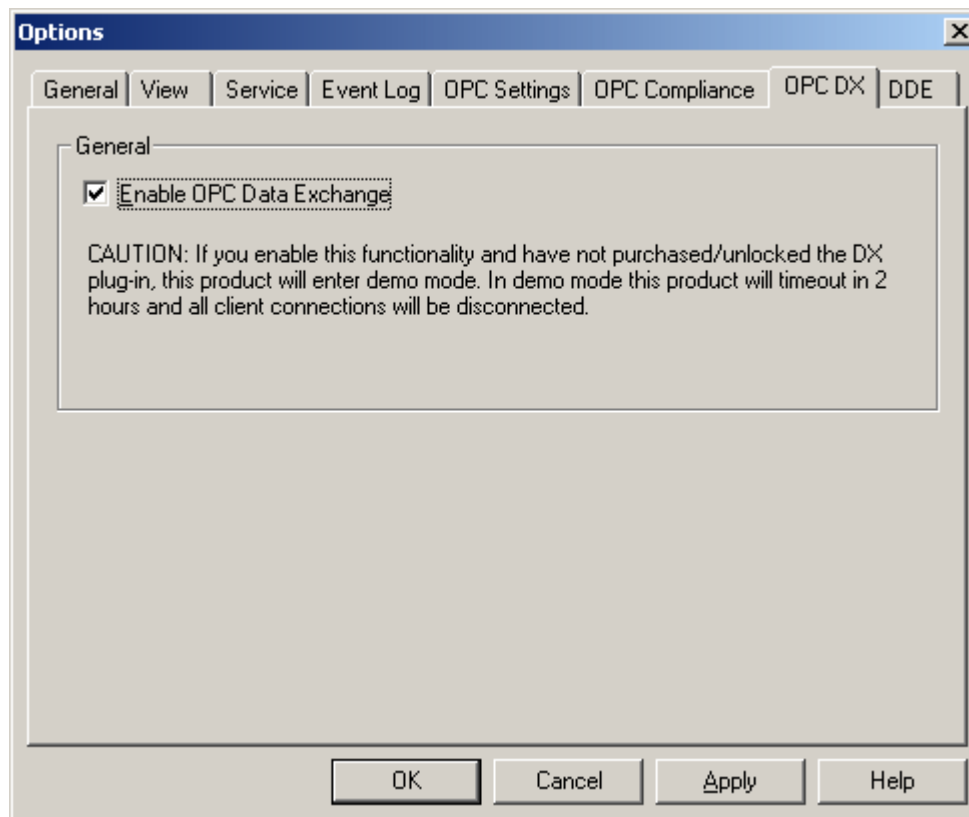
The **Return synchronous device read failure if one or more items would result in bad quality** selection will, when enabled, cause the server to return a failure if one or more items for a synchronous device read results in a bad quality read. Compliance requires the server to return success indicating that the server could complete the request, even though the data for one or more items may include a bad and/or uncertain quality.

The **Return initial updates for items in a single callback** selection will, when enabled, cause the server to return all outstanding initial item updates in a single callback. When not selected, the server will return initial updates as they are available which could result in multiple callbacks.

The **Respect Language ID set by client when performing data type conversions** selection will, when enabled, determine whether the server uses the Locale ID of the running Windows Operating System or the Locale ID set by the OPC client when performing data type conversions. For example, a string representing a floating point number such as 1,200 would be converted to One Thousand - Twelve Hundred if converted using English metrics, but would be One and Two-Tenths if converted using German metrics. Thus, if you have German software running on an English OS, you need to determine how the comma will be handled. This setting allows for such flexibility. By default, and due to historical implementation, the server respects the Locale ID of the operating system.

OPC DX Options

The OPC-DX Options page is used to enable or disable OPC Data Exchange functionality in the server. When OPC-DX is enabled, the server will register itself as an OPC-DX server that can be configured by any compliant OPC Data Exchange Configuration Client. When this setting is disabled, the OPC-DX functionality is not available.



Important: This page is only displayed if the OPC-DX plug-in has been installed on the system. To install the DX plug-

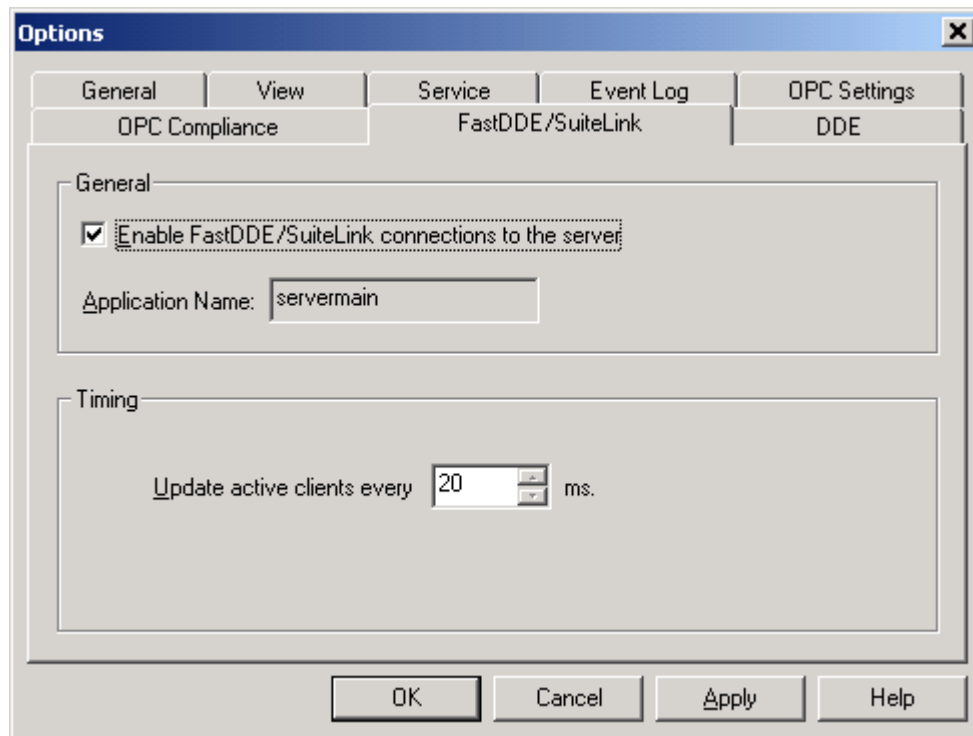
in, re-run the installation and select the DX plug-in as part of the installation modification options.

Note: For more information, refer to [OPC Data Exchange Plug-in](#).

FastDDE & SuiteLink Options

The server's support of Wonderware Corporation's FastDDE and SuiteLink simplifies the task of connecting the server with FactorySuite applications. The server uses the Wonderware connectivity toolkit to simultaneously provide OPC and FastDDE/SuiteLink connectivity while allowing for quick access to device data without the use of intermediary bridging software.

Important: In order for this page to be displayed in the server's **Tools | Options** menu, the Wonderware FS2000 Common Components or the InTouch Runtime Component version 8.0 or higher must be installed on the PC for proper FastDDE/SuiteLink operation.



The **Enable FastDDE/SuiteLink** option allows support of these Client/Server protocols to be turned on or off. By default this setting will be enabled when a Wonderware product is installed on the PC. If the FastDDE/SuiteLink operation is turned off, the server will not respond to any request for FastDDE or SuiteLink data. If the server will only be used for OPC connectivity, it is recommended that FastDDE/SuiteLink operation should be disabled for better performance and security.

The **Application Name** for FastDDE/SuiteLink is set to "servermain".

The **Update active clients** option configures how often new data will be sent to FastDDE/SuiteLink client applications. The range is 20 to 32000 milliseconds, with a default of 100 milliseconds. The "Update active clients" timer is used to allow FastDDE/SuiteLink data to be batched up for transfer to client applications. When using a Client/Server protocol like FastDDE or SuiteLink, performance gains only come when large blocks of server data can be sent in a single response. To improve the ability of the server to gather a large block of data, the update timer can be set to allow a pool of new data to accumulate before being sent to a client application. An important fact to remember is this update rate applies to how often data is sent to the client application, not how often data is read from the device. The tag setting "scan rate" can be used to adjust how fast or slow the server acquires data from an attached device. For more information, refer to [Scan Rate](#).

Using Server data in Wonderware

This section describes FastDDE/SuiteLink server support and how it can be used to get data into the FactorySuite

application. It does not cover the specifics of connecting with regular DDE or FactorySuite. Those specifics can be found either in the connectivity guide supplied with the purchased product or on our web site.

Note: This server is first and foremost an OPC server. As such, the common FastDDE/SuiteLink nomenclature of "Application name," "Topic name," and "Item name" is applied differently than its OPC equivalent.

Adding an Access Name in Wonderware's FactorySuite

When using the **Add Access Name** dialogue in Wonderware's FactorySuite application, the following information is required.

Access Name

This is an arbitrary **Access Name** that will represent the connection.

Node Name

If intending to use SuiteLink as the *remote* Client/Server protocol, configure a PC **Node Name** which matched the name of the PC.

Note: If the server and FactorySuite application exist on the same PC, this would not apply and the field would be left blank.

Application Name

The Application Name must match the server's predefined FastDDE/SuiteLink service name of "ervermain" for proper FastDDE and SuiteLink operation. When FastDDE/SuiteLink is enabled in the server, it registers itself as a FastDDE/SuiteLink service using the server's executable filename of "servermain" as the Application Name. If planning to use regular DDE, utilize the [DDE service name](#) provided in the DDE options page of the server.

Topic Name

The **Topic Name** identifies a group or category of data in the server application. If planning to use the fully qualified item path in the FactorySuite application's Tag Dictionary, specify the **Topic Name** with the server's default topic name of "_ddedata". However, planning to add many items to the Tag Dictionary, create an [alias](#) for the item path in the server and use it as the topic name. This method would only require specifying the Access Name and the tag item in the FactorySuite application's Tag Dictionary.

Choosing Protocol

For FastDDE or regular DDE, select **DDE** as the protocol; otherwise select **SuiteLink**.

Using the Tagname Dictionary in Wonderware's FactorySuite

As shown in the **Add Access Name** above, the use of **_ddedata** as the Topic Name requires the full item path be defined in the Tagname Dictionary dialogue.

Tagname Dictionary

Main Details Alarms Details & Alarms Members

New Restore Delete Save << Select... >> Cancel Close

Tagname: ToolDepth Type: ... I/O Integer

Group: ... \$System Read only Read Write

Comment: AccessLevel

Log Data Log Events Retentive Value Retentive Parameters

Initial Value: 0 Min EU: -32768 Max EU: 32767

Deadband: 0 Min Raw: -32768 Max Raw: 32767

Eng Units: Log Deadband: 0

Conversion: Linear Square Root

Access Name: ... AccessName_1

Item: Channel1.Device1.ToolDepth Use Tagname as Item Name

However, if **alias** names are to be used as Topic Names, a FactorySuite Access Name can be defined that contains all of the information needed to access a data item in the server as illustrated in the following figures.

Add Access Name

Access Name: AccessName_1 OK

Node Name: MyPC Cancel

Application Name: Servermain

Topic Name: Channel1_Device1_Machine1_Cell2

Which protocol to use: DDE SuiteLink

When to advise server: Advise all items Advise only active items

As can be seen below, if **alias** names are used as your Topic Name, it only requires specifying the Access Name and the tag item in the FactorySuite application's Tag Dictionary.

Operational Note: While the **Channel Name**, **Device Name**, and possible **sub group names** of the server can be any thing that fits the needs of the application and is allowed for that parameter, we strongly suggest keeping these names to the smallest length that will properly display their meaning. This will ensure the application can take advantage of future enhancements that could simplify connecting the server with the Wonderware FactorySuite application.

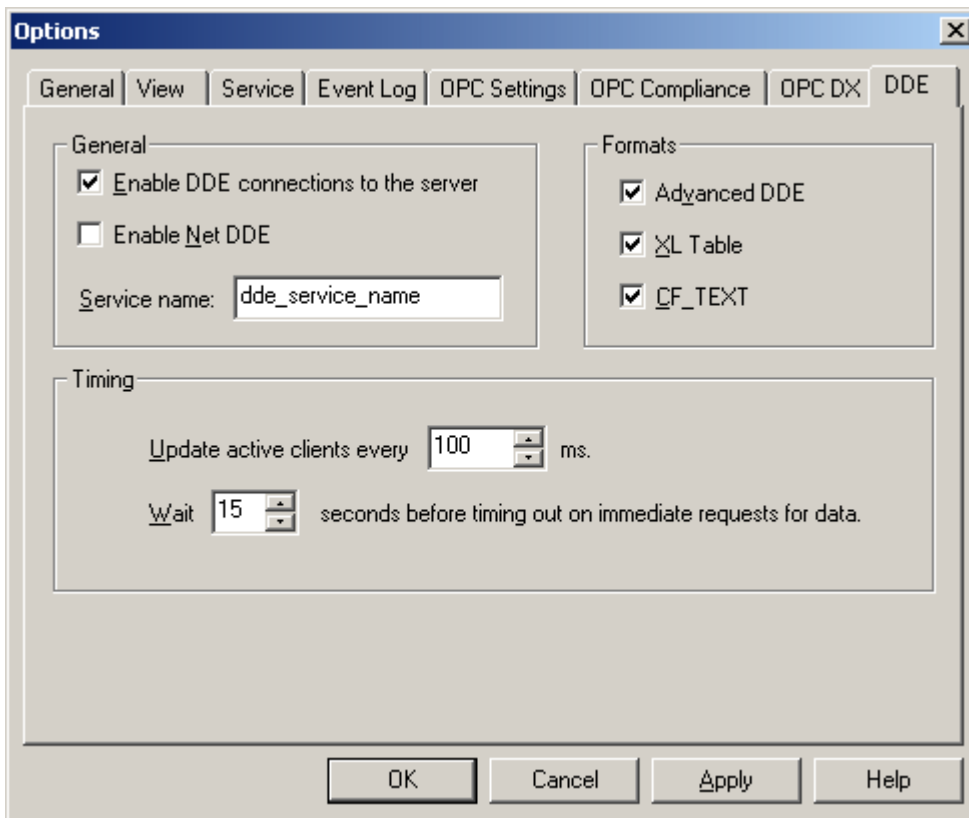
Note: FastDDE, SuiteLink, FactorySuite, InTouch and Wonderware are all Trademarks of Wonderware Corporation.

DDE Options

While the server is first and foremost an OPC server, there are still a number of applications that require the use of DDE (Dynamic Data Exchange) to share data. The server still provides access to DDE applications that support one of the DDE methods supported. The server supports three DDE formats, CF_Text, XL_Table, and AdvancedDDE. Formats CF_Text and XL_Table are standard DDE formats developed by Microsoft for use with all DDE aware applications. AdvancedDDE is a high performance format supported by a number of client applications specific to the industrial market.

DDE Option Dialog

The DDE Option dialog can be reached by selecting from the **Tools | Options** menu in the server. This dialog allows a great deal of control in determining how the server provides DDE data. As seen in the following figure, there are a number of options that allow DDE operation to be tailored to the needs of the application.



Enable DDE connections to the server

This check box allows the DDE server portion of the server to be turned either on or off. If DDE operation is turned off, the server will not respond to any request for DDE data. If you intend to use the server only as an OPC server you may want to disable DDE operation. This can increase the security of your data and improve the overall performance of the server. **See Also:** [Using DDE](#).

Enable Net DDE

This check box allows the use of Microsoft's Net DDE services to be disabled. If you intend to use the server only with local DDE client applications Net DDE should be left disabled which is the default. Starting the Net DDE services can be a time-consuming process that can slow the startup of the server. If you do need to use Net DDE, enabling it here will cause the server to automatically register its share names and start the Net DDE service manager. DDE shares will also be removed when the server shuts down. **See Also:** [Using Net DDE](#).

Service Name

This field is used to change how the server appears as an application name to DDE clients. Initially this name will be set to allow compatibility with the previous versions of the server. If however you need to replace an existing DDE server you can change the service name of the server to match the DDE server being replaced. The service name allows a string of 1 to 32 characters to be entered.

Formats

The server is used to configure what format of DDE you want to provide to client applications. Normally all three formats are enabled by default. If, however, you are experiencing problems connecting a DDE client application to the server, each of the DDE formats can be disabled to allow for isolating a specific format for testing purposes. Keep in mind that every DDE-aware application must at a minimum support CF_Text.

Update active clients

This interval setting is used to allow DDE data to be batched up for transfer to client applications. When using a DDE format like AdvancedDDE performance gains only come when large blocks of server data can be sent in a single DDE response. To improve the ability of the server to gather a large block of data, the update timer can be set to allow a pool of new data to accumulate before being sent to a client application. The valid range of the update timer is 20 - 60000 milliseconds. The default is 100 milliseconds.

Wait

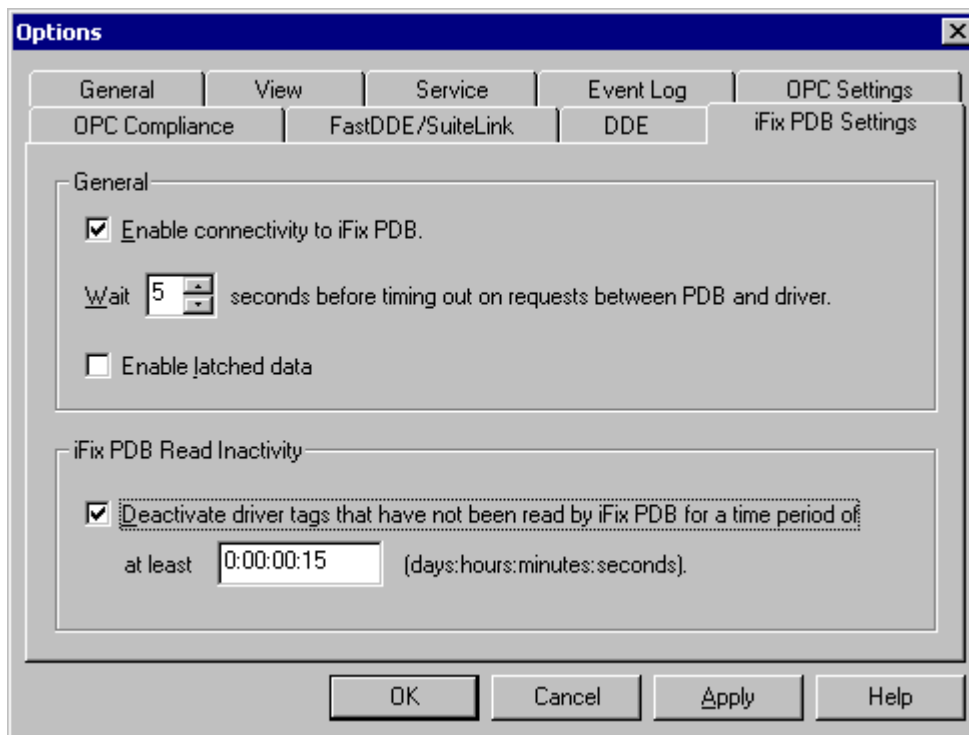
The wait time allows a timeout to be configured for the completion of DDE request. If a DDE client request either a read or write operation on the server and that request can not be completed within the specified timeout an error will be returned to the DDE Client. The valid range is 1 - 30 seconds. The default is 15 seconds.

Note: Changes made to the DDE options will only take effect upon the next restart of the server.

iFIX PDB Options

The iFIX PDB Settings tab contains fields that enable users to adjust the behavior between the processing of the iFIX process database (PDB) tags and the server tags. This tabs can be accessed by clicking **Tools | Options** menu. The iFIX PDB Settings tab displays in the Options dialog box only if iFIX is installed on the computer.

The following graphic shows the iFIX PDB Settings tab.



The following fields are available in the iFIX PDB Settings tab. It is recommended that the default values be used for each of these fields. Settings must meet the requirements of the application being used

The **Enable Intellution iFIX PDB connections** option is used to turn support of the IFIX PDB interface On or Off. This setting is disabled by default.

Important: If iFIX PDB operation is turned off (disabled), the server will not respond to any request for data by IFIX PDB. If you intend to use the server only as an OPC server, you may want to disable Intellution iFIX PDB operation. By doing so, you can increase the security of your data and improve the overall performance of the server.

Timing - Wait xx seconds before timing out on requests between PDB and Driver – The time you set here represents the amount of time the iFix PDB will wait for a response from an add/remove/read/write request before timing out. If the iFix PDB times out, it will fail the request on behalf of the server. This timeout can occur if the server is busy processing other requests, or if iFix PDB has lost communications with the server. In the case of lost communications, the iFix PDB will automatically re-establish communications with the server so that successive timeouts do not occur.

Valid Range	Default Value
-------------	---------------

5 to 60 seconds

5 seconds

Enable latched data

Normally, data links in your iFix application will display a series of question marks "???" if a communication failure has occurred. In some cases, users may want to have a value displayed at all times, such as when reports are generated. By enabling latched data, the last value successfully read will be preserved on the screen. This feature is not enabled (unchecked) by default. The Enable Latched Data setting in the server is located in **Tools | Options** under the iFix PDB tab.

Note: Data latching is not supported for AR and DR blocks.

iFIX PDB Read Inactivity

The server can automatically deactivate tags that have not been read by iFix for the time period given here. By doing so, unnecessary polling of the process hardware can be reduced.

When iFIX PDB Read Inactivity feature is enabled, the server will look through its list of tags every 15 seconds, and deactivate any that are idle. A tag is considered idle if iFix has not performed a read request of this tag for at least the time period specified here. Since the server checks for idle tags on a 15 second cycle, a tag may not get set inactive at precisely this time from its last read; it could be up to 15 seconds longer depending on when the last read occurred in the check cycle. If iFix requests data from a tag that has been previously deactivated, the server will reactivate the tag and resume polling the hardware.

By default, this feature is disabled upon install of the driver. However, once this feature is enabled, it becomes applied to all projects. You may specify an idle time of up to 6:23:59:59 (1 week).

Caution: This feature is meant to be used with register tags only and can cause non-register tags to go off scan. To avoid this situation when using this feature, be sure to set the inactivity timer greater than the longest scan time configured in the iFix database.

Format	Valid Range	Default Value
[days:hours:minutes:seconds]	0:00:00:15 to 6:23:59:59	0:00:00:15 (15 seconds)

The time period can also be specified in seconds. For example, you could enter "62". The next time you bring up the page, you will see 0:00:01:02.

Examples

20 seconds	0:00:00:20 or 20
1 minute	0:00:01:00 or 60
1 hour and 30 minutes	0:01:30:00 or 5400
2 days	2:00:00:00

Creating Datablocks Inside FIX Applications

You do not have to use the server to define **static** tag. In fact, it is recommended that **dynamic** tags be used in FIX Database Manager since iFIX PDB does not support server tag browsing.

Before adding server items to iFIX Database Manager, complete the following prerequisites:

- Create and configure the server project adding channel, device (and item if you are using static tags)
- Know the three-letter acronym for the server. For our server/driver, the acronym is **IDS** (Industrial Data Server).
- Configure the server in iFIX SCADA Configuration (look for the acronym **IDS**).

Refer to the next section for details on entering driver data and datablock addresses in Database Manager.

To enter driver specifications for a database block in iFIX Database Manager

1. Select **Add** from the Blocks menu in the iFIX Database Manager to add a database block. Database Manager prompts you to select the type of database block.
2. Select the type of block and click **OK**. The block's dialog box appears as shown below.

Database Block Dialog Box

The screenshot shows the 'Analog Input' dialog box with the following fields and values:

- Tag Name:** (empty)
- Description:** (empty)
- Previous:** (empty)
- Next:** (empty)
- Driver:** SIM Simulation Driver
- I/O Address:** 0
- Signal Conditioning:** (empty)
- Hardware Options:** (empty)
- Process by Exception:**
- Scan Time:** 1
- Phase At:** (empty)
- Low Limit:** 0.00
- High Limit:** 100.00
- Units:** (empty)

3. Enter a name in the Tag Name field.

4. Complete the driver fields with the appropriate information for your driver.

I/O Driver Fields

The screenshot shows the 'I/O Driver Fields' section with the following values:

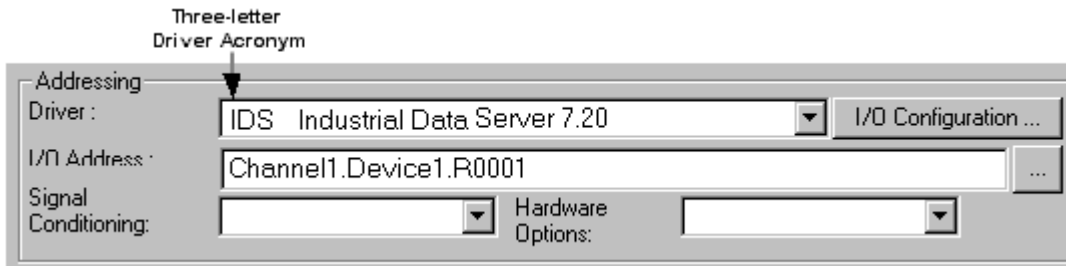
- Driver:** IDS Industrial Data Server 7.20
- I/O Address:** Channel1.Device1.R0001
- Signal Conditioning:** (empty)
- Hardware Options:** (empty)

Note 1: For more information on available signal conditioning options, refer to [iFix Signal Conditioning Options](#).

Note 2: The Hardware Options field is not used for this driver.

Specifying the I/O Driver in iFIX Database Manager

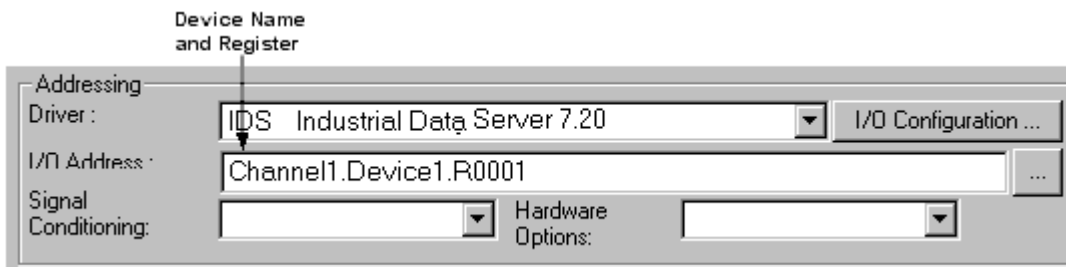
The Driver field in Database Manager identifies the I/O driver that the database block accesses. This field accepts your driver's three-letter acronym. The drop-down menu for this field provides the same Configured I/O Driver list as in the SCADA Configuration dialog box of the System Configuration Utility (SCU). For our server, enter **IDS** in this field.



Note: The server must appear in iFix's SCU's Configured I/O Driver list box in order for Database Manager to recognize the acronym that was entered.

Specifying I/O Addresses in iFix Database Manager

Specify the data block address that the database accesses in the Database Manager I/O Address field. Server I/O Addresses typically consist of the name of the channel, device and tag name or address. The I/O address is specific to the driver. This field is not case sensitive.



For this example we use the dynamic addressing format where R0001 is the actual register address in our device. The I/O address for the driver has the following format:

CHANNEL_NAME.DEVICE_NAME.DEVICE_ADDRESS

Where:

CHANNEL_NAME	Is the protocol or driver used in the server project. This name must match the channel name in the server configuration.
DEVICE_NAME	Is the PLC or other hardware that the server communicates with. This name must match the device name for the specified channel in the server configuration.
DEVICE_ADDRESS OR TAG_NAME	Is an address within the PLC or other hardware device that the server communicates with (dynamic) or is an actual tag name specified in the server (static).
	Note: If tags are static, you must include the full path to the tag name.

Project Startup for iFix

In order to have the startup behavior required by iFix, the OPC Server must begin polling immediately upon connection with iFix. (Other clients initiate polling some time after a connection is made.) To accomplish this, the server maintains a list of validated item ID's that have been referenced by iFix. This data is accumulated as blocks are created in iFix, and saved in an ini file. The ini file will be created by the server and given a name based on the current server project file name (opf file). The ini file will be placed in the same folder as the project file. This is not to be confused with the ServerMain.ini file which contains server settings that are used regardless of project file or client. Whenever iFix makes

its initial connection with the server, all items listed in the ini file are set active immediately.

Depending on the application, it may not be important that the server reads all items before SAC processing begins. Users may choose to remove certain items from the ini file or delete the ini file altogether in these cases. If AO or DO blocks, which are read by SAC one time only at startup, are being used it is essential that these items be included in the ini file. Also, if users have configured alarms connected with data quality, the relevant items should be in the ini file to ensure that the data has been read, and therefore of good quality, upon SAC startup.

This server feature was added after the initial release of the iFix interface. To create the ini file for an existing iFix project, export the PDB database from the iFix Database Manager, and import it back. Users may select "Yes to all" when the "Confirm tag replacement" message box comes up.

Please note that entries in the ini file are not removed when you delete the corresponding block(s) in your iFix project, or change the I/O Address of the block(s). Users may find it necessary to manually edit the ini file to remove items that are no longer used, or to recreate the ini file upon completing your PDB database. Reconstruction is the recommended practice. To recreate the file, simply delete it, then export and import your PDB database.

If an item ID listed in the ini file is ever found to be invalid, as would be the case if a static tag in the server is removed, the entry will automatically be removed the next time iFix is connected.

When manually editing the ini file, make sure the **Count** record accordingly.

Important: Users must configure iFix to delay the start of SAC processing long enough for the server to complete its initial reads of all items listed in the ini file. The default delay of 8 seconds should be sufficient in most situations, but may need to be significantly longer in some cases. You may change this delay by going into **System Configuration**, and selecting **Task Configuration**. Select "WSACTASK.EXE" and add the command line argument "Dseconds", where "seconds" is the number of seconds that will delay SAC processing. Note that command line arguments for WSACTASK.EXE must **not** be preceded with the minus sign "-".

Note: In addition to the SAC delay time, you should make sure that the server's "iFix PDB Read Inactivity timeout", described above, is set such that tags will not be deactivated before SAC makes its first read of the data.

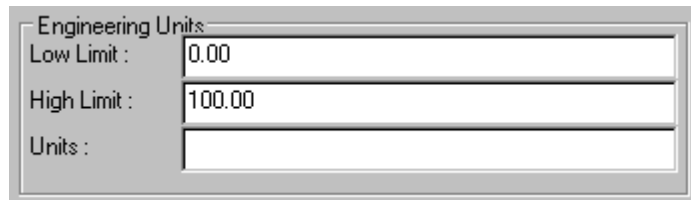
iFix Signal Conditioning Options

The IDS driver may apply signal conditioning directly to the data. Signal conditioning options are configured for each block defined in the FIX Database Manager.

1. To do so, specify the desired algorithm from the **Signal Conditioning field**. For no signal conditioning, select **None**.



2. Next, specify the **EGU (Engineering Unit)** range for the conditioned data.



3. The following signal conditioning options are available through the FIX Database Manager:

[3BCD](#)

[4BCD](#)[8AL](#)[8BN](#)[12AL](#)[12BN](#)[13AL](#)[13BN](#)[14AL](#)[14BN](#)[15AL](#)[15BN](#)[20P](#)[TNON](#)

None: Linear and logarithmic scaling is available through the server for [static tags](#) only. For more information, refer to [Scaling](#).

3BCD Signal Conditioning

Description	3-digit Binary Coded Decimal (BCD) value.
Input Range	0 - 999.
Scaling	Scales 3-digit Binary Coded Decimal values to the database block's EGU range.
Read Algorithm	Reads from a 3-digit BCD register. The Raw_value is then separated into three nibbles (4 bits) prior to scaling the value. Each nibble is examined for a value greater than 9 (A-F hex). If a hexadecimal value between A and F is found, a range alarm is generated, indicating the value is not within BCD range. Otherwise, the value is scaled with the following algorithm: Result=((Raw_value/999) * Span_egu) + Lo_egu.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 3-digit BCD register using the following algorithm: Result=(((InputData - Lo_egu) / Span_egu) * 999 + .5).
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

4BCD Signal Conditioning

Description	4-digit Binary Coded Decimal (BCD) value.
Input Range	0 - 9999.
Scaling	Scales 4-digit Binary Coded Decimal values to the database block's EGU range.
Read Algorithm	Reads from a 4-digit BCD register. The Raw_value is then separated into four nibbles (4 bits) prior to scaling the value. Each nibble is examined for a value greater than 9 (A-F hex). If a hexadecimal value between A and F is found, a range alarm is generated, indicating the value is not within BCD range. Otherwise, the value is scaled with the following algorithm: Result=((Raw_value/9999) * Span_egu) + Lo_egu.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values.

	Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 4-digit BCD register using the following algorithm: Result= $((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 9999 + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

8AL Signal Conditioning

Description	8-bit binary number.
Input Range	0 - 255.
Scaling	Scales 8-bit binary values to the database block's EGU range.
Read Algorithm	Reads from a 16-bit register using the same algorithm as 8BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{Raw_value}/255) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the same algorithm as 8BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{InputData} - \text{Lo_egu})/\text{Span_egu}) * 255) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

8BN Signal Conditioning

Description	8-bit binary number.
Input Range	0 - 255.
Scaling	Scales 8-bit binary values to the database block's EGU range. Ignores the most significant byte.
Read Algorithm	Reads from a 16-bit register using the following algorithm: Result = $((\text{Raw_value}/255) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to an 8-bit register using the following algorithm: Result = $((\text{InputData} - \text{Lo_egu})/\text{Span_egu}) * 255) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

12AL Signal Conditioning

Description	12-bit binary number.
Input Range	0 - 4095.
Scaling	Scales 12-bit binary values to the database block's EGU range.

Read Algorithm	Reads from a 16-bit register using the same algorithm as 12BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{Raw_value}/4095) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the same algorithm as 12BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{InputData} - \text{Lo_egu})/\text{Span_egu}) * 4095) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

12BN Signal Conditioning

Description	12-bit binary number.
Input Range	0 - 4095.
Scaling	Scales 12-bit binary values to the database block's EGU range. Ignores the most significant nibble (4-bits). Out of range value are treated as 12-bit values. For example, 4096 is treated as 0 because the four most significant bits are ignored.
Read Algorithm	Reads from a 16-bit register using the following algorithm: Result = $((\text{Raw_value}/4095) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the following algorithm: Result = $((\text{InputData} - \text{Lo_egu})/\text{Span_egu}) * 4095) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

13AL Signal Conditioning

Description	13-bit binary number.
Input Range	0 - 8191.
Scaling	Scales 13-bit binary values to the database block's EGU range.
Read Algorithm	Reads from a 16-bit register using the same algorithm as 13BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{Raw_value}/8191) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the same algorithm as 13BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK.

	$\text{Result} = (((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 8191) + .5.$
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

13BN Signal Conditioning

Description	13-bit binary number.
Input Range	0 - 8191.
Scaling	Scales 13-bit binary values to the database block's EGU range. Ignores the most significant 3 bits.
Read Algorithm	Reads from a 16-bit register using the following algorithm: $\text{Result} = ((\text{Raw_value} / 8191) * \text{Span_egu}) + \text{Lo_egu}.$
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the following algorithm: $\text{Result} = (((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 8191) + .5.$
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

14AL Signal Conditioning

Description	14-bit binary number.
Input Range	0 - 16383.
Scaling	Scales 14-bit binary values to the database block's EGU range.
Read Algorithm	Reads from a 16-bit register using the same algorithm as 14BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. $\text{Result} = ((\text{Raw_value} / 16383) * \text{Span_egu}) + \text{Lo_egu}.$
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the same algorithm as 14BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. $\text{Result} = (((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 16383) + .5.$
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

14BN Signal Conditioning

Description	14-bit binary number.
Input Range	0 - 16383.
Scaling	Scales 14-bit binary values to the database block's EGU range. Ignores the most significant 2 bits.
Read Algorithm	Reads from a 16-bit register using the following algorithm: $\text{Result} = ((\text{Raw_value} / 16383) * \text{Span_egu}) + \text{Lo_egu}.$

Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the following algorithm: Result= $((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 16383) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

15AL Signal Conditioning

Description	15-bit binary number.
Input Range	0 - 32767.
Scaling	Scales 15-bit binary values to the database block's EGU range.
Read Algorithm	Reads from a 16-bit register with alarming using the same algorithm as 15BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{Raw_value} / 32767) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register with alarming using the same algorithm as 15BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result= $((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 32767) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

15BN Signal Conditioning

Description	15-bit binary number.
Input Range	0 - 32767.
Scaling	Scales 15-bit binary values to the database block's EGU range. Ignores the most significant bit.
Read Algorithm	Reads from a 16-bit register using the following algorithm: Result = $((\text{Raw_value} / 32767) * \text{Span_egu}) + \text{Lo_egu}$.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the following algorithm: Result = $((\text{InputData} - \text{Lo_egu}) / \text{Span_egu}) * 32767) + .5$.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

20P Signal Conditioning

Description	6400 – 32000 Clamp.
--------------------	---------------------

Input Range	6400 – 32000.
Scaling	Scales binary values to the database block's EGU range. Clamps value to 6400 – 32000 range.
Read Algorithm	Reads from a 16-bit register using the following algorithm: Result =(((Raw_value - 6400)/25600) * Span_egu) + Lo_egu.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the following algorithm: Result =(((InputData - Lo_egu)/Span_egu) * 25600) + 6400.5.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

TNON Signal Conditioning

Description	0 – 32000 Clamp.
Input Range	0 – 32000.
Scaling	Scales binary values to the database block's EGU range. Clamps value to 0 – 32000 range.
Read Algorithm	Reads from a 16-bit register using the following algorithm: Result =((Raw_value/32000) * Span_egu) + Lo_egu.
Read Algorithm Variables	Lo_egu - the database block's low engineering value. Span_egu - the span of the engineering values. Raw_value - the value stored in the field device's register. Result - the scaled value stored in the database block.
Write Algorithm	Writes to a 16-bit register using the following algorithm: Result =(((InputData - Lo_egu)/Span_egu) * 32000) + .5.
Write Algorithm Variables	Lo_egu - the low engineering value. Span_egu - the span of the engineering values. InputData - the database block's current value. Result - the value sent to the process hardware.

License Transfer Instructions (Step 1)

The server uses a software-based licensing system with the goal of reducing unlicensed use of the server without undue hardship on legitimate users. We chose a software based protection system over a hardware-based key in response to customer feedback. One of the features normally available with hardware based protection systems is the ability to move a license from one machine to another. With a hardware key this is usually just a matter of removing the hardware key from one machine and placing it on the new machine. Software based systems in many cases have attempted to achieve this same transportable license through the use of a Key disk. The problem with many Key disks is that they are created using a specific format that prevents normal copying of the disk. While this may provide some additional level of security for the software vendor it usually results in problems for the customer when the Key disk becomes corrupt.

The software based protection system of the server provides a mechanism for transferring individual driver or plug-in licenses from one machine to another. This is very useful for developers and integrators who need to design and test on their desktop or laptop systems before the final project is installed on the end user machine. The license transfer system does not require a special key disk to be used. Instead of a key disk you are required to perform a little bit of the old "Sneaker Net" to move selected licenses from one PC to another. While the server's transfer utility uses removable media such as a USB key or floppy disk, there is no special formatting required for the disk and it can be copied if needed.

Note: There are six steps required to move a license from one machine to another.

Caution: The process of transferring a license from one machine to another must be completed once the license has been removed from the source machine. If the License Transfer process is canceled after the license has been removed, it will become corrupt and can not be installed on either the source or target PC. Make sure all the steps described can be completed before beginning the license transfer process.

License Transfer Agreement (Step 2)

When moving a driver or plug-in license from one machine to another, users should refamiliarize themselves with the software license agreement. If accepting the agreement, simply check the accept selection. This invokes [Step 3](#) of the transfer utility.

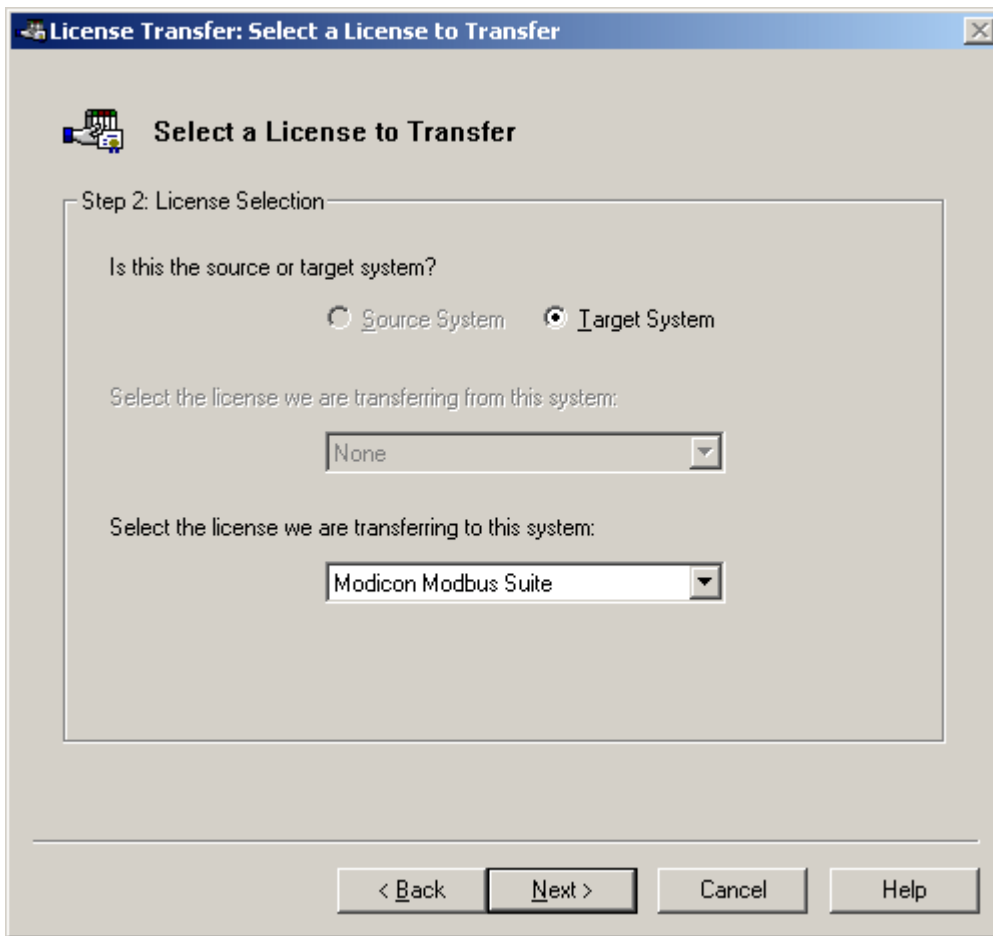
Caution: The process of transferring a license from one machine to another must be completed once the license has been removed from the source machine. If the License Transfer process is canceled after the license has been removed, it will become corrupt and cannot be installed on either the source or target PC. Make sure all the steps described can be completed before beginning the license transfer process.

License Transfer Selection (Step 3)

There are several basic requirements for transferring a license from one machine to another. The first requirement for transferring a license from one machine to another is the driver or plug-in. The desired driver or plug-in must be installed on both the source and target machines. The second requirement is a valid license for the driver or plug-in must exist on the source machine. The final requirement is that both source and target machines must have a compatible removable media drive (e.g., Floppy Disk Drive, USB Key).

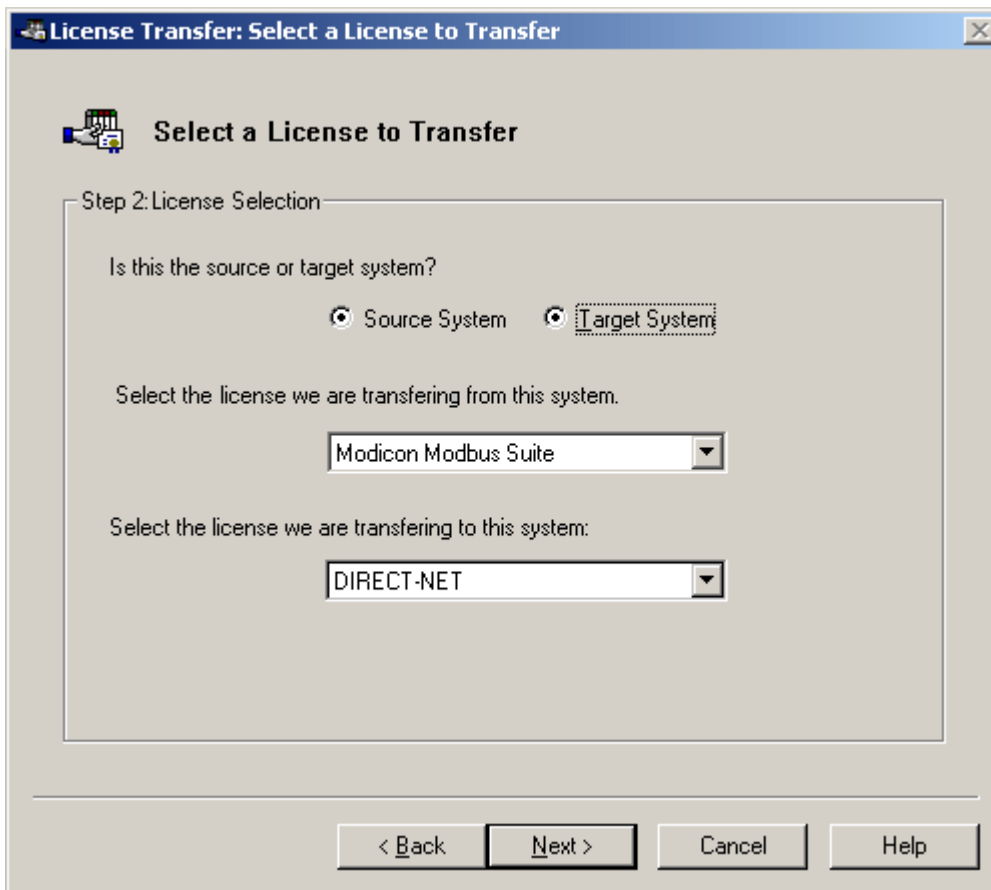
Note: If using floppy diskettes to transfer the license, it is essential to use a clean, newly formatted diskette before beginning the License transfer process. Also, removing the diskette from the floppy drive during either the target disk preparation or actual license transfer portion on the source machine may cause an error that will result in the loss of the license information.

If all of these requirements are met, begin Step 2. There are two possible paths that can be taken at this point depending on the nature of the system. If the machine being used has no licensed drivers or plug-ins, the only selection will be to make this system the target. The following figure shows how the transfer dialog appears in target only mode.



In this mode, the only choice is a target system. The drivers and plug-ins that are installed on this target system will dictate which items are available in the drop down menu. Once the desired item has been selected, proceed to [Step 4](#). Choose an item that is licensed on the source machine.

If the system being used contains both licensed and unlicensed drivers or plug-ins, choose to be either a source or target system. When this is the case, the figure below shows how the transfer dialog may appear when both source and target modes can be selected.

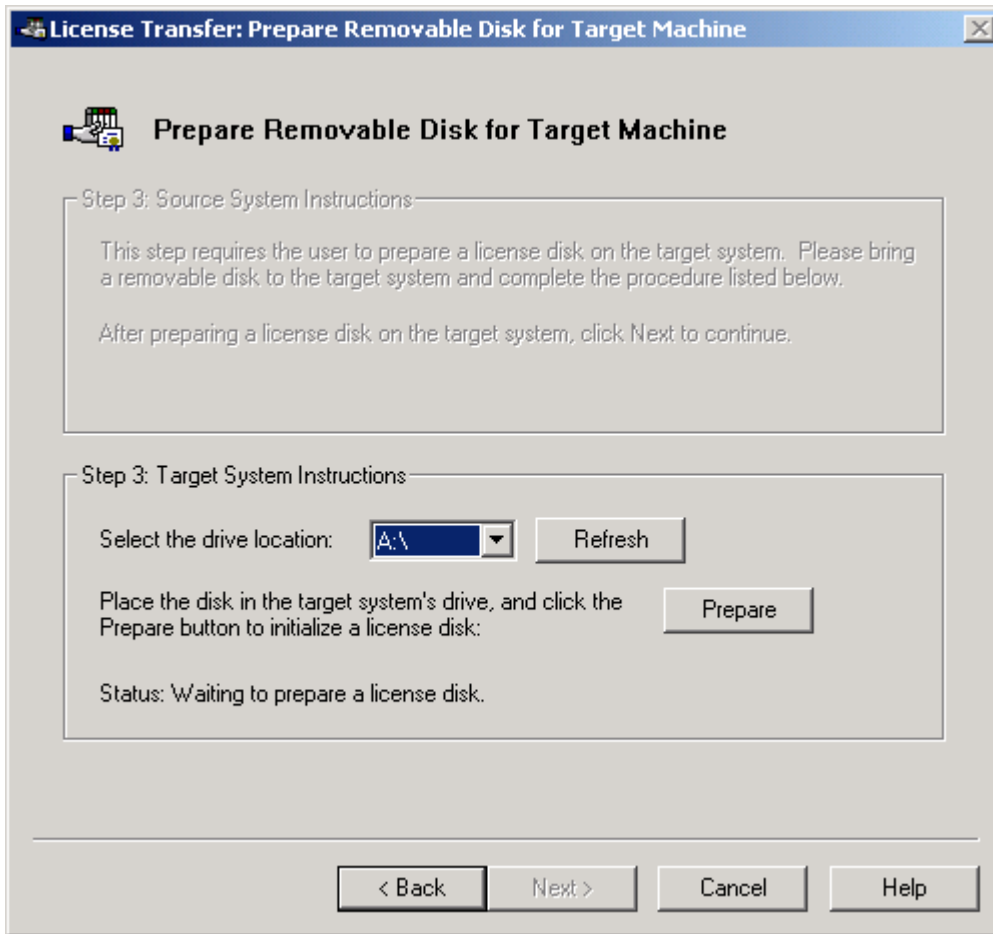


If the PC is chosen to be the source machine, select a driver or plug-in from the licensed drop-down menu.

Caution: The process of transferring a license from one machine to another must be completed once the license has been removed from the source machine. If the License Transfer process is canceled after the license has been removed, it will become corrupt and cannot be installed on either the source or target PC. Make sure all the steps described can be completed before beginning the license transfer process.

License Transfer Preparing Removable Disk (Step 4)

In Step 3, users were required to prepare a removable disk for license transfer. Depending on the machine and mode being used, users will then be asked to insert the formatted removable disk either on this machine or on the second machine. The follow dialog assumes that the machine is the target.



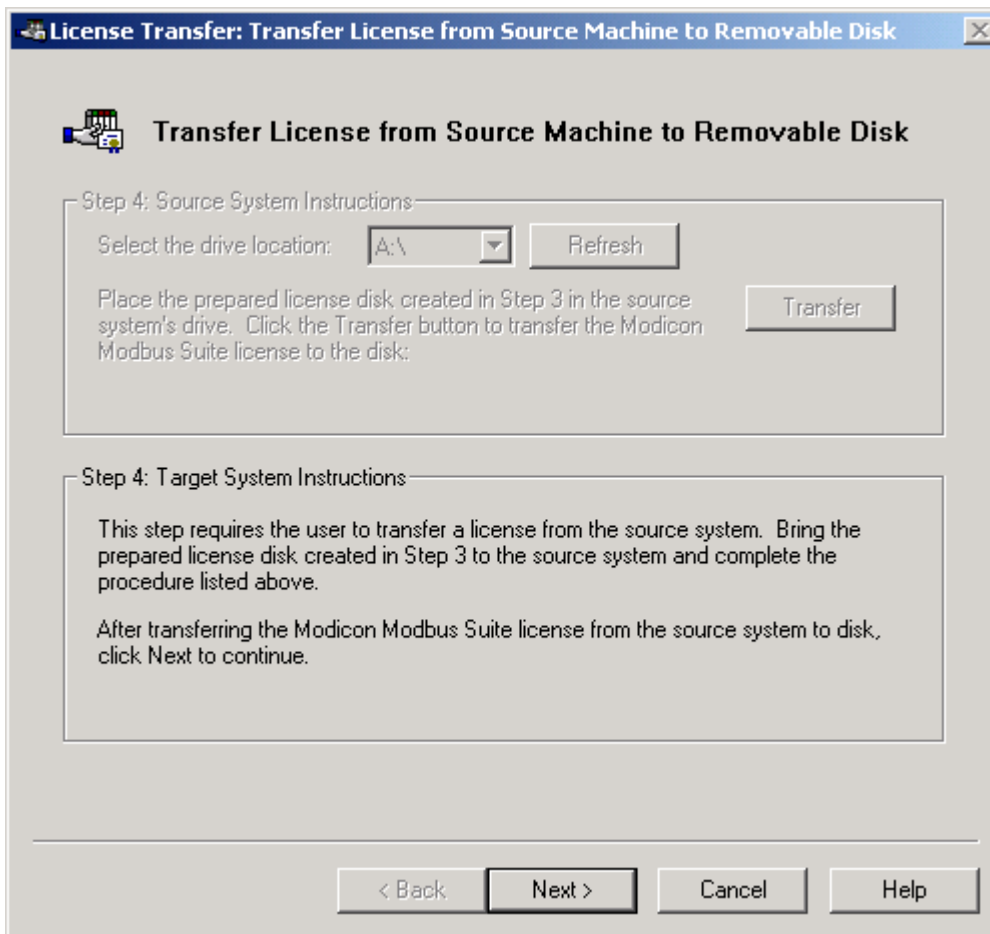
The diskette preparation done at this point produces the media required to move a license from the source machine. The target system is required to produce the diskette needed to move a license from the source machine. If the diskette preparation is interrupted, a license will not be able to be transferred from the source machine.

Note: Once the diskette is prepared (or if this is the source machine) proceed to [Step 5](#).

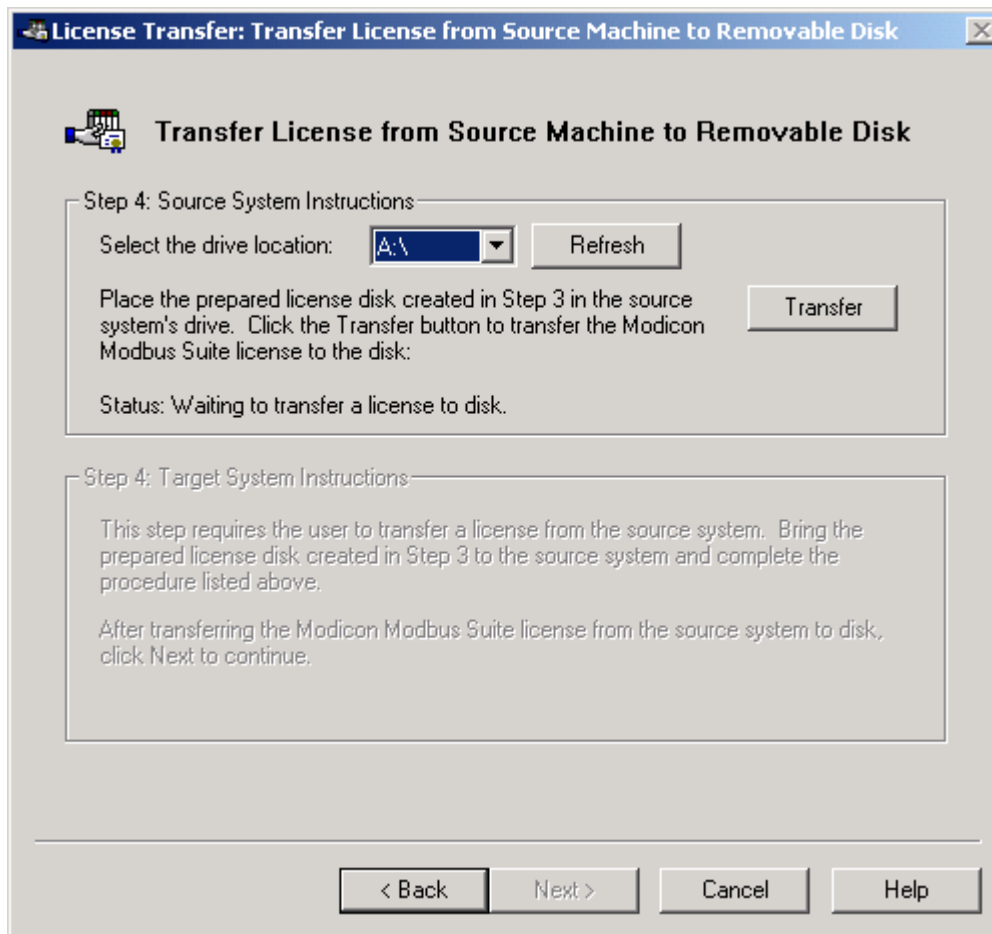
Caution: The process of transferring a license from one machine to another must be completed once the license has been removed from the source machine. If the License Transfer process is canceled after the license has been removed, it will become corrupt and cannot be installed on either the source or target PC. Make sure all the steps described can be completed before beginning the license transfer process.

License Transfer Moving the License (Step5)

At this stage, the removable media should now be prepared to move a license from the source machine to the target machine. Since these steps are from the perspective of the target machine, the following dialog should be visible on the target system.



1. Take the removable media (that prepared in Step 3) over to the source machine. The dialog on the source machine should appear as shown below.



2. Insert the removable media (prepared in Step 2 on the target machine) into the removable media port of the source machine and then press **Transfer**. The transfer utility will now transfer the selected driver license to the removable disk.

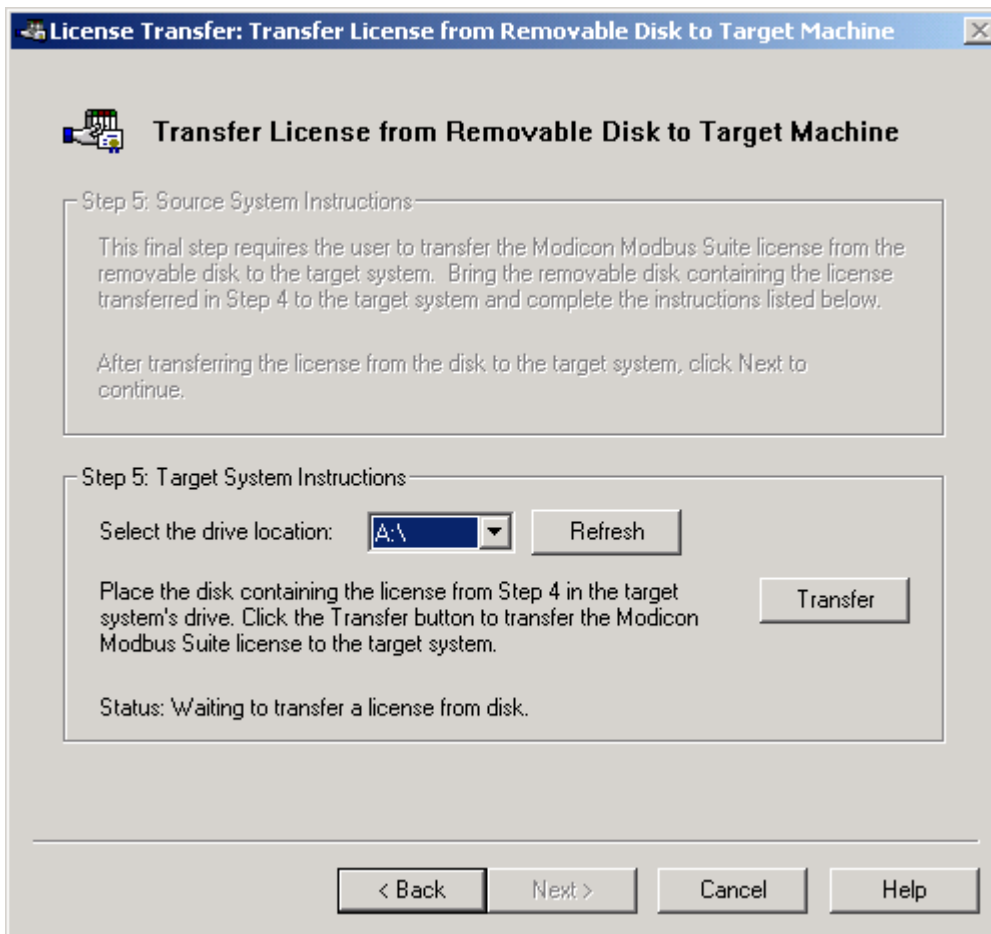
Caution: Removing the media from the PC before the media's busy indicator light has been turned off may cause the license information to be lost. Allow all disk operations to complete before attempting to remove any disks from the PC.

Note: Once the license has been transferred to the disk media, proceed to [Step 6](#).

Caution: The process of transferring a license from one machine to another must be completed once the license has been removed from the source machine. If the License Transfer process is canceled after the license has been removed, it will become corrupt and cannot be installed on either the source or target PC. Make sure all the steps described can be completed before beginning the license transfer process.

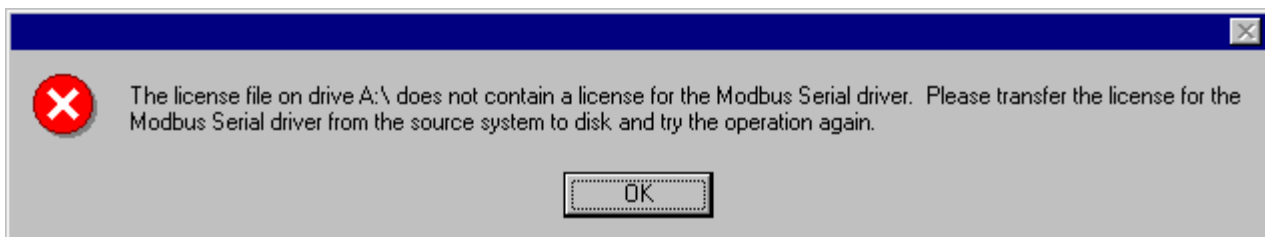
License Transfer License Installation (Step6)

At this point, the removable disk should contain a license that is ready to be installed on the target system. Place the disk in the selected drive and press **Transfer**. Do not attempt to remove the disk from the drive after, as it may cause an error that results in the loss of the license information.



Once the transfer is complete, click **Next** in order to proceed to the [completion screen](#). If all has gone well, the driver license has been successfully transferred to the target machine.

Note: If the license information failed to transfer from the source machine to the prepared disk, the following dialog will be invoked.



Once the transfer of a license has been completed successfully, the disk used no longer contains any license information and does not need to be maintained.

Caution: The process of transferring a license from one machine to another must be completed once the license has been removed from the source machine. If the License Transfer process is canceled after the license has been removed, it will become corrupt and cannot be installed on either the source or target PC. Make sure all the steps described can be completed before beginning the license transfer process.

License Transfer Completion (Step7)

The license has been transferred successfully.

Licensing Overview

The license and unlock procedure for full time use of a driver or plug-in is a simple two step process.

Step 1: License a Driver or Plug-in

The first step to license a driver or plug-in requires that the appropriate product license number be obtained from the vendor or authorized distributor and then be entered in the server application.

Step2: Unlock a License Driver or Plug-in

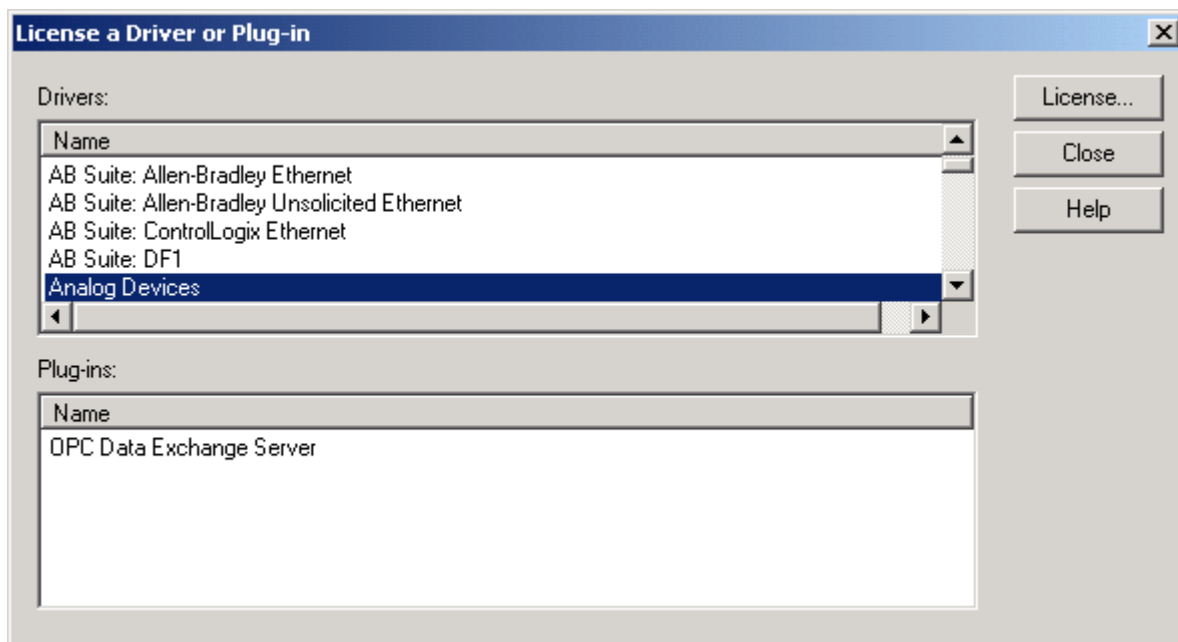
The second step is to unlock the license for full time use.

License a Driver or Plug-in

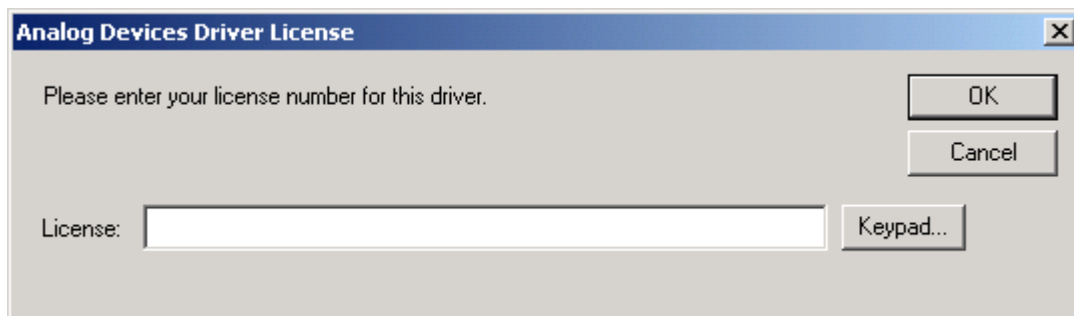
Step 1

To license a driver or plug-in, click **Help | License a Driver or Plug-in** from the server's menu bar. The invoked dialog will indicate whether there are currently unlicensed drivers or plug-ins that are available to be licensed.

Note: If the "License a Driver or Plug-in" selection from the server's menu bar is grayed out, that indicates that no drivers or plug-ins have been installed or they have already been licensed.



To license a driver or plug-in, select the desired item from the dialog's list and then click **License**. A new dialog will appear indicating the driver or plug-in that was selected and will contain a field for entering the license number.



A product license number can be obtained by contacting the vendor or an authorized distributor. Enter the product license number in the provided field. If a keyboard is unavailable, use the on screen keypad to enter the license

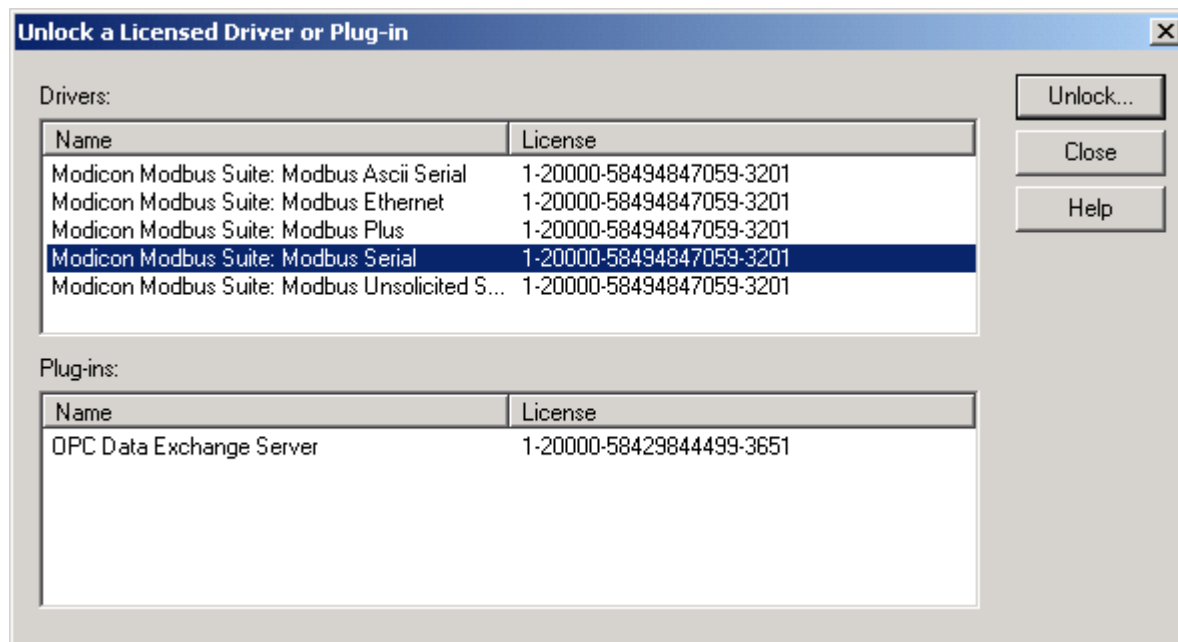
number. Once the product license number has been entered, the program will be running in a 10 day installation grace period in which time users must contact a qualified representative to unlock the driver or plug-in for full time use. Step 1 prepares a specific driver or plug-in to be unlocked for full-time, unrestricted use. A license must be unlocked to complete the licensing sequence for full time use.

See Also: [Step2: Unlock a Licensed Driver or Plug-in](#)

Unlock a Licensed Driver or Plug-in

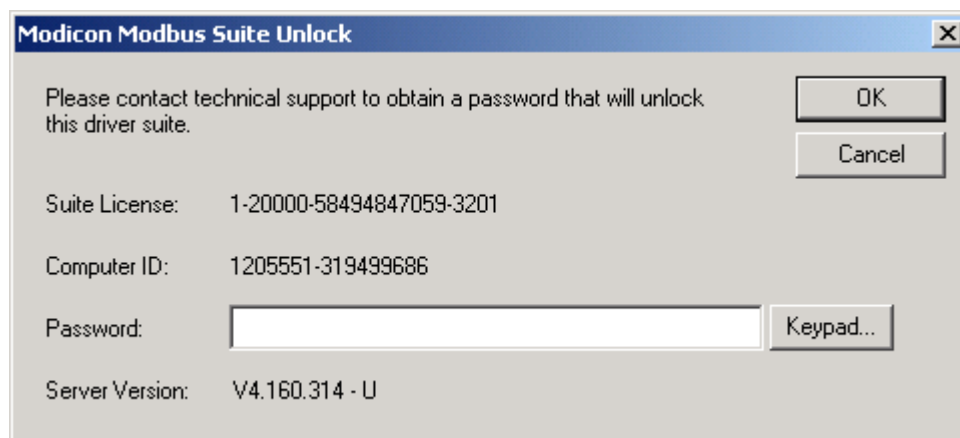
Step 2

Once a driver or plug-in has been licensed, it can then be unlocked for full-time use on the target PC. To perform an unlock, select **Help | Unlock a Licensed Driver or Plug-in** from the server's menu bar. If there are driver or plug-in licenses that have not yet been unlocked for full time use, the following dialog will appear.



Note: This option will only be enabled if a valid license is detected that has not already been unlocked.

A driver or plug-in can be unlocked by selecting it from the list of licensed drivers or plug-ins and then clicking **Unlock** button. This will display the unlock license dialog. The example below displays a driver suite unlock dialog.

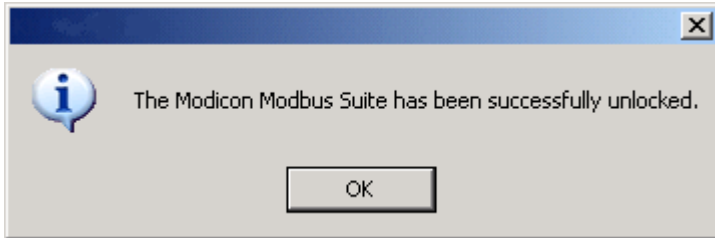


At this point, users should contact their vendor by phone, fax or email. The data presented on this dialog must be provided to the vendor to receive an unlock code. The License and Computer ID must be returned accurately to allow

the vendor to generate the appropriate password. Once the password has been obtained from the vendor, it can be entered in this dialog to fully unlock the driver or plug-in. If a keyboard is not available, the on screen keypad can be used to enter the code.

If the vendor cannot be contacted immediately, the driver may be run in its initial 10-day licensed run mode. The 10-day installation mode is designed to let users contact their vendors when it is convenient. If the vendor can not be reached immediately, simply hit the cancel button to start the 10-day installation grace period. A message will be placed in the server event log indicating that the 10-day installation grace period has begun.

When a driver or plug-in license has been successfully unlocked, the dialog will appear as shown below.



How Do I

The following list contains a sample of common questions posed by customers.

How Do I...

- [Configure the Server to Run as an NT Service](#)
- [Configure an Ethernet Connection for a Specific NIC Card](#)
- [Configure Ethernet Encapsulation](#)
- [Map Aliases](#)
- [Use DDE with the Server](#)
- [Use Net-DDE with the Server](#)
- [Process Array Data in a DDE client](#)

Index

- A -

access 114
Address Validation Popup 76
AdvancedDDE 121
alias 22
Alias Map 76
Alias Properties 77
answer 29
auto 29
auto-demotion 65
Automated OPC Tag Database Generation 66

- B -

Basic Server Components 43
baud 45
bits 45
browse 11, 13, 45, 55, 58
buy 141

- C -

CF_Text 121
channel 43, 45, 48
Channel Functions 44
Channel Properties - Comm. Parameters 45
Channel Properties - Ethernet Encapsulation 51
Channel Properties - Identification 45
Channel Properties - Manual RTS Flow Control 48
Channel Properties - Modem 48
Channel Properties - Network Interface 46
Channel Properties - Write Optimizations 49
comment 22
communications 45, 63
compatibility 76
compliance 116
connection 63
CSV delimiter 106
CSV Import and Export 21

- D -

data 114
DDE 4, 11, 22, 58, 76, 118, 121
DDE Options 121
debug 80
delay 48
Designing a Project 86
Designing a Project (Adding and Configuring a Channel) 87
Designing a Project (Adding and Configuring a Device) 91
Designing a Project (Adding Tag Scaling) 98
Designing a Project (Adding Tags to the Project) 94
Designing a Project (Running the Server) 87
Designing a Project (Saving and Testing the Project) 99
Designing a Project (Starting a New Project) 87
device 13, 43, 55, 56, 58
Device Functions 54
Device Properties - General 58
Device Properties - ID 58
Diagnostic Tags 11
Diagnostic Window 80
diagnostics 8, 11, 45, 80
dial 25, 28, 29
Dial Tag 28
DialNumber Tag 29
disconnect 29
disk 113
driver 45
Driver License 141
Driver Registration 142
DTR 45, 47
Dynamic Tags 71

- E -

edit 43
email 80
errors 11, 63, 80
ethernet 56
Ethernet Encapsulation 51, 60
event 42, 43
Event Log Print - Page Setup 42
Event Log Properties 113

events 113
exit 106

- F -

FastDDE 118
FastDDE & SuiteLink Options 118
file 43
flow 45, 47, 48
Flow Control RTS & DTS 47
fonts 42
foundation 114

- G -

General Options 106
groups 43

- H -

hangup 25, 29
Hangup Tag 29
How Do I 143

- I -

identification 45
iFIX PDB Options 123
iFix Signal Conditioning Options 127
inter-request delay 63
Introduction 4

- L -

LastEvent Tag 29
license 141, 142
License Transfer Utility Page 3 134
License Transfer Utility Page 4 136
License Transfer Utility Page 5 137
License Transfer Utility Page 6 139
License Transfer Utility Page 7 140
License Transfer Utility Page1 133
License Transfer Utility Page2 134
log 42, 113
logger 113

- M -

machine 22
manual 48
margins 42
menu 43
mode 25
Mode Tag 29
model 55, 58
modem 25, 28, 29, 30, 31, 45, 48
Modem Support 25
MRU 106

- N -

name 45
NetDDE 22
New Channel - Driver Page 45
New Channel - Name Page 45
New Channel - Summary 54
New Device - ID 56
New Device - Model 55
New Device - Name 55
New Device - Summary 69
New Device - Timeout 63
node 56, 58
noise 63

- O -

OPC 4, 8, 13, 45, 55, 106, 114, 116
OPC Compliancy Options 116
OPC Options 114
OPC Quick Client Launch 32
options 113, 114, 116
originate 29

- P -

page 42
parity 45
phone 25, 30, 31
Phone Number 32
Phonebook 31
PhoneNumber Tag 30
port 45

print 42
printer 42
Processing Array Data in a DDE Client 24
project 22
Project Properties 22
Project Startup for iFIX 123
protocol 80
purchase 141, 142

- R -

radio 48
register 141, 142
remote 22
retries 63
RTS 45, 47, 48

- S -

save 106
Scaling 8, 72
Server Features 8
service 121
Service Options 109
settings 116
shares 22
specification 114
Static Tags User Defined 72
station 56, 58
status 25, 30, 31, 43
Status Tag 30
stop 45
StringLastEvent Tag 31
StringStatus Tag 31
SuiteLink 118
support 80, 142
System Requirements 8
System Tags 13

- T -

tag 11, 28, 29, 30, 31
Tag Functions 69
Tag Group Properties 74
Tag Properties 69
Tag Selection 75
tags 8, 13, 25, 43, 80

tapi 30, 31
timeout 63, 114
Tools 43
topic 22, 76, 118

- U -

unlock 141, 142
user 8
User Manager 41
User Properties 41
Users 43, 113
Using NetDDE 22
USRobotics Quick Reference 33

- V -

view 43
View Options 108

- X -

XL_Table 121