kepware® kepserverex®

Table of Contents

KEPServerEX V6

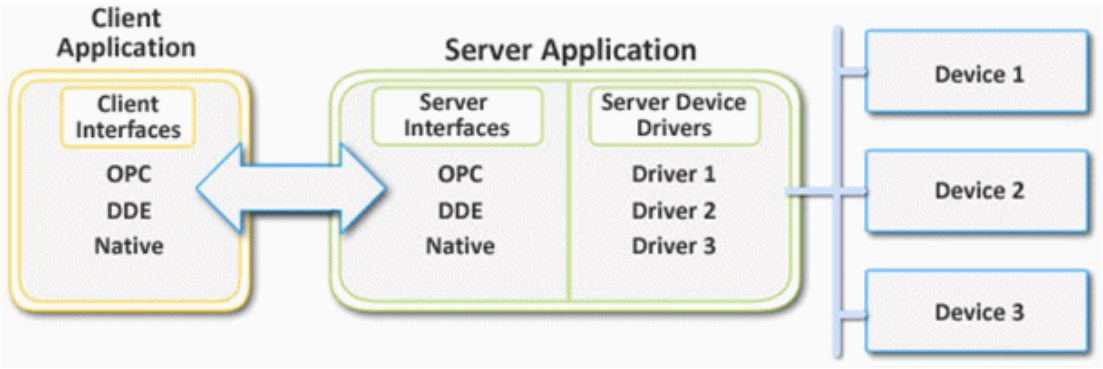
CONTENTS

For information regarding product licensing, refer to the License Utility help file. To access the help file through the server Configuration menu, click ***Help | Server Help | License Utility****. To access the help file through the server Administration menu, right-click on the KEPServerEX icon in the System Tray and select* ***Help | License Utility****.*

Introduction

Version 1.606

This software based server is designed for accurate communications, quick setup, and unmatched inter-operability between client applications, industrial devices, and systems. The server provides a wide range of plug-ins and device drivers and components that suit most communication needs. The plug-in design and single user interface provides consistent access from standards-based applications and non-standards-based applications with native interfaces.



System Requirements

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

This application supports the following Microsoft Windows operating systems:

- Windows 10 x64 (Pro and Enterprise Edition)[3]
- Windows 10 x86 (Pro and Enterprise Edition)
- Windows 8.1 x64 (Windows 8, Pro, and Enterprise Edition)[3]
- Windows 8.1 x86 (Windows 8, Pro, and Enterprise Edition)
- Windows 8 x64 (Windows 8, Pro, and Enterprise Edition)[3]
- Windows 8 x86 (Windows 8, Pro, and Enterprise Edition)
- Windows 7 x64 (Professional, Ultimate, and Enterprise Edition)[3]
- Windows 7 x86 (Professional, Ultimate, and Enterprise Edition)
- Windows Server 2019 x64[3,4]
- Windows Server 2016 x64[3,4]
- Windows Server 2012 x64 R2[3]
- Windows Server 2012 x64[3]
- Windows Server 2008 x64 R2[3]

Notes

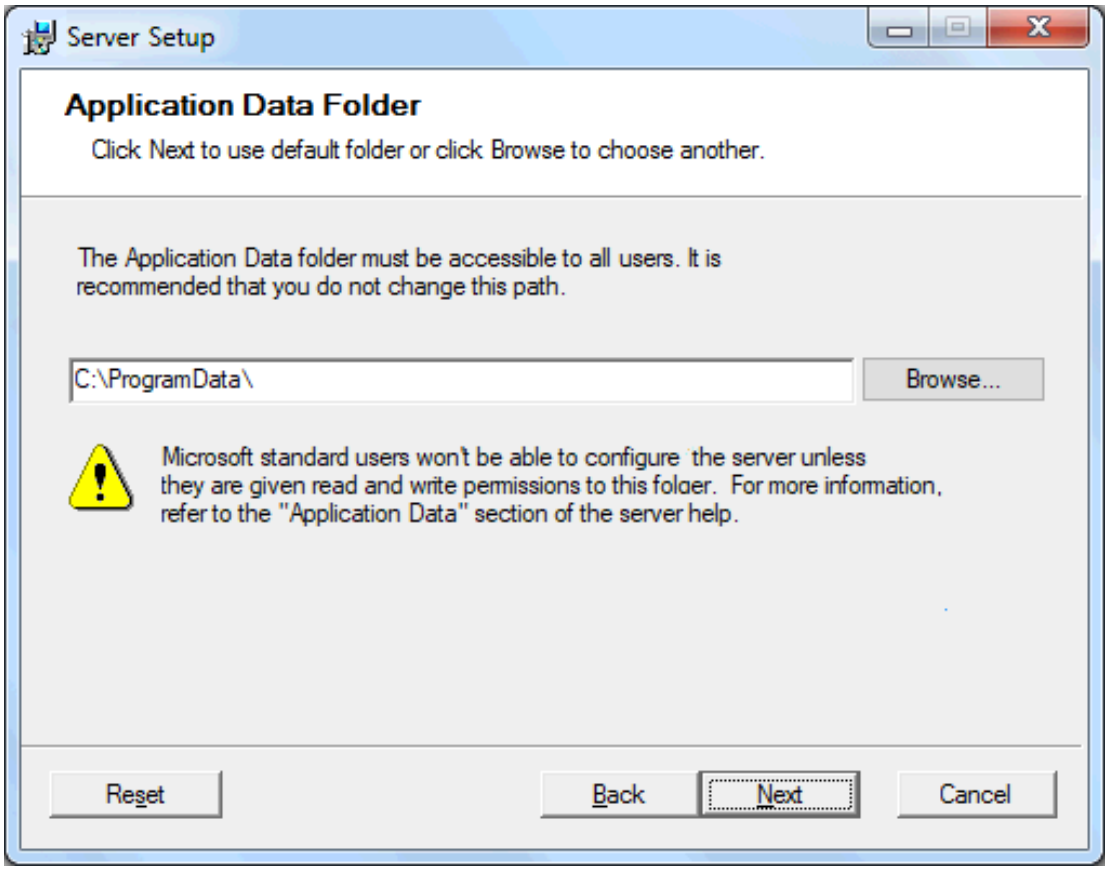
1. When installed on a 64-bit operating system, the application runs in a subsystem of Windows called WOW64 (Windows-on-Windows 64 bit). WOW64 is included on all 64-bit versions of Windows and is designed to make differences between the operating systems transparent to the user. WOW64 requires the following minimums:
    - 1 GHz Processor
    - 1 GB installed RAM (defer to the suggestion for the OS)
    - 180 MB available disk space
    - Ethernet Card
2. Verify the latest security updates are installed for the operating system.
3. Runs in the 32-bit compatibility mode.
4. Windows Server Core deployments are not supported.

*Additional resources are available on the Kepware and PTC websites. In particular, the following resources are helpful in planning stages: **KEPServerEX Install Guide**, **Secure Deployment Guide**. Contact a staff system engineer for guidance on requirements and recommendations for more complex systems.*

Application Data

Microsoft standard users must have the appropriate permissions on the Application Data directory. This folder contains files critical to the proper functioning of the server; permissions on this folder dictate which users are able to configure the product. By default, the server stores application data in C:\ProgramData\<server>. This setting is configured during installation and can only be changed by reinstalling the product. Permissions only need to be configured during a new installation as upgrades inherit the previously

configured Windows security settings. The dialog below shows where a new installation provides the opportunity to configure the location of the application data folder.



Microsoft standard users must be granted both read and write permissions to the folder and its contents. Execute permission is not required to run the server. The application does not provide tools to add permissions to this folder; they must be granted using Windows Explorer. Users who don't have permissions receive the following error when attempting to start the application: "This account does not have permission to run this application. Contact the system administrator".

The server does not modify the permissions of the configured folder; it inherits the default permissions configured at its location. The default (ProgramData) location inherits read-only permissions for the Users default Windows group. Read permissions alone are not sufficient to configure the product; however, they do potentially allow users who shouldn't have access the ability to read contents of the folder. By default, Windows administrators have the correct permissions.

To implement least privilege, follow these best practices:

- Only grant permissions to users or groups that require access to the application; do not grant permissions to all users. It is common for members of the Users default windows group to contain more users than should have access to the application.
- Remove the default permissions granted to users who shouldn't have access. For example, if the default directory is used, remove the inherited read-only permission granted to members of the "Users" default windows group. This should be done unless ALL users on the machine should be able to work with the product.
- Don't manage permissions with individual users or the "Users" default windows group. Instead, cre-

ate a custom user group and configure its permissions. Add users who should be granted permissions to that group.

Components

The server implements client/server architecture. The components include Configuration, Runtime, Administration, and Event Log.

Configuration

The Configuration is the client-user interface that is used to modify the runtime project. The Configuration can be launched by multiple users and supports remote Runtime configuration.

CSV Import and Export

This server supports the import and export of tag data in a Comma Separated Variable (CSV) file. When using CSV import and export, tags are created quickly in the desired application.

- *For more information, refer to **CSV Import and Export**.*

Runtime

The Runtime is the server component that starts as a service by default. Clients can connect to the runtime remotely or locally.

Administration

The Administration is used to view and/or modify settings and launch applications that pertain to user management and the server. By default, the Administration is started and sent to the System Tray when a user account logs onto the operating system.

Project

The Project file contains the channel, device, and tag definitions as well as preferences and other saved settings.

- *For more information, refer to **Designing a Project.***

Event Log

The Event Log service collects information, warning, error, and security events. These events are sent to the Configuration's Event Log window for viewing.

- *For more information, refer to **What is the Event Log?***

- ***See Also**: **Basic Server Components***

Process Modes

The Runtime process mode can be changed while the server is running; however, doing so while a client is connected interrupts the connection for a short period. The modes of operation are System Service and Interactive.

System Service

By default, the server is installed and runs as a service. When System Service is selected, the Runtime does not require user intervention and starts when the operating system opens. This provides user independent access to the server by the clients.

Interactive

When Interactive is selected, the Runtime remains stopped until a client attempts to connect to it. Once started, it runs until all clients have disconnected and then shuts down. The Runtime also shuts down if the user account logs off the operation system.

Note: The Runtime process mode may be changed to meet client applications' needs through the Administration settings dialogs.

System Service is required for the following conditions:

- When iFIX is required to run on an operating system while UAC is enabled.

Interactive is required for the following conditions:

- When a communication interface (such as DDE) must exchange information with the user desktop and the server is installed on Windows Vista, Windows Server 2008, or later operating systems.

See Also:
Settings - Runtime Process
How To... Allow Desktop Interactions

Interfaces and Connectivity

This communications server simultaneously supports the client / server technologies listed below. Client applications can use any of these technologies to access data from the server at the same time. For more information on a specific interface, select a link from the list below.

OPC DA
OPC AE
OPC UA
DDE
FastDDE/SuiteLink
iFIX Native Interfaces
ThingWorx Native Interface

OPC DA

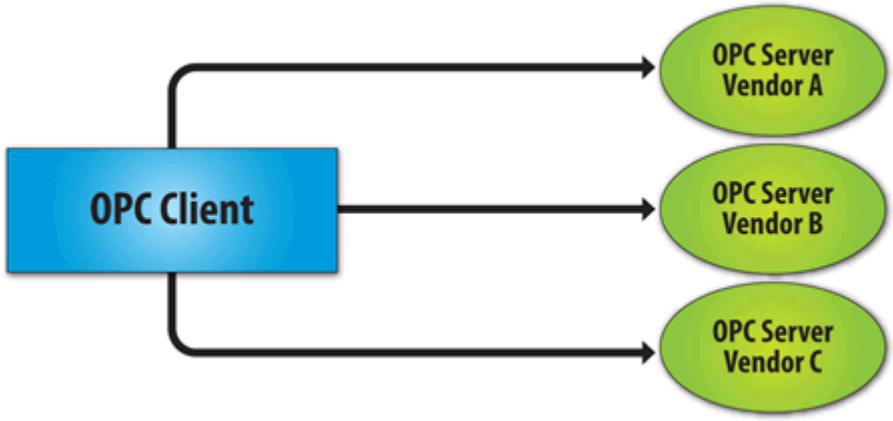
Supported Versions

1.0a
2.05a
3.0

Overview

"OPC" stands for Open Productivity and Connectivity in industrial automation and the enterprise systems that support industry. It is a client/server technology where one application acts as the server (providing data) and another acts as a client (using data).

OPC is composed of a series of standards specifications: OPC Data Access (DA) is the most prolific standard. OPC DA is a widely accepted industrial communication standard that enables data exchange between multi-vendor devices and control applications without proprietary restrictions. An OPC server can communicate data continuously among PLCs on the shop floor, RTUs in the field, HMI stations, and software applications on desktop PCs. OPC compliance makes continuous real-time communication possible (even when the hardware and software are from different vendors).



OPC Data Access 1.0a was the original specification developed by the OPC Foundation in 1996. Although it continues to be supported by many of the OPC client applications in use today, OPC Data Access 2.0 Enhanced OPC better utilizes the underlying Microsoft COM technology. OPC Data Access 3.0 is the latest version of the OPC DA interface.

See Also: *Project Properties — OPC DA*

OPC AE

Supported Versions

1.0
1.10

Overview

OPC Alarms & Events (AE) is a specification developed by the OPC Foundation to standardize the way that alarm and event information is shared among systems. Using the standard, AE clients can receive alarms and event notices for equipment safety limits, system errors, and other abnormal situations.

Simple Events

Simple Events include the server events displayed in the Event Log (such as information, warning, error, and security events). The server supports the following filtering options for Simple Events for AE clients:

- **Event Type** Simple.
- **Event Category** Filter by server-defined categories. Each event is assigned to one category. Descriptions of the categories are as follows:
    - **Runtime Error Events** Simple events that are shown as errors in the Event Log.
    - **Runtime Warning Events** Simple events that are shown as warnings in the Event Log.
    - **Runtime Information Events** Simple events that are shown as informational in the Event Log.

Condition Events

Condition Events are created by server conditions, which are currently only configurable through the use of the Alarms & Events plug-in. The server supports the following filtering options for Condition Events for AE clients:

1. **Event** Condition.
2. **Category** Filter by server-defined categories. Each event is assigned to one category. Descriptions of the categories are as follows:
    - **Level Alarms** Events that are generated by process level conditions. For example, tank level > 10.
    - **Deviation Alarms** Events that are generated by deviation conditions. For example, tank level ± 10.
    - **Rate of Change Alarms** Events that are generated by rate of change conditions.
3. **Severity** Filter by severity level. Levels range from 0 to 1000; 1000 is the most severe. Each event is assigned a severity.
4. **Area** Filter by a process area to get alarms and events from only that area. An area is used to organize alarm and event information.
5. **Source** Filter by source to get events from only that source. A source is an Alarms & Events area that was created by a source (such as a server tag) that belongs to an area.

Note: The Alarms & Events Plug-In allows conditions to be configured through server tags. For example, a Temperature tag can be configured through the Alarms & Events Plug-In to generate an event when the maximum value is reached. For more information on the Alarms & Events Plug-In, contact an OPC vendor.

See Also: *Project Properties — OPC AE*

Optional Interfaces

The AE server interface does not support the following optional interfaces:

- **IOPCEventServer::QueryEventAttributes** This interface manages event attributes, which are not supported by the server. Attributes allow custom information to be added to an event (such as special messages or server tag values). This also applies to the IOPCEventSubscriptionMgt::SelectReturnedAttributes interface and the IOPCEventSubscriptionMgt::GetReturnedAttributes interface.
- **IOPCEventServer::TranslateToItemIDs** This interface allows AE clients to get the OPC DA item related to the event. This is because in some cases, events are related to the value of a server tag.
- **IOPCEventServer2:** This interface allows clients to enable/disable areas and sources. This interface is not supported by the server, because it would allow one client to enable/disable an area or source for all clients.

Note: The AE server interface does not support tracking events.

OPC UA Interface

Supported Version

1.02 optimized binary TCP

Overview

OPC Unified Architecture (UA) is an open standard created by the OPC Foundation with help from dozens of member organizations. It provides an additional way to share factory floor data to business systems (from shop-floor to top-floor). UA also offers a secure method for remote client-to-server connectivity without

depending on Microsoft DCOM. It has the ability to connect securely through firewalls and over VPN connections. This implementation of the UA server supports optimized binary TCP and the DA data model.

Note: Currently, neither UA via HTTP/SOAP web services nor for complex data is supported. For more information, refer to the OPC UA Configuration Manager help file.

OPC UA Profiles

OPC UA is a multi-part specification that defines a number of services and information models referred to as features. Features are grouped into profiles, which are then used to describe the functionality supported by a UA server or client. For a full list and a description of each OPC UA profile, refer to http://www.opcfoundation.org/profilereporting/index.htm.

Fully Supported OPC UA Profiles

- Standard UA Server Profile
- Core Server Facet
- Data Access Server Facet
- SecurityPolicy - Basic128Rsa15 (Deprecated)
- SecurityPolicy - Basic256 (Deprecated)
- SecurityPolicy - Basic256Sha256
- SecurityPolicy - None (Insecure)
- UA-TCP UA-SC UA Binary

CAUTION: Security policies Basic128Rsa15 and Basic256 have been deprecated by the OPC Foundation as of OPC UA specification version 1.04. The encryption provided by these policies is considered less secure and usage should be limited to providing backward compatibility.

Partially Supported OPC UA Profiles

- Base Server Behavior Facet

Note: This profile does not support the Security Administrator – XML Schema.

See Also: *Project Properties — OPC UA*

OPC .NET

Supported Version

1.20.2

Overview

OPC .NET is a family of APIs provided by the OPC Foundation that leverage Microsoft's .NET technology and allow .NET clients to connect to the server. This server supports OPC .NET 3.0 WCF, formally known as OPC Xi. Unlike other OPC .NET APIs, OPC .NET 3.0 uses Windows Communication Foundation (WCF) for connectivity, avoiding DCOM issues and providing the following benefits:

- Secure communication via multiple communications bindings (such as Named Pipe, TCP, Basic HTTP, and Ws HTTP).
- Consolidation of OPC Classic Interfaces.
- Simple development, configuration, and deployment of Windows environment.

The server adds OPC .NET 3.0 support using a customized version of the OPC .NET 3.0 WCF Wrapper supplied by the OPC Foundation. The wrapper runs as a system service called "xi_server_runtime.exe". It wraps

the existing server's OPC AE and DA interfaces, providing WCF clients access to the server's tag and alarm data. It does not support Historical Data Access (HDA).

Note: The OPC .NET service is only started when the server starts and the interface is enabled. Unlike OPC DA, clients cannot launch the server. For more information on configuration, refer to **Project Properties – OPC .NET**.

Requirements
To install and use OPC .NET 3.0, Microsoft .NET 3.5 must be present on the machine before server installation.

DDE

Supported Formats
CF_Text
XL_Table
Advanced DDE

Overview
Although this server is first and foremost an OPC server, there are still a number of applications that require Dynamic Data Exchange (DDE) to share data. As such, the server provides access to DDE applications that support one of the following DDE formats: CF_Text, XL_Table, and Advanced DDE. CF_Text and XL_Table are standard DDE formats developed by Microsoft for use with all DDE aware applications. Advanced DDE is a high performance format supported by a number of client applications specific to the industrial market.

CF_Text and XL_Table
The DDE format CF_Text is the standard DDE format as defined by Microsoft. All DDE aware applications support the CF_Text format. XL_Table is the standard DDE format as defined by Microsoft that is used by Excel. *For more information on DDE, refer to **How To... Use DDE with the Server**.*

Advanced DDE
Advanced DDE is the DDE format defined by Rockwell Automation. Today, all Rockwell client applications are Advanced DDE aware. Advanced DDE is a variation on the normal CF_Text format, which allows larger amounts of data to transfer between applications at higher rates of speed (and with better error handling).

Requirements
For the DDE interface to connect with the server, the Runtime must be allowed to interact with the desktop. *For more information, refer to **How To... Allow Desktop Interactions**.*

See Also**: **Project Properties — DDE

FastDDE / SuiteLink

Overview
FastDDE is a DDE format defined by Wonderware Corporation. It allows larger amounts of data to transfer between applications at higher speed (and with better error handling) than generic DDE. SuiteLink is a client-server communication method that has succeeded FastDDE. It is TCP/IP based and has improved bandwidth and speed. Both FastDDE and SuiteLink are supported by all Wonderware client applications.

Note: The Wonderware connectivity toolkit is used to simultaneously provide OPC and FastDDE / SuiteLink connectivity, allowing quick access to device data without the use of intermediary bridging software.

For security reasons, it is recommended that users utilize the most recent Wonderware DAServer Runtime Components. *For more information and available downloads, refer to the Invensys Global Technical Support WDN website.*

Requirements

For the FastDDE interface to connect with the server, the Runtime must be allowed to interact with the desktop. *For more information, refer to* ***How To... Allow Desktop Interactions***.

See Also: ***Project Properties — FastDDE / SuiteLink***

iFIX Native Interfaces

Overview

The iFIX native interface simplifies the connection task by allowing a direct connection to the local iFIX application without the use of the iFIX OPC Power Tool. When supported, this interface also has the ability to refine the connection between the server and the iFIX Process Database (PDB).

See Also: ***Project Properties — iFIX PDB Settings***

ThingWorx Native Interface

Overview

ThingWorx is a connectivity platform that allows users to create useful and actionable intelligence based on their device data. The ThingWorx Native Interface allows a user to provide data to the ThingWorx Platform with little additional configuration using the ThingWorx Always On technology. With the introduction of the ThingWorx Next Gen Composer, the ThingWorx Native interface has been updated to allow a better user interface integration with the Composer.

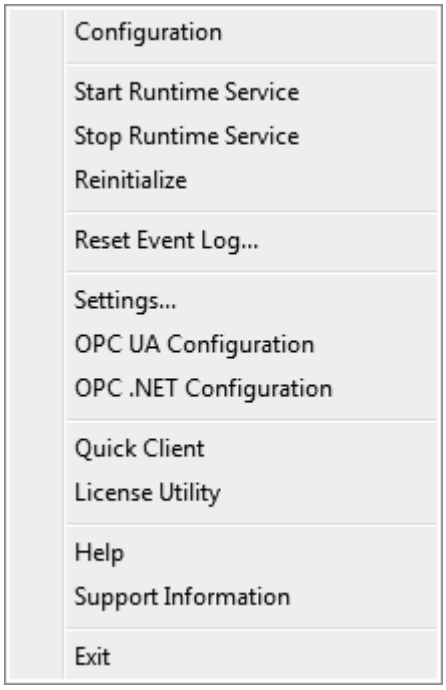
As noted in the ThingWorx documentation, configuration of a ThingWorx Application Key is crucial to providing a secured environment. The Application Key that is used should provide the appropriate privileges to allow the proper exchange of data between the server instance and the ThingWorx Platform.

The ThingWorx Native Interface supports Store and Forward to cache property updates when the industrial server becomes disconnected from the ThingWorx platform.

See Also:
Project Properties – ThingWorx Native Interface
Fill Rate Example
Store and Forward System Tags

Accessing the Administration Menu

The Administration Menu is a tool that is used to view and/or modify user management settings and launch server applications. To access the Administration Menu, right-click on the Administration icon located in the System Tray.



Configuration: This option launches the OPC server's configuration.

Start Runtime Service: This option starts the server Runtime process and loads the default Runtime project.

Stop Runtime Service: This option disconnects all clients and saves the default Runtime project before stopping the server Runtime process.

Reinitialize: This option disconnects all clients and resets the Runtime server. It automatically saves and reloads the default Runtime project without stopping the server Runtime process.

Reset Event Log: This option resets the Event Log. The date, time, and source of the reset are added to the Event Log in the configuration window.

Settings...: This option launches the Settings group. For more information, refer to **Settings**.

OPC UA Configuration: This option launches the OPC UA Configuration Manager, if available.

OPC .NET Configuration: This option launches the OPC .NET Configuration Manager.

Quick Client: This option launches the Quick Client.

License Utility: This option launches the server's license utility.

Help: This option launches the server's help documentation.

Support Information: This option launches a dialog that contains basic summary information on both the server and the drivers currently installed for its use. For more information, refer to **Server Summary Information**.

Exit: This option closes the Administration and removes it from the System Tray. To view it again, select it from the Windows Start menu.

Settings

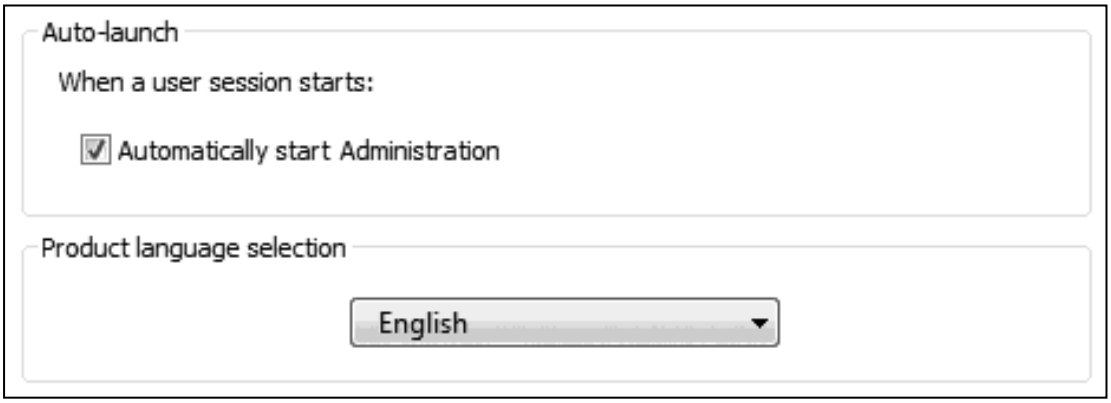
To access the Settings groups, right-click on the Administration icon located in the System Tray. Select **Settings**. For more information, select a link from the list below.

Settings — Administration
Settings — Configuration
Settings — Runtime Process
Settings — Runtime Options
Settings — Event Log
Settings — ProgID Redirect
Settings — User Manager
Settings — Configuration API Service
Settings — Certificate Store
Settings — Service Ports

Security Policies — A plug-in is available for user permissions and access control. Consult the product help system.
Local Historian — A plug-in is available for data storage and access. Consult the product help system.
IoT Gateway — A plug-in is available for Industrial Internet of Things integration. Consult the product help system.

Settings — Administration

The Administration group is used to configure the Runtime Administration's actions.



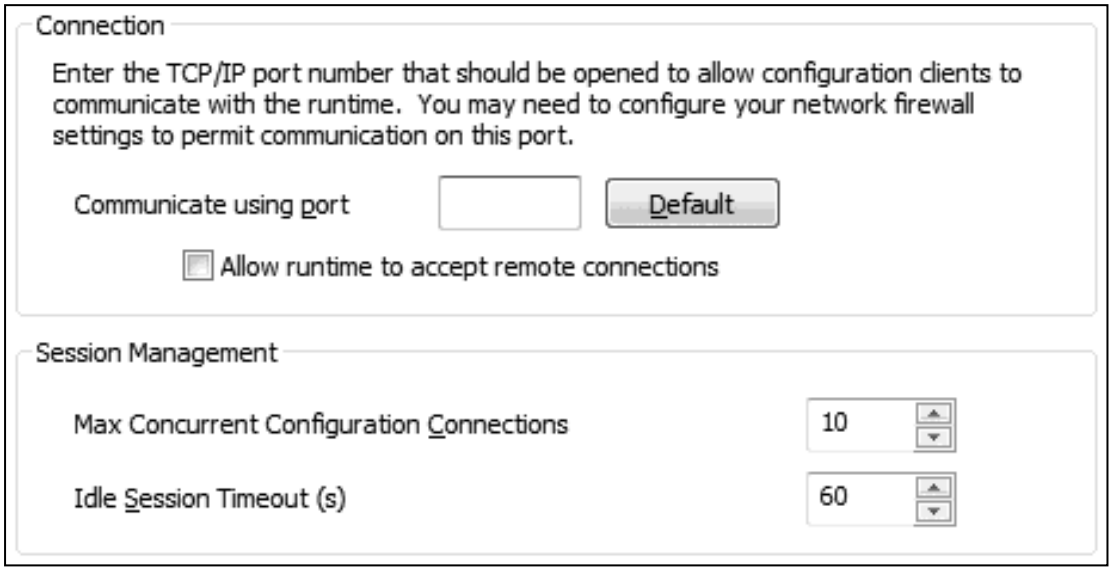
Automatically start Administration: When enabled, this property enables the Administration to start automatically. The Administration is a System Tray application that allows quick links to various server tools including the Settings Console, Configuration, Licensing Utility, User Manager Console and controls for stopping and starting the Runtime service.

Product Language Selection: Select the preferred user interface language from the drop-down menu.

Tip: The language settings defaults to the language of the install, which defaults to the language setting in the operating system, if possible.

Settings — Configuration

The Configuration group is used to configure how the Configuration both connects to and interacts with the Runtime.



Connection

Communicate using port: This property is the TCP/IP port to be used to communicate between the Configuration and the Runtime. To obtain the default setting, click **Default**.

Allow runtime to accept remote connections: When enabled, the runtime accepts remote connections. The default setting is disabled.

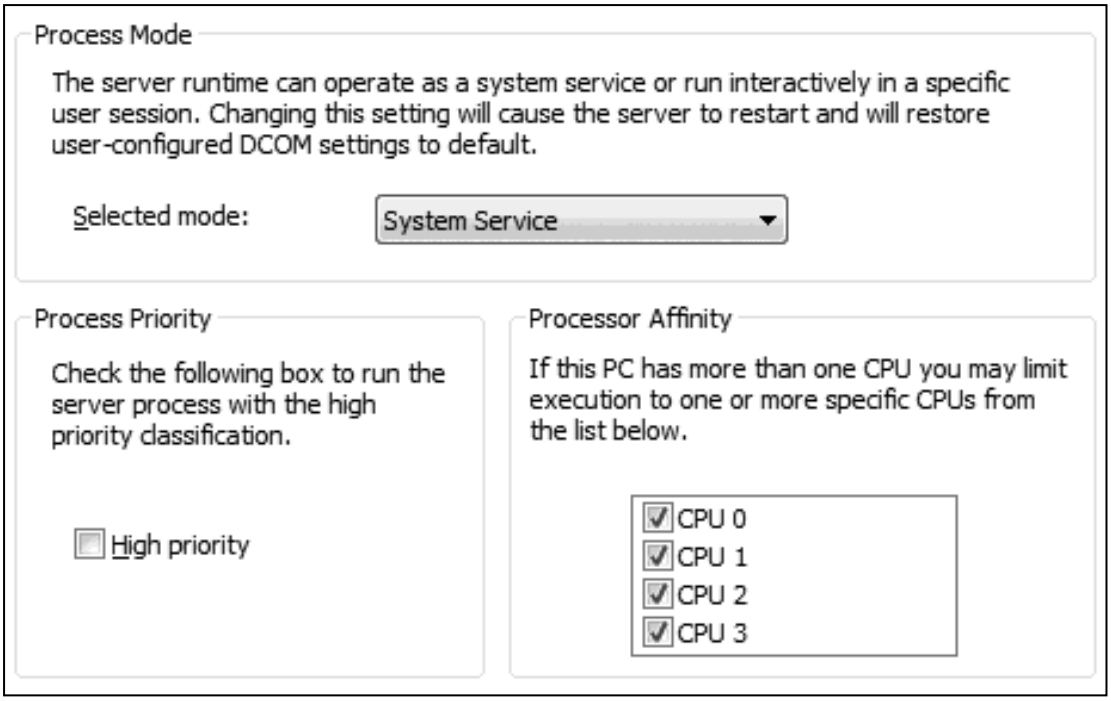
Session Management

Max Concurrent Configuration Connections: Specify the number of Configuration connections that can be made to the Runtime at one time. The range is 1 to 64. The default is 10.

Idle Session Timeout: Set the length of time the console connection can be inactive before it is shut down. The range is 10 to 3600 seconds. The default is 60 seconds.

Settings — Runtime Process

The Runtime Process group is used to specify the server Runtime's process mode, as well as how it utilizes the PC's resources.



Selected Mode: This property is used to specify whether the server is running as **System Service** or **Interactive**. By default, the server installs and runs as System Service. Changing this setting causes all clients, both Configuration and process, to be disconnected and the server to be stopped and restarted. It also restores user-configured DCOM settings to default.

High Priority: This property is used set the server process priority to high. The default setting is normal. When enabled, this setting allows the server to have priority access to resources.

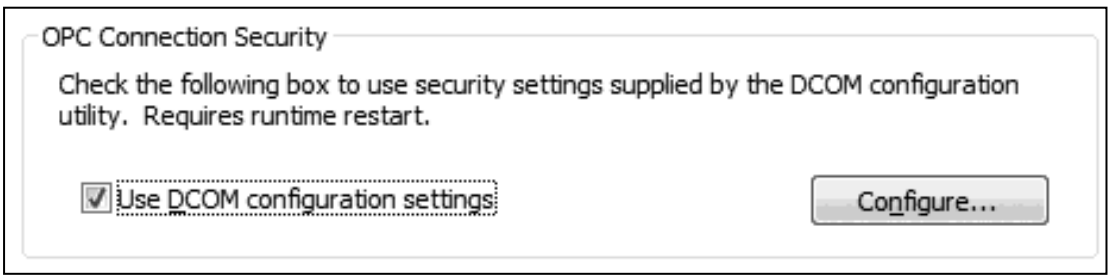
Note: Microsoft recommends against setting applications to a high priority as it can adversely affect other applications running on the same system.

Processor Affinity: This property is used to specify on which CPUs the server can be executed when it is run on PCs containing more than one.

Settings — Runtime Options

The Runtime Options group is used to change settings in the project being executed in the Runtime.

OPC Connection Security



Use DCOM configuration settings: Enable to use authentication and security from the DCOM Configuration.

Configure... Click to launch the DCOM Configuration Utility to specify the level of security and restrict access for certain users and/or applications.

When this setting is disabled, the server overrides the DCOM settings set for the application and does not perform any authentication on the calls received from client applications. It impersonates the security of the client when performing any actions on behalf of the client application. Disabling this setting provides the lowest level of security and is not recommended. If this setting is chosen, ensure that the client and server applications are running in a secure environment so that the application is not compromised.

Project Backups

Backup the Runtime project prior to replacement: This property enables the Runtime project to be backed up before it is overwritten. The backup location is displayed in the Event Log. This option is enabled by default.

Note: The Runtime project is overwritten if either **New** or **Open** is selected while connected to the Runtime. In addition, connecting to the Runtime while working offline with a project may result in Runtime project replacement.

Keep the most recent: This property limits the number of backup files to be saved to disk. The range is 1 to 1000. The default is 10.

Clean up now...: This property invokes a confirmation dialog that allows users to delete all the Runtime project backups. Doing so does not affect the current running project.

Tip: It is a best practice to save a copy of the project file on a regular basis for disaster recovery purposes. The default directories for these backups are:

For 64-bit OS versions, backup project files are saved in:
C:\ProgramData\Kepware\KEPServerEX\V6\Project Backups

For 32-bit OS versions, backup project files are saved in:
C:\ProgramData(x86)\Kepware\KEPServerEX\V6\Project Backups

Tip: If the file has been saved to an alternate location, search for *.opf, *.sopf, or *.json to locate available project files.

Settings — Event Log

The Event Log group is used to define the communication and persistence settings for the Event Log, OPC Diagnostics Log, and Communications Diagnostics Log.

The settings for each individual log type are independent of the settings for the other log types.

Connection

Port: Specify the TCP/IP port to be used to communicate between the Log and the Runtime. The valid range is 49152 to 65535. To restore the default port setting, enter a blank value.

Event Log Settings

Persistence Mode: icon to open the log's persistence mode. Options include Memory, Single File, and Extended Datastore. The default setting for the Event Log Setting is Single File. The default setting for both OPC Diagnostics Log Settings and Communications Diagnostics Log Settings is Memory (no persistence). Descriptions of the options are as follows:

- **Memory (no persistence)**: When selected, this mode records all events in memory and does not generate a disk log. A specified number of records are retained before the oldest records start being deleted. The contents are removed each time the server is started.
- **Single File**: When selected, this mode generates a single disk-based log file. A specified number of records are retained before the oldest records start being deleted. The contents are restored from this file on disk when the server is started.
- **Extended Data Store**: When selected, this mode persists a potentially large number of records to disk in a data store distributed across many files. The records are retained for a specified number of days before being removed from the disk. The contents are restored from the distributed file store on disk when the server is started.

Max. records: Specify the number of records that the log system retains before the oldest records start being deleted. It is only available when the Persistence Mode is set to Memory or Single File. The valid range is 100 to 100,000 records. The default setting is 25,000 records.

Note: The log is truncated if this property is set to a value less than the current size of the log.

Log file path: Specify where the disk log is stored. It is only available when the Persistence Mode is set to Single File or Extended Datastore.

Note: Attempts to persist diagnostics data using a mapped path may fail because the Event Log service is running in the context of the SYSTEM account and does not have access to a mapped drive on the local host. Users that utilize a mapped path do so at their own discretion. It is recommended that the Uniform Naming Convention (UNC) path be used instead.

Max. single file size: Specify the size that a single datastore file must attain before a new datastore file can be started. It is only available when the Persistence Mode is set to Extended Datastore. The valid range is 100 to 10000 KB. The default setting is 1000 KB.

Min. days to preserve: Specify that individual datastore files are deleted from disk when the most recent record stored in the file is at least this number of days old. It is only available when the Persistence Mode is set to Extended Datastore. The valid range is 1 to 90 days. The default setting is 30 days.

See Also: *Built-In Diagnostics*

When saving to file, monitor the Windows Event Viewer for errors relating to the persistence of data to disk.

Restoring Persisted Datastores from Disk

The Event Log restores records from disk either at start up or when the following occurs:

1. The Persistence Mode is set to Single File or Extended Datastore.

   Note: When Single File persistence is selected, the server loads all persisted records from disk before making any records available to clients.

2. The log file path is set to a directory that contains valid persisted log data.

Extended Datastore Persistence

The Extended Datastore Persistence Mode has the potential to load a very large number of records from disk. To remain responsive, the log services client requests for records while records are loaded from disk. As the record store is loaded, clients are provided with all records in the log regardless of filtering. Once all the records have been loaded, the server applies filters and sorts the records chronologically. The client views are updated automatically.

Note: Loading large record stores may cause the log server to be less responsive than usual. It regains full responsiveness once the loading and processing completes. Resource usage is higher than usual during loading and settles on completion.

Disk Full Behavior

The Extended Datastore Persistence Mode has the potential to fill a storage medium quickly, especially when persisting OPC Diagnostics. If a disk error is encountered while persisting records, an error posts to the Windows Event Viewer.

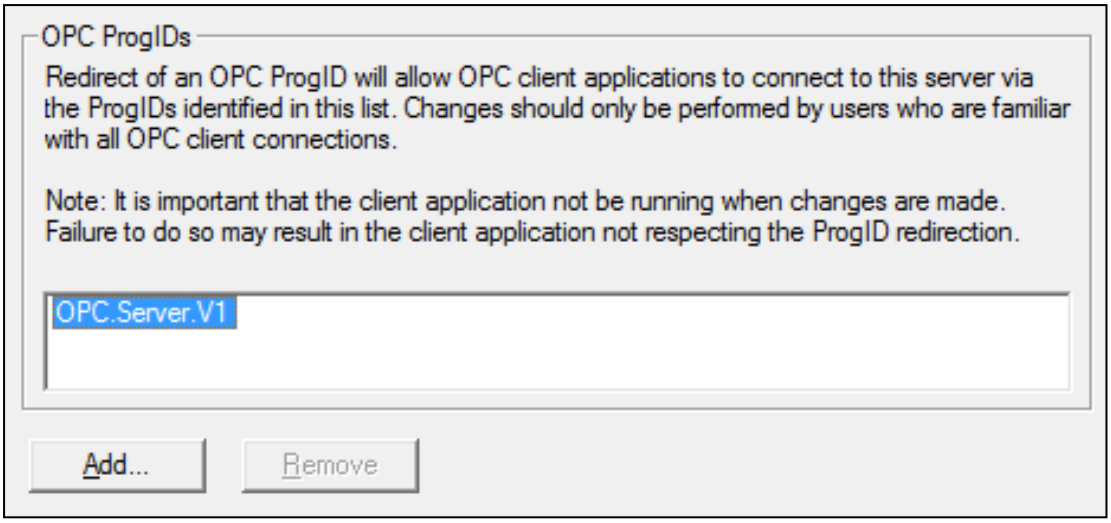
See Also: OPC Diagnostics Viewer

The Event Log system would be useless if there was no mechanism to protect its contents. If operators could change these properties or reset the log, the purpose would be lost. Utilize the User Manager to limit what functions an operator can access.

Settings — ProgID Redirect

Many OPC client applications connect to an OPC server through the OPC server's ProgID. Users who need to migrate or upgrade to a new OPC server often prefer to do so without changing their tag database (which can contain thousands of tags that link to the OPC server ProgID). This server offers ProgID redirection to assist users in these transitions.

The ProgID Redirect feature allows users to enter the legacy server's ProgID. The server creates the necessary Windows Registry entries to allow a client application to connect to the server using the legacy server's ProgID.



Add: This button is used to add a ProgID to the redirection list. When clicked, it invokes the "Add New ProgID" dialog. For more information, refer to "Adding a New ProgID" below.

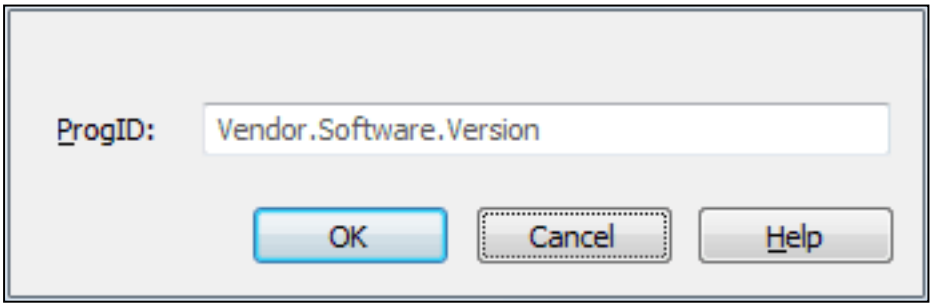
Remove: This button is used to remove a selected ProgID from the redirection list.

Note: A redirected ProgID cannot be browsed by OPC client applications that use the OpcEnum service to locate OPC servers. In most cases, users can enter the redirected ProgID into the client application manually.

Adding a New ProgID

For more information, refer to the instructions below.

1. In the **ProgID Redirect** group, click **Add**.
2. In **ProgID**, enter the ProgID of the legacy server.



3. Once complete, click **OK**.

The client application should not be running while the legacy server's ProgID is being added to the redirection list. Failure to observe this warning may result in the client application not respecting the newly redirected ProgID.

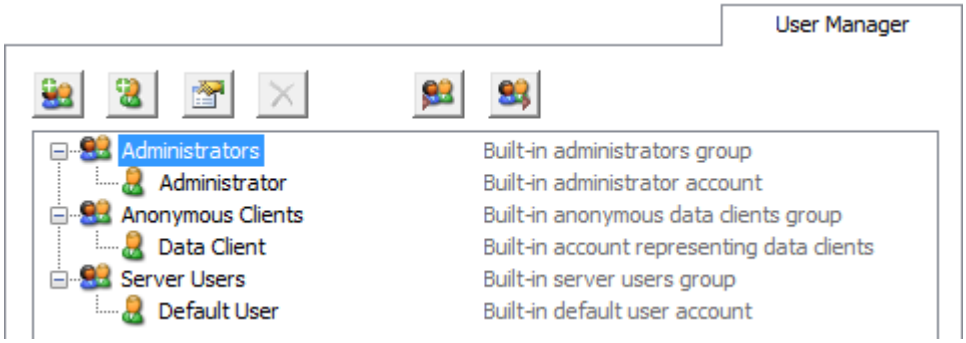
Settings — User Manager

The User Manager controls client access to the project's objects (which are the channels, devices, tags. etc.) and their corresponding functions. The User Manager allows permissions to be specified by user groups. For example, the User Manager can restrict the Data Client user access to project tag data based on its per-

missions from the Anonymous Clients user group. The User Manager can also transfer user information between server installations through its import / export function.

The User Manager has three built-in groups that each contain a built-in user. The default groups are Administrators, Server Users, and Anonymous Clients. The default users are Administrator, Default User, and Data Client. Users cannot rename or change the description fields. Neither the default groups nor the default users can be disabled.

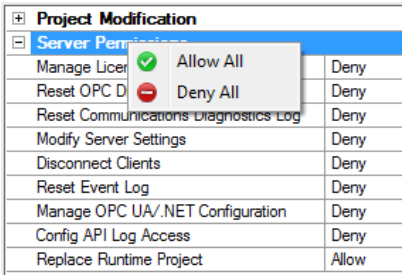
Note: Although the Administrator's settings cannot be changed, additional administrative users can be added.



New Group: When clicked, this button adds a new user group.
For more information, refer to ***User Group Properties***.

New User: When clicked, this button adds a new user to the selected user group. This function is disabled for anonymous clients.
For more information, refer to ***User Properties***.

Edit Properties When clicked, this button allows users to edit the properties of the selected user or user group.
Tip: To update multiple permissions at the same time, right-click on the property group and select the desired permissions.



Disable Selected User / Group: When clicked, this button disables the selected user or user group. This function is only available to custom users and user groups. Disabling a user group disables all users within it.
Note: Disabling a user or user group invokes the **Show disabled**: option. If enabled, this option makes any disabled users and user groups visible in the user group and user list.

Restore Selected User / Group: When clicked, this button restores the selected user or user group. Restoring a user group returns the users within it to the state they were in prior to disabling. This icon is only available once a user or user group has been disabled.
Note: If all disabled users and user groups are restored, the **Show disabled** option is not displayed.

Import User Information: When clicked, this button imports user information from an XML file. For the import to succeed, the file that is selected must have been exported from the server's Administration utility. This function is only enabled when the built-in Administrator is logged in.

Export User Information: When clicked, this button exports user information to an XML file. This is useful for users that need to move the project from one machine to another. Administrators also have the option to password protect the XML file: if utilized, the correct password must be entered for the import to succeed on the new machine. The XML file cannot be edited and re-imported. This function is enabled at all times.

- The Import / Export User Information features were released in server version 5.12. Any user passwords that were set while using previous server versions must be changed in 5.12 before an export is attempted; otherwise, the export fails.

- After upgrading the server or importing User Information, it is recommended to review the User Manager permissions for accuracy.

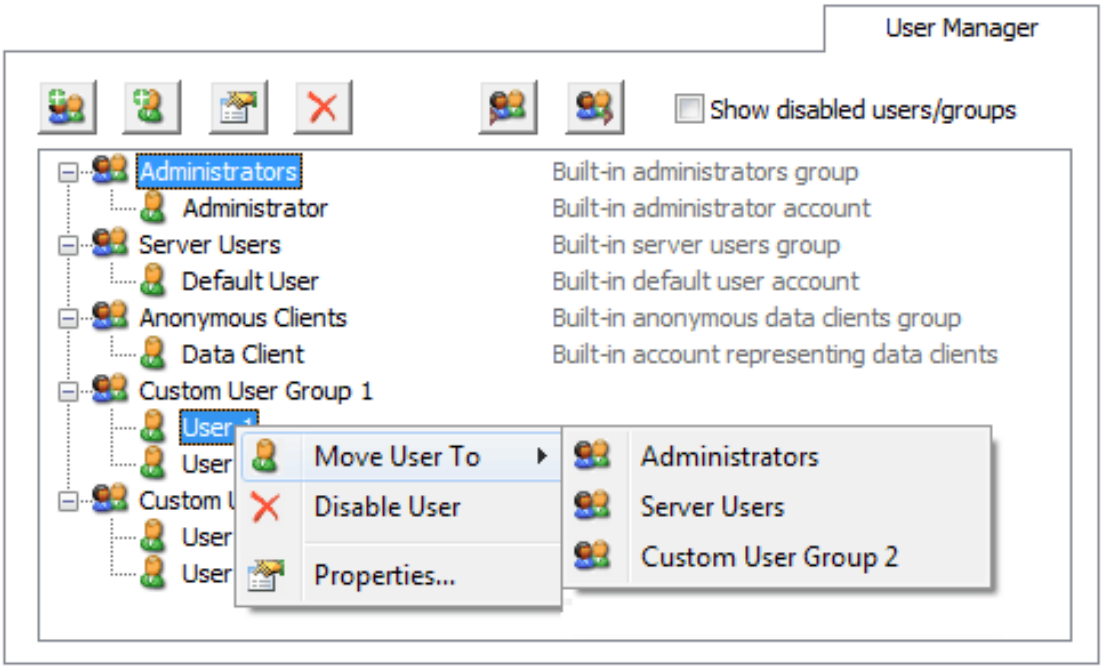
- **Note**: Although custom users and user groups cannot be deleted once they have been created, the Import User Information option replaces existing users and user groups with those being imported (except for the Administrator built-in user).

- For the sake of project preservation, it is recommended that users export a copy of the user information once it is complete. A project cannot load without correct user information.

- ***See Also***: *ThingWorx Interface Users if connecting to the ThingWorx platform.*

Accessing Additional Settings

Shortcuts and additional settings may be accessed through the context menus for user groups and users.

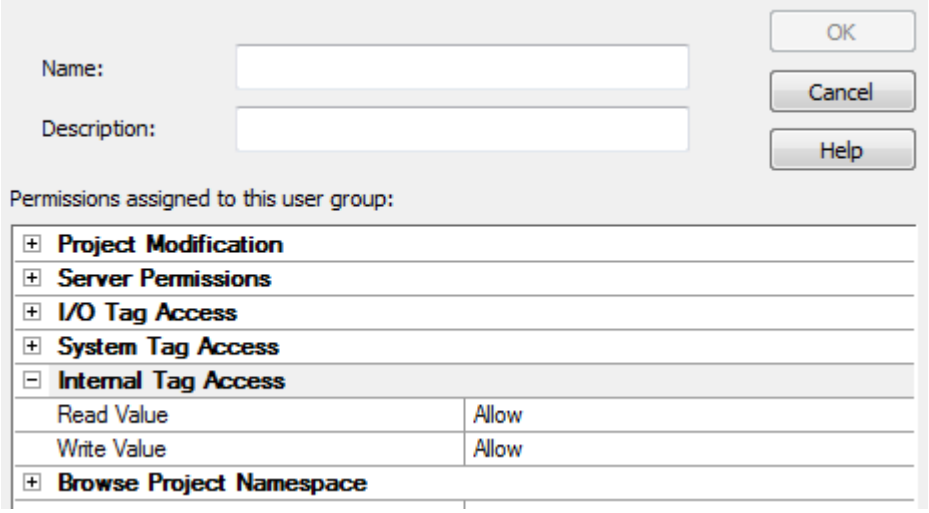


Move User To This option moves the user to a different user group. The status of the group does not matter: both disabled and enabled groups appear in the list. An active user moved to a disabled group becomes disabled as well. A disabled user moved to an enabled group persists in status until changed.

User Group Properties

The user group properties may also be accessed by right-clicking on a user group and selecting **Properties**.

Tip: To quickly allow or deny all options in a category, right-click on the category and select **Allow All** or **Deny All**. A setting that displays bold text indicates that its value has been changed. Once the change is saved, the text displays as normal.



Name: Click the icon to open the name of the new user group. The maximum number of characters allowed is 31. Duplicate names are not allowed.

Description: This optional property provides a brief description of the user group. This can be particularly helpful for operators creating new user accounts. The maximum number of characters allowed is 128.

Permissions assigned to this user group: This field assigns permissions for the selected user group. Permissions are organized into the following categories: Project Modification, Server Permissions, I/O Tag Access, System Tag Access, Internal Tag Access, and Browse Project Namespace. More information on the categories is as follows:

- **Project Modification**: This category specifies permissions that control default project modifications.
- **Server Permissions**: This category specifies permissions that control access to server functionality. These permissions are not supported by the anonymous client.
- **I/O Tag Access**: This category specifies permissions that control access to device-level I/O tag data. These tags require device communications, and are described as Static tags in the server.
- **System Tag Access**: This category specifies permissions that control access to System tags. These tags begin with an underscore and exist in a server-defined location. For more information, refer to System Tags.
- **Internal Tag Access**: This category specifies permissions that control access to internal tags. These tags are either driver-managed (controlling some aspect of the driver's operation) or user-specified (at a plug-in level).
- **Browse Project Namespace**: This category specifies permissions that control browse access to the project namespace in clients that support browsing. This is only supported by a few client types at this time.

Tip: To view more information on a specific object in a category, select it.

User Properties

The user properties may be accessed by double-clicking on the user or right-clicking on the user and selecting **Properties...**.

Old Password: This field holds the password that has been active for this user.

Password: Enter a new or updated password this user must enter to log into the system. It is case-sensitive with a maximum of 512 characters allowed.

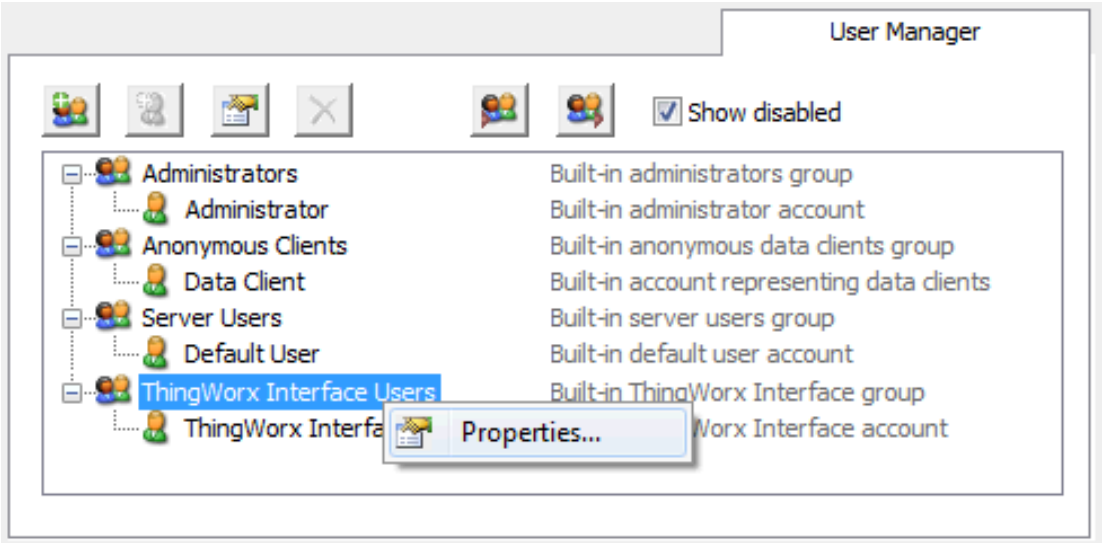
Confirm Password: Re-enter the same password. It must be entered exactly the same in both the New Password and Confirm Password fields.

Note: It is recommended that the password be at least 14 characters and include a mix of uppercase and lowercase letters, numbers, and special characters. Avoid well-known, easily guessed, or common passwords.

Settings — User Manager ThingWorx Interface Users

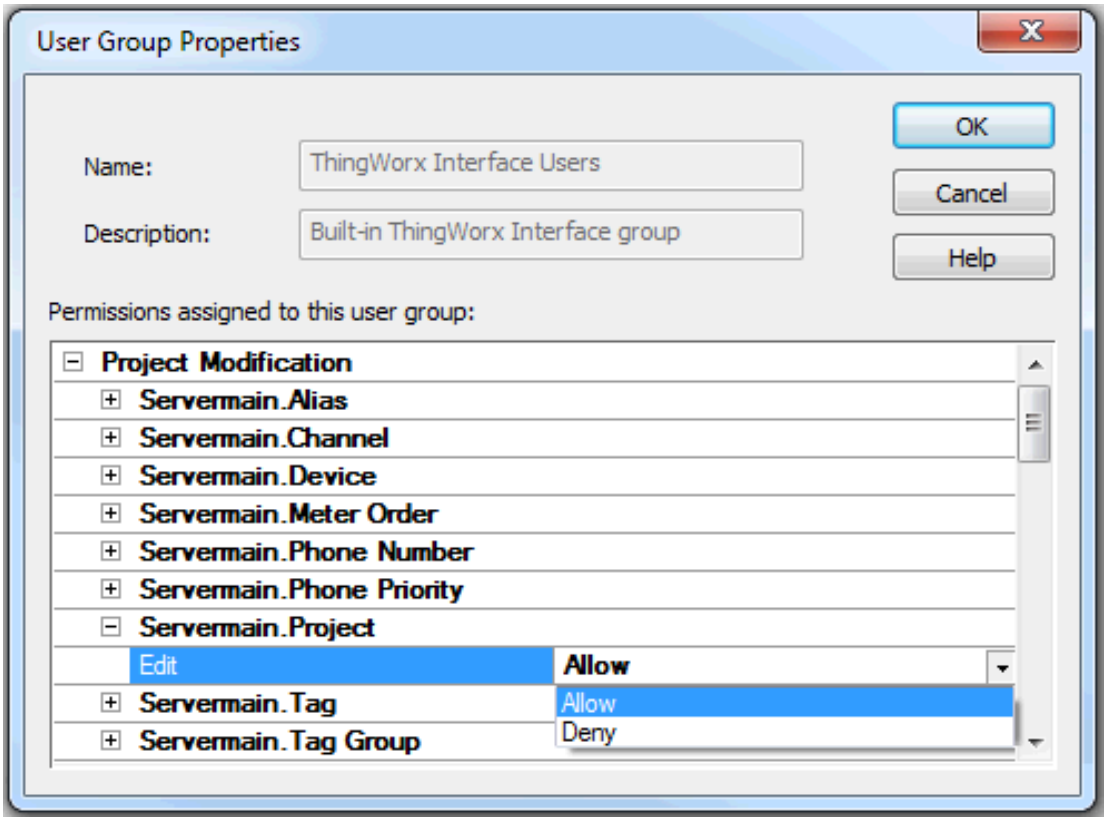
The User Manager controls client access to project objects and their corresponding functions. All of the buttons and controls function as described in the general User Manager section. The ThingWorx Interface Users group controls access to, data exchange with, and analysis in a connected ThingWorx Platform instance.

See Also: *User Manager*



To allow adequate access for data transfer between the server and the ThingWorx Platform, project modification and store and forward must be enabled. To grant the correct access for this functionality:

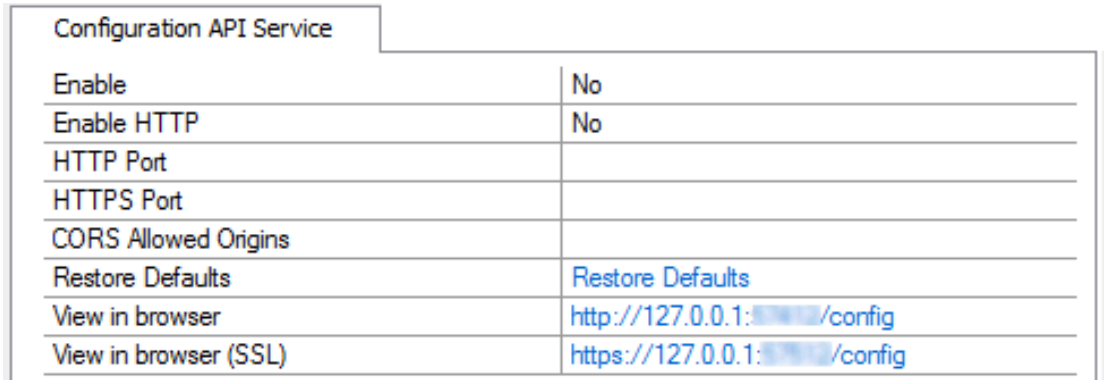
1. Select the **ThingWorx Interface Users** group.
2. Right-click and select **Properties...**.
3. Expand the **Project Modification** group.
4. Locate and expand the **Servermain.Project** rights.
5. Using the drop-down menu, select **Allow** to grant permission to change the project file.
6. Click **OK** to close.



Settings — Config API Service Configuration

The Configuration API Service is configured on installation. If the settings need to be adjusted, access the Configuration API Service settings by right-clicking on the Administration icon in the system tray and selecting **Settings | Configuration API Service**.

- If the Administrative icon is not in the system tray, re-launch it by selecting **Start | All Programs | Kepware | KEPServerEX 6 | KEPServerEX 6 Administration | Settings**.



Enable: Choose Yes to enable the Configuration API Server. If disabled (No); the service runs, but does not bind to the HTTP and HTTPS ports and clients cannot access the server.

Enable HTTP: Select No to limit data transfer to only secure / encrypted protocols and endpoints. Select Yes to allow unencrypted data transfer.

- **Tips**:

1. HTTP is only recommended for internal networks because user authentication, application keys, and other sensitive data is transmitted as plain text.
2. To prevent external access over unsecure HTTP, this port should be blocked by the Windows firewall.

HTTP Port: Specify the TCP/IP port for the REST client to communicate over unencrypted HTTP. The valid range is 1 to 65535. HTTP and HTTPS ports must not match. The default port number of 57412.

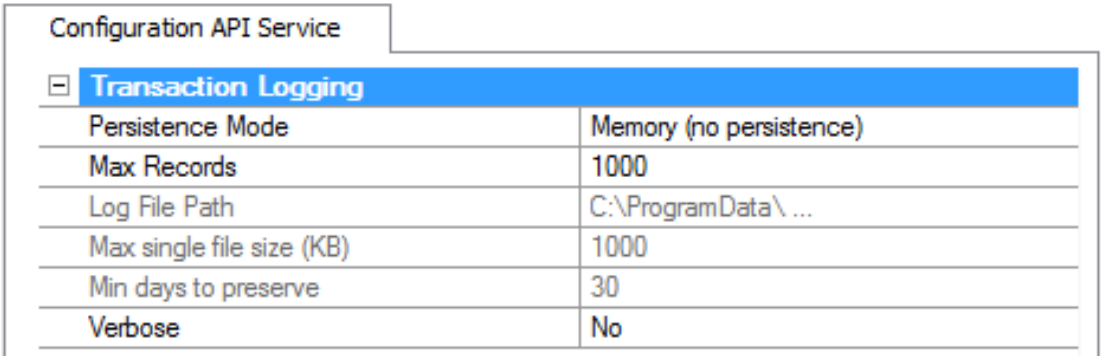
HTTPS Port: Specify the TCP/IP port for the REST client to communicate over secure HTTP. The valid range is 1 to 65535. HTTP and HTTPS ports must not match. The default port number of 57512.

CORS Allowed Origins: Specify an approved "white-list" of comma-separated domain specifications that may access the Configuration API Server for Cross Origin Resource Sharing (CORS) requests.

Restore Defaults: click to blue link to the right to restore the default HTTP and HTTPS port values.

View in Browser: click the blue address link to the right to open the Configuration API documentation landing page in a browser.

View in Browser (SSL): click the blue address link to the right to open the Configuration API documentation landing page in a browser via the secure URL.



Transaction Logging

Persistence Mode: Select the record retention method for the system log. The default setting is Memory (no persistence). The options are:

- **Memory (no persistence)**: records all events in memory and does not generate a log that is saved to disk. A specified number of records are retained before the oldest records start being deleted. The contents are available only while the server is running.
- **Single File**: generates a recorded log file saved to disk. A specified number of records are retained before the oldest records start being deleted. The contents are restored from this file when the server is started.
- **Extended Datastore**: saves a potentially large number of records to disk distributed across multiple files. The records are retained for a specified number of days before being removed from the disk. The contents are restored from the distributed files on the disk when the server is started.

Max. Records: Specify the number of transactions the log retains before the oldest record is deleted. Available when the Persistence Mode is set to Memory or Single File. The valid range is 100 to 30000 records. The default setting is 1000 records.

Note: The log is truncated if this parameter is set to a value less than the current size of the log.

Log File Path: Indicate where the log is stored on disk. Available when the Persistence Mode is set to Single File or Extended Datastore.

Attempts to persist diagnostics data using a mapped path may fail because the Transaction Log service is running in the context of the SYSTEM account and does not have access to a mapped drive on the local host. Use a mapped drive path with caution. A Uniform Naming Convention (UNC) path is recommended.

Max. Single File Size: Indicate the size limit, in KB, of a single datastore file at which a new datastore file is started. Available when the Persistence Mode is set to Extended Datastore. The valid range is 100 to 10000 KB. The default setting is 1000 KB.

Min. Days to Preserve: Specify the number of days individual datastore files kept before being deleted from disk. Available when the Persistence Mode is set to Extended Datastore. The valid range is 1 to 90 days. The default setting is 30 days.

Verbose: Select Yes to record a detailed level of data is recorded in the log. Verbose logging includes HTTP request and response bodies in addition to the parameters included with non-verbose logging. *See **Verbose Logging** for more information.* Select No to record much less data and keep log files smaller.

Certificate Management

Note: An X.509 certificate is used to establish SSL communication between the client and the REST server. A default self-signed certificate is generated when the REST server is installed, but accessing the server from outside a secure network requires a trusted certificate.

View Certificate: Click to blue link to the right to open the current certificate to review.

Export Certificate: Click to blue link to the right to save the current certificate in .PEM format (such as for importing into third-party REST clients).

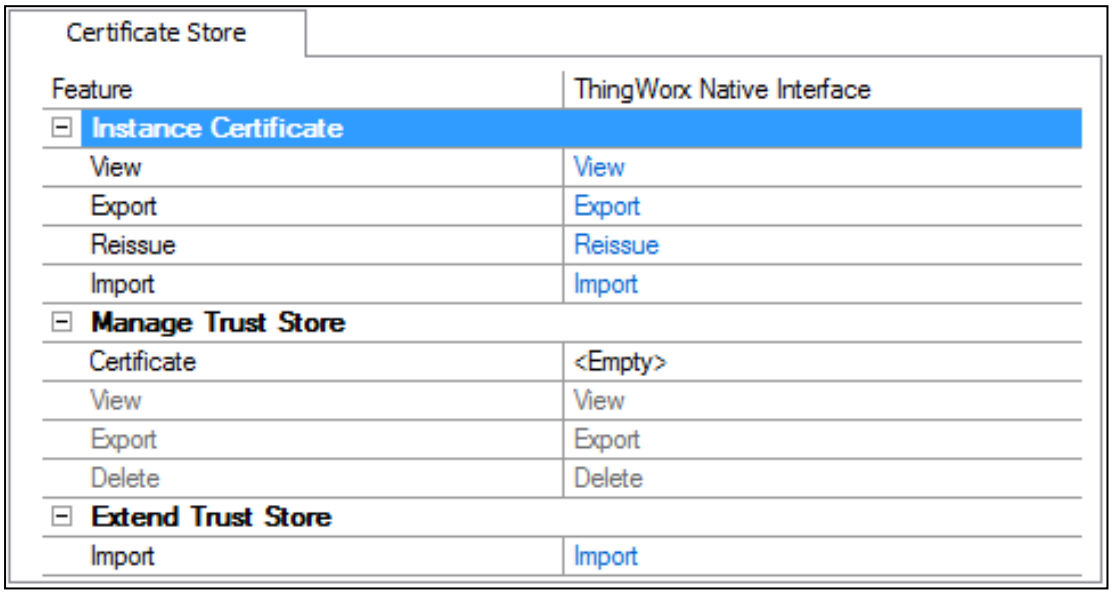
Reissue Certificate: Click to blue link to the right to create a new certificate, replacing the current certificate.

Import Certificate: Click to blue link to the right to import a certificate in .PEM format.

Note: A certificate is created on installation without additional configuration. When reissuing or importing a certificate, the new certificate in not applied until the Configuration API is stopped and restarted via the Windows Service Control Manager or the system restarts.

Settings — Certificate Store

The Certificate Store may be used to configure certificates for features that communicate securely using Transport Layer Security (TLS) or its older variant, Secure Socket Layer (SSL). This tab only appears if a feature is installed that is able to leverage it (such as the **ThingWorx Native Interface**).



Instance Certificate

View: Click the View link to view the currently selected feature's instance certificate.

Export: Save the currently selected feature's instance certificate to a directory chosen by the user. The suggested file name is the thumbprint of the certificate – though the user is free to change this. The output is PEM encoded and includes a single certificate.

Reissue: Reissue the currently selected feature's instance certificate. Certificates generated by the certificate store are self-signed, and expire in 10 years.

Import: Import the currently selected feature's instance certificate. Use this option to import a certificate that has been signed by a certificate authority that is trusted by the TLS/SSL peer.

Manage Trust Store

Certificate: The trust store may contain zero to many certificates. The user must select a certificate to view, export, or delete.

View: View the currently selected trust certificate for the currently selected feature.

Export: Export the currently selected trust certificate for the currently selected feature. As with the instance certificate, the output file is PEM encoded and contains a single certificate.

Delete: Delete the currently selected trust certificate for the currently selected feature. The feature no longer trusts peers that present certificates that include this certificate in their chain of trust.

Extend Trust Store

Import: Import one or more certificate authority or self-signed certificate(s) into the trust store. The feature trusts a TLS/SSL peer that presents this certificate or a certificate that is signed by the imported certificate.

Instance Certificate Import Behavior

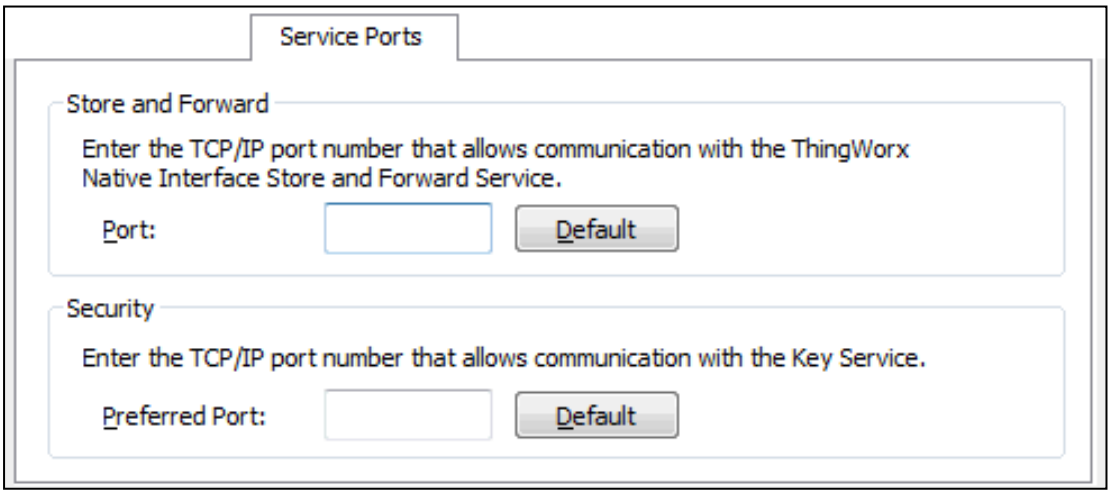
- The import file *must* contain a certificate and an unencrypted private key.
- The certificate cannot be imported if it contains an invalid signature.
- The user is prompted if the certificate is expired. The TLS/SSL peer may reject certificates that are expired.

Trust Certificate Import Behavior

- The import file should contain one or more certificate(s).
- No private key is necessary but can be present in the file.
- The import is not allowed to succeed if one or more certificates have an invalid signature.
- The import is not allowed to succeed if one or more certificates duplicate a certificate that is already present in the trust store.
- The user is prompted if any of the certificates in the import file are expired. The feature may reject certificates that rely on an expired certificate in the chain of trust.

Settings — Service Ports

The Administration group is used to configure the Runtime Administration's actions. The Service Ports administrative settings are automatically configured on installation. If the settings must be updated, access the Service Ports system settings by right-clicking on the Administration icon located in the system tray and selecting **Settings | Service Ports**.



Store and Forward

Port: Specify the TCP/IP port that the Store and Forward clients use to communicate with the Store and Forward service. The valid range is 1024 to 65535. The default is configured by the server.

Default: Click to populate this field with the default port number.

Tips:

- The default port is recommended unless there is a conflict with another server application using that port.
- The Store and Forward Service does not accept remote connections, so there should be no firewall implications associated with this port assignment.
- The permissions required to allow a user to enable SAF include project modification. Grant the user or group (possibly Anonymous Clients) the ability to modify the servermain project through the **User Manager**. ThingWorx users need the same access through the ThingWorx Interface Users group according to the procedure in **User Manager ThingWorx Interface Users**.

See Also**: **Project Properties ThingWorx

Security

Preferred Port: Specify a TCP/IP port that the Key Service can use to communicate within the server. The valid range is 1024 to 65535. The default is configured by the server. If the **Preferred Port** is unavailable or inappropriate for any reason, the service will attempt to secure an alternate port.

Default: Click to populate this field with the default port number.

Store and Forward Service

The Store and Forward Service allows different server components to store data on a local disk for a period of time. The service installs with components that require store and forward functionality. The Store and Forward service starts and stops automatically based on features that support store and forward.

See Also:
ThingWorx Project Properties
Store and Forward Configuration Settings
Store and Forward System Tags
ThingWorx Access Rights

Navigating the User Interface

The Configuration provides the general means of interacting with the server Runtime. While various plug-ins and drivers add buttons, menus, and icons; the standard interface elements are described below.

Title Bar

Displays the application name, when Configuration is connected to the Runtime, and the current Runtime project when applicable.

Menu Bar

File	Includes the project-level commands; such as Save, Open, Import, and Export.
Edit	Includes action commands; such as Copy, Paste, and New Channel.
View	Includes the display commands; such as which elements of the user interface are visible or hidden and the type of tree organization to display.
Tools	Includes the configuration commands; such as general options, connection settings, and Event Log Filters.
Runtime	Includes server connectivity commands; such as Connect..., Disconnect, and Reinitialize.

Help Includes commands to access the product documentation, by server, driver, or plug-in.

Button Bar

The standard buttons are described below. Plug-ins and drivers add, remove, enable, and disable buttons based on available functionality for the active items and view.

New Project: Initiates creation of a new project file to replace the active project. The project file defines the devices connected, their settings, and how they are grouped.

Open Project: Allows the user to browse for an existing project file to load, replacing the active project.


Save Project: Implements any recent changes and writes the active project file to disk.


Save As: Writes the active project with changes, such as to a new location or file name.


New Channel: Creates a new group or medium for data collection.


New Device: Defines a new hardware component or PLC for data collection.


New Tag Group: Defines a new collection of data points, or tags, that can be organized as a single unit.


New Tag: Defines a new data points for collection.


Bulk Tag Creation: Defines tags discovered in the target device or environment.


Duplicate Tag: Creates a copy of the selected tag.


Properties: Allows viewing and editing of parameters for the selected item.


Undo: Resets the value or item to its configuration prior to the most recent change.

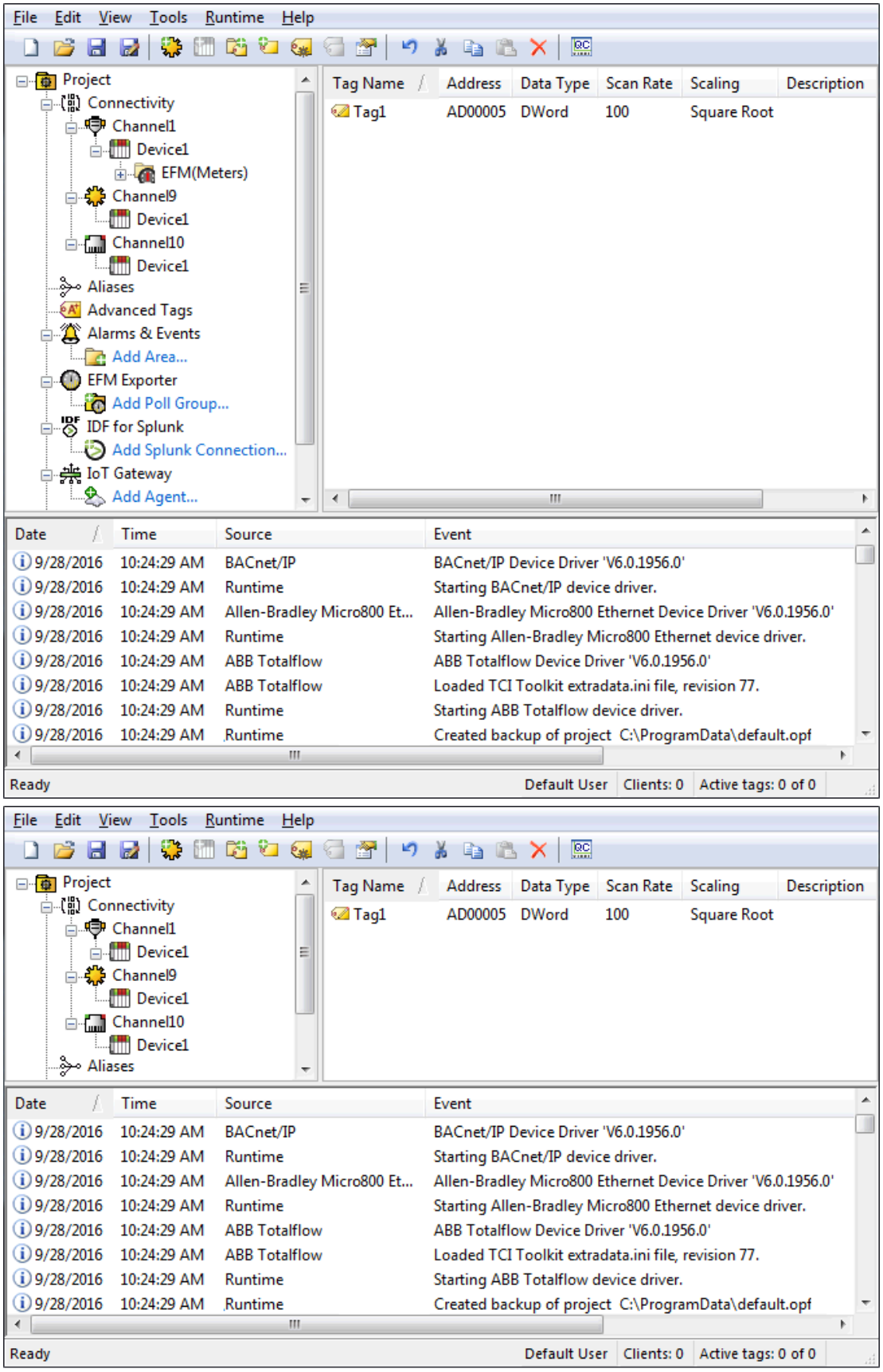

Cut: Removes the selected item and stores it on the clipboard.


Copy: Creates a duplicate of the selected item and stores it on the clipboard.


Paste: Inserts an item currently in the clipboard into the selected area.


Delete: Removes the selected item and / or its definition.


Quick Client: Runs the integrated client interface.



Project Tree View

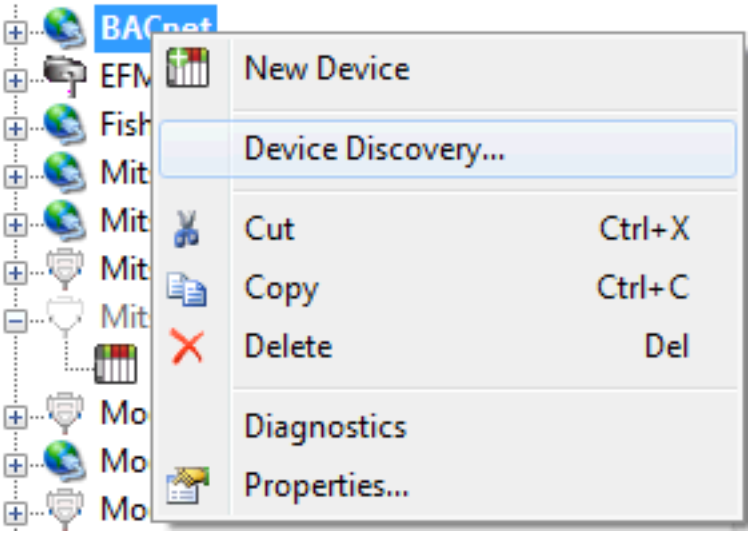
This view displays the current project contents, organization, and settings in a hierarchy view. The Project Tree View is designed as unified location for all aspect of the project. Nodes expand to allow detailed drill-down to the device, tag group, or tag level. Features and Plug-ins appear as nodes in the tree view to facilitate configuration work in one location. The major nodes of the tree are:

Project - where global settings for the active project are stored or updated.
Connectivity - where channels and devices are organized, right-click actions are available, and properties can be accessed for display in the Detail pane.
Aliases - where mappings to system resources, legacy paths, and complex routings can be given shorter, more user-friendly, or SCADA compatible names and shortcuts.
Advanced Tags - where operations or analysis can be built into tag processing and stored. This is a separate product Plug-in.
Alarms & Events - where system monitoring can be defined and managed. This is a separate product Plug-in.
DataLogger - where data can be organized and stored in an ODBC-compliant database. This is a separate product Plug-in.
EFM Exporter - where flow and trend data can be captured and coordinated. This is a separate product Plug-in.
IDF for Splunk - where data feeds into data management and data mining can be configured. This is a separate product Plug-in.
IoT Gateway - where connections to enterprise systems, monitoring, and analytics are managed. This is a separate product Plug-in.
Local Historian - where data collection, logging, storage, and retention is defined. This is a separate product Plug-in.
Scheduler - where data collection, publication, and bandwidth management can be coordinated. This is a separate product Plug-in.
SNMP Agent - where communication bridges into Information technology and SNMP protocols can be created. This is a separate product Plug-in.

In very large projects or if some features are used more than others, the tree can be customized through filtering. Hide or show tree nodes under the **View** menu.

The Project Tree provides a variety of appropriate options through a right-click menu. For example, devices and channels can be copied and pasted to start a new configuration based on existing choices and settings. The name is duplicated and a numbered added (that increments if many are pasted) to keep names unique. For drivers that support additional features, those are available on the right-click menu as well.

Device Discovery, for example, searches the reachable network for compatible devices and adds them automatically.



Detail View

This view displays one of several configuration selection options for the active project. Its information is related to the current Project Tree View.

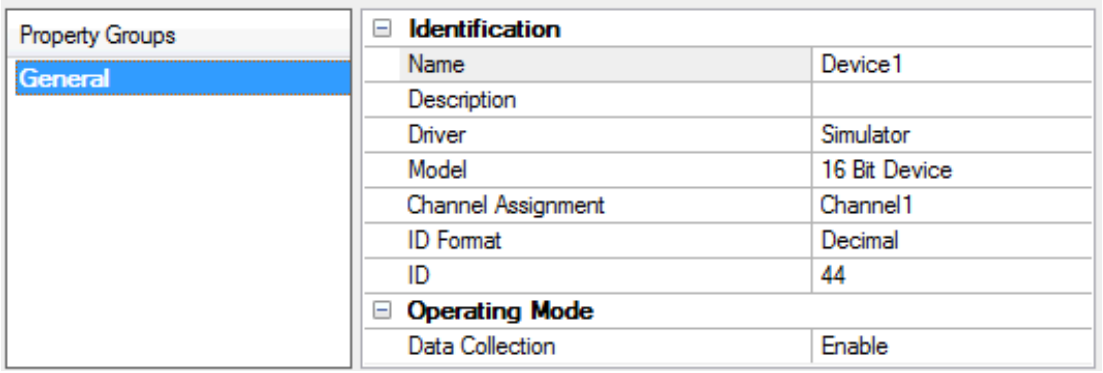
Note: When selecting a Project Tree View, the Detail View columns persist until a channel or device is chosen. At that time, the columns revert to displaying the device or tag information.

Event Log

This view, in the bottom pane, displays four types of recorded messages: General Information, Security Alerts, Warnings, and Errors from the server, drivers, or plug-ins. By default, log entries include the date, time, source, and event description. *For more information, see* ***Event Log Options***.

Property Editor

Some properties can be edited in the property editor. The standard buttons in the property editor operate as follows:



Defaults restores settings for the selected property group to their default values (both applied and pending changes).

Ok exits the property editor and implements all changes.

Cancel exits the property editor without implementing pending changes. Closing the property editor has the same effect.

Apply implements pending changes in all property groups.

Help opens Help for the selected property.

- Pending changes appear in bold until they are applied.

Status Bar

Displays the current status of the Configuration (Connecting, Ready, etc.) as well as mouse-over hints for the Menu Bar and Button Bar items.

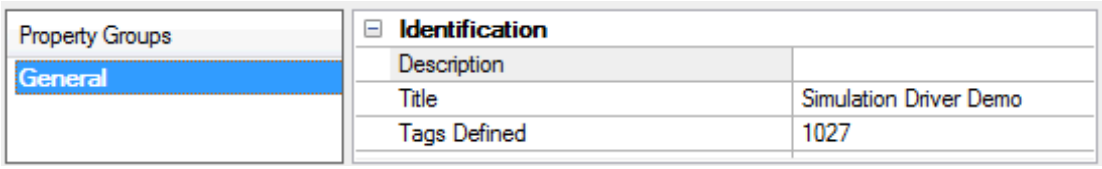
Project Properties

To access the Project Properties groups from the configuration, click **Edit | Project Properties**. For more information, select a link from the list below.

Project Properties — General
Project Properties — OPC DA
Project Properties — DDE
Project Properties — FastDDE/SuiteLink
Project Properties — iFIX PDB Settings
Project Properties — OPC UA
Project Properties — OPC AE
Project Properties — OPC HDA
Project Properties — ThingWorx

Project Properties — General

The general properties are used to attach a title and comment to a project for reference as well as manage security settings for the project. Although the Title field supports a string of up 64 characters, the Description field has no practical limit. Limiting the Description to the area available within the field, however, improves project load time.



Identification

Description: Enter an optional phrase to help identify this project in reports and monitoring systems.

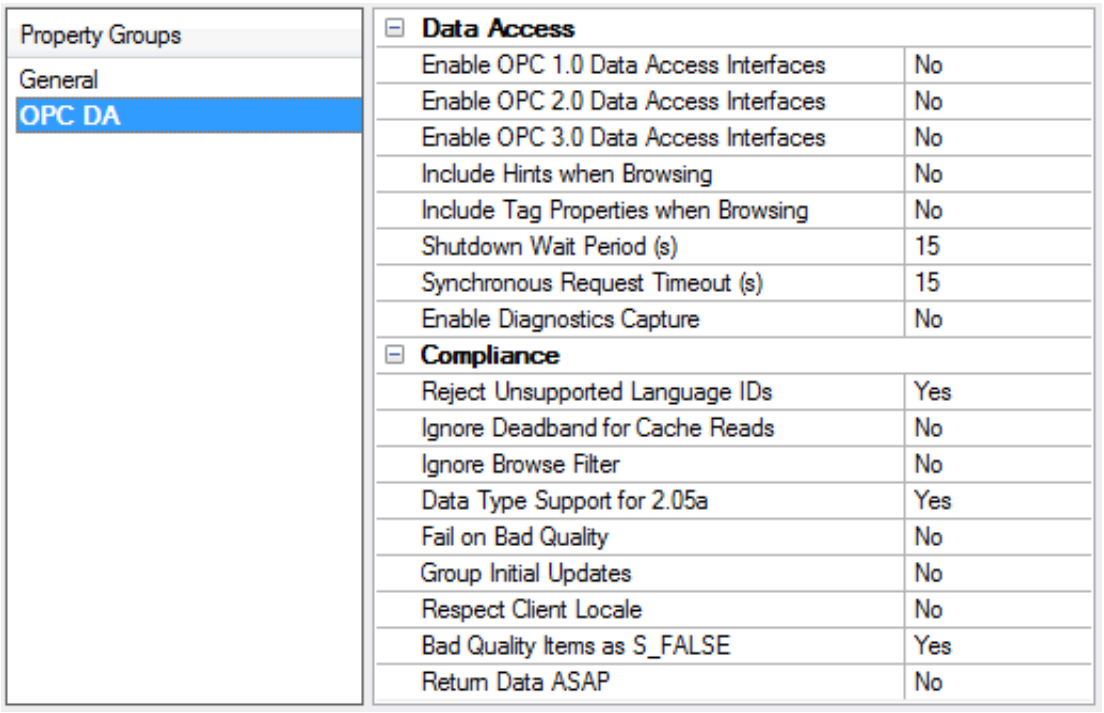
Title: Enter an optional word or phrase to identify this project in file names and reports.

Tags Defined: Verify that the tag count matches expectations of data collection for this project (and licensing, if applicable).

- The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — OPC DA

This 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 DA Compliance dialog allows users to customize operation of the server to better meet the needs of rare OPC clients. These options seldom need to be adjusted for the majority of OPC client applications.



Data Access

Enable OPC 1.0 Data Access Interfaces: Select Yes to allow the server to accept OPC client connections from OPC clients that support the 1.0 specification. The default setting is enabled.

Enable OPC 2.0 Data Access Interfaces: Select Yes to allow the server to accept OPC client connections from OPC clients that support the 2.0 specification. The default setting is enabled.

Enable OPC 3.0 Data Access Interfaces: Select Yes to allow the server to accept OPC client connections from OPC clients that support the 3.0 specification. The default setting is enabled.

Include Hints When Browsing: Select Yes to allow OPC client applications to browse the address formatting Hints available for each communications driver. The Hints provide a 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 not valid OPC tags. Some OPC client applications may try to add the Hint tags to their tag database. When this occurs, the client receives an error from the server. This is not a problem for most clients, although it can cause others to stop adding tags automatically or report errors. Prevent this by disabling Hints. The default setting is disabled (No).

Include Tag Properties When Browsing: Select Yes to allow OPC client applications to browse the tag properties available for each tag in the address space. The default setting is disabled.

Shutdown Wait Period: Specify how long the server waits for an OPC client to return from the server shutdown event. If the client application does not return within the timeout period, the server completes shutdown and exit. The valid range is 10 to 60 seconds. The default setting is 15 seconds.

Synchronous Request Timeout: Specify how long the server waits for a synchronous read operation to complete. If a synchronous operation is in progress and the timeout is exceeded, the server forces the operation to complete with a failure to the client. This prevents clients from locking up when using synchronous operations. The valid range is 5 to 60 seconds. The default setting is 15 seconds.

Note: Synchronous writes do not use this property setting; only reads / requests utilize this property.

Enable Diagnostics Capture: Select Yes to allow OPC diagnostics data to be logged to the Event Log service for storage (typically used for troubleshooting). The default setting is disabled (No).

*For more information on the OPC Data Access 1.0, 2.0, and 3.0 Custom Specifications, refer to the OPC Foundation website **www.opcfoundation.org**.*

Compliance

Reject Unsupported Language IDs: Select Yes to only allow Language IDs that are natively supported by the server. If the 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 rejects the group addition. To resolve this particular issue, disable the compliant feature to force the server to accept any Language ID.

Ignore Deadband for Cache Reads: Select Yes for the server to ignore the deadband setting on OPC groups added to the server. For some OPC clients, passing the correct value for deadband causes problems that may result in the OPC client (such as, having good data even though it does not appear to be updating frequently or at all). This condition is rare. As such, the selection should normally be left in its default disabled state.

Ignore Browse Filter: Select Yes for 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.

Data Type Support for 2.05a: Select Yes for the server to adhere to the data type requirements and expected behaviors for data type coercion that were added to the 2.05a specification.

Fail on Bad Quality: Select Yes for 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.

Group Initial Updates: Select Yes for the server to return all outstanding initial item updates in a single callback. When disabled, the server returns initial updates as they are available (which can result in multiple callbacks).

Enabling this may result in loss of buffered data when using drivers that support data buffering (Event Playback) for unsolicited device protocols. The compliance setting should be disabled if loss of buffered data is a concern.

Respect Client Locale: Select Yes for the server to use 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 - Two Hundred if

converted using English metrics, but would be One and Two-Tenths if converted using German metrics. If German software is running on an English OS, users need to determine how the comma is handled. This setting allows for such flexibility. By default, and due to historical implementation, the server respects the Locale ID of the operating system.

Bad Quality Item as S_FALSE: Select Yes for the server to return S_FALSE in the item error array for items without good quality. This setting defaults to Yes for existing projects that are set to full compliance and No for those that are not. When set to No, the legacy behavior of returning E_FAIL (0x80004005) occurs.

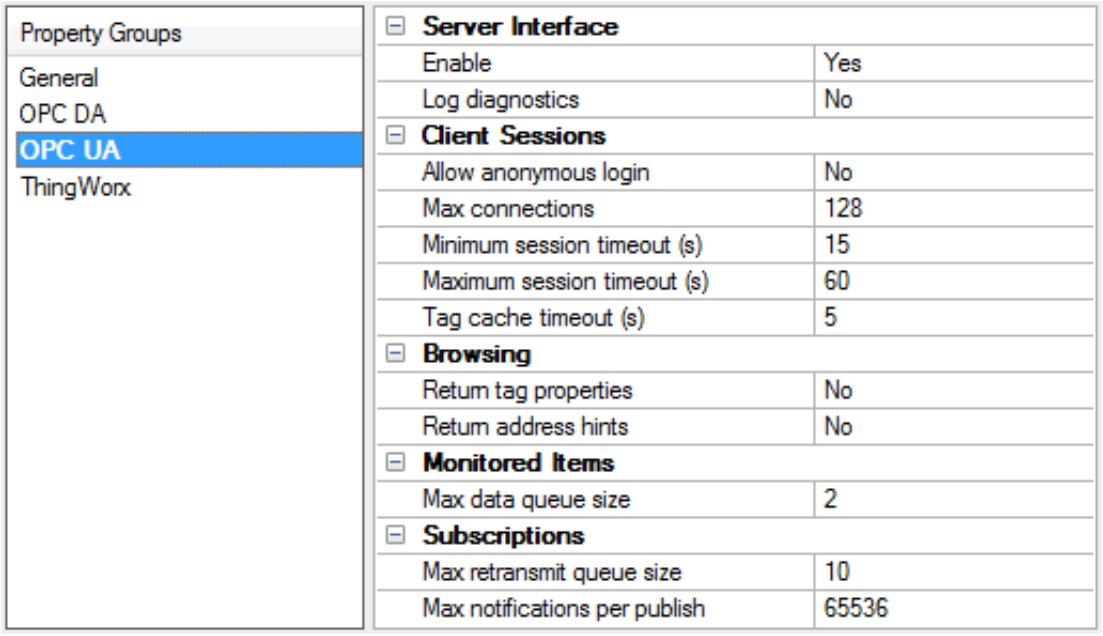
Return Data ASAP: Select Yes to enable all groups to update the client. When enabled, an active item that experiences a change in value or quality triggers a client update. The group update rate specified by the client is used to set the client requested scan rate for the items added to that group. The default setting is disabled.

The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — OPC UA

OPC Unified Architecture (UA) provides a platform independent interoperability standard. It is not a replacement for OPC Data Access (DA) technologies: for most industrial applications, UA complements or enhances an existing DA architecture. The OPC UA group displays the current OPC UA settings in the server.

Note: To change a setting, click in the specific property's second column. This invokes a drop-down menu that displays the options available.



Server Interface

Enable: When enabled, the UA server interface is initialized and accepts client connections. When disabled, the remaining properties on this page are disabled.

Log diagnostics: When enabled, OPC UA stack diagnostics are logged to the OPC Diagnostics Viewer. This should only be enabled for troubleshooting purposes.

Client Sessions

Allow anonymous login: This property specifies whether or not a user name and password are required to establish a connection. For security, the default setting is No to disallow anonymous access and require credentials.

Note: If this setting is disabled, users cannot login as the default user in the User Manager. Users can login as the Administrator provided that a password is set in the User Manager and is used to login.

When the client supplies a password on connect, the server decrypts the password using the encryption algorithm defined by the security policy of the endpoint.

Max. connections: specify the maximum number of supported connections. The valid range is 1 to 128. The default setting is 128.

Minimum session timeout: specify the UA client's minimum timeout limit for establishing a session. Values may be changed depending on the needs of the application. The default value is 15 seconds.

Maximum session timeout: specify the UA client's maximum timeout limit for establishing a session. Values may be changed depending on the needs of the application. The default value is 60 seconds.

Tag cache timeout: specify the tag cache timeout. The valid range is 0 to 60 seconds. The default setting is 5 seconds.

Note: This timeout controls how long a tag is cached after a UA client is done using it. In cases where UA clients read / write to unregistered tags at a set interval, users can improve performance by increasing the timeout. For example, if a client is reading an unregistered tag every 5 seconds, the tag cache timeout should be set to 6 seconds. Since the tag does not have to be recreated during each client request, performance improves.

Browsing

Return tag properties: Enable to allow UA client applications to browse the tag properties available for each tag in the address space. This setting is disabled by default.

Return address hints: Enable to allows UA client applications to browse the address formatting hints available for each item. Although the hints are not valid UA tags, certain UA client applications may try to add them to the tag database. When this occurs, the client receives an error from the server. This may cause the client to report errors or stop adding the tags automatically. To prevent this from occurring, make sure that this property is disabled. This setting is disabled by default.

Monitored Items

Max. Data Queue Size: specify the maximum number of data notifications to be queued for an item. The valid range is 1 to 100. The default setting is 2.

Note: The data queue is used when the monitored item's update rate is faster than the subscription's publish rate. For example, if the monitored item update rate is 1 second, and a subscription publishes every 10 seconds, then 10 data notifications are published for the item every 10 seconds. Because queuing data consumes memory, this value should be limited when memory is a concern.

Subscriptions

Max. retransmit queue size: specify the maximum number of publishes to be queued per subscription. The valid range is 1 to 100. A value of zero disables retransmits. The default setting is 0.
Note: Subscription publish events are queued and retransmitted at the client's request. Because queuing consumes memory, this value should be limited when memory is a concern.

Max. notifications per publish: specify the maximum number of notifications per publish. The valid range is 1 to 65536. The default setting is 65536.
Note: This value may affect the connection's performance by limiting the size of the packets sent from the server to the client. In general, large values should be used for high-bandwidth connections and small values should be used for low-bandwidth connections.

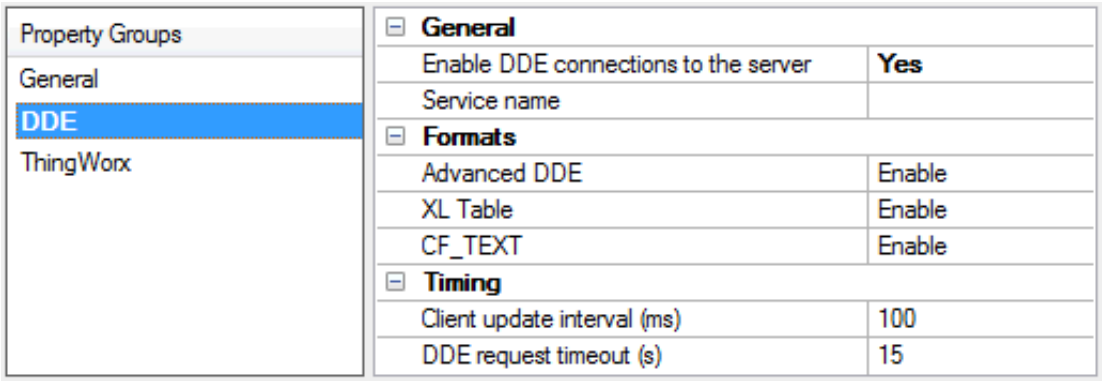
The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — DDE

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

For the DDE interface to connect with the server, the Runtime must be allowed to interact with the desktop. For more information, refer to ***How To... Allow Desktop Interactions****.*

To access the DDE server settings through the Configuration, click **Edit | Project Properties** and locate the **DDE** properties. Its properties can be used to tailor the DDE operation to fit the application's needs.



General

Enable DDE connections to the server: This property determines whether the DDE server portion of the server is enabled or disabled. If DDE operation is disabled, the server does not respond to any request for DDE data. If intending to use the server only as an OPC server, users may want to disable DDE operation. Doing so can increase the data security and improve overall server performance. DDE is disabled by default.
See Also: How To... Use DDE with the Server

Service name: This property allows users to change how the server appears as an application name to DDE clients. This name is initially set to allow compatibility with the previous versions of the server. If users need to replace an existing DDE server however, the server's service name can be changed to match the DDE server being replaced. The service name allows a string of 1 to 32 characters to be entered.

Formats

This property allows users to configure the DDE format to provide to client applications. Choose to enable or disable **Advanced DDE**, **XL Table**, and **CF_Text**. All three formats are enabled by default. This is particularly useful when users experience problems connecting a DDE client application to the server: each of the DDE formats can be disabled to isolate a specific format for testing purposes.

Note: Every DDE-aware application must support CF_Text at a minimum.

Timing

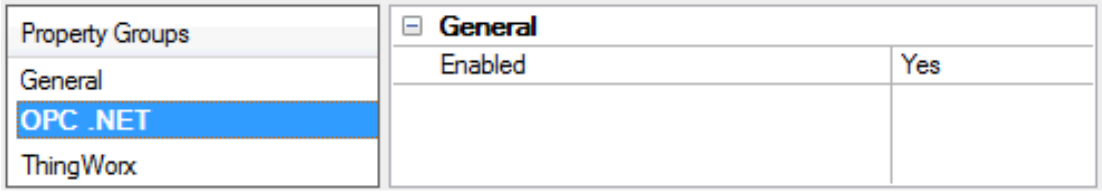
Client update interval: This interval setting is used to batch up DDE data so that it can be transferred to client applications. When using a DDE format performance gains only come when large blocks of server data can be sent in a single DDE response. To improve the ability of the server to gather a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

DDE request timeout: This property is used to configure a timeout for the completion of DDE request. If a DDE client request (either a read or write operation) on the server cannot be completed within the specified timeout, an error is returned to the DDE client. The valid range is 1 to 30 seconds. The default setting is 15 seconds.

Note: The server Runtime may need to be reinitialized for changes to take effect.

Project Properties — OPC .NET

To access the OPC .NET server settings through the Configuration, click **Edit | Project Properties** and select the **OPC .NET** tab.



Enabled: When enabled, the OPC .NET Wrapper is initialized and accept client connections.

Tips:

1. The OPC .NET Wrapper runs as a System Service called "xi_server_runtime.exe". It is only started when the server starts and the option described above is enabled. Unlike OPC DA, clients cannot launch the server.
2. To use and install OPC .NET, Microsoft .NET 3.5 must be present on the machine prior to server installation.

The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — OPC AE

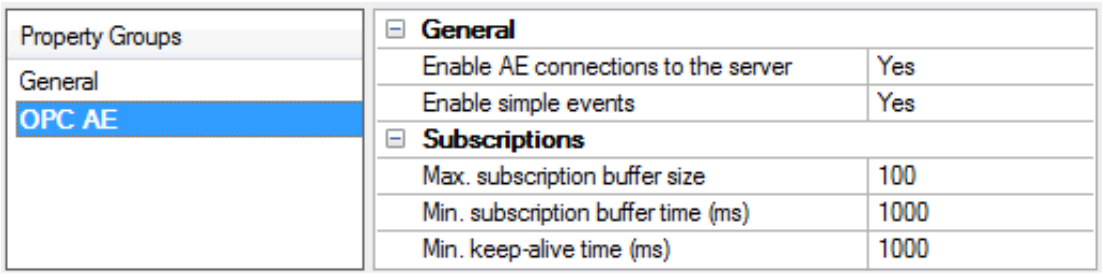
Events are used to signal an occurrence in the server and are similar to data updates in OPC Data Access. The OPC AE functionality allows users to receive Simple Events from the server, including system startup

and shutdown messages, warnings, errors, and so forth. These events are displayed in the Event Log.

The OPC AE group is used to specify a number of project-level AE settings. Changes made to these settings take effect after all A&E clients disconnect from the server.

The Alarms & Events plug-in allows Alarms & Events (A&E) clients to receive A&E data from the OPC server. It is used to convert OPC server events into A&E format and to create custom alarms using OPC server tags.

For more information, contact the OPC vendor.



General

Enable AE Connections to the Server: This property turns the OPC AE server on and off.

Enable Simple Events: When enabled, simple events are made available to clients. When disabled, the events are sent. The default setting is enabled.

Subscriptions

Max. Subscription Buffer Size: Specify the maximum number of events sent to a client in one send call. The range is 0 to 1000. The default setting is 100. 0 means there is no limit.

Min. Subscription Buffer Time: Specify the minimum time between send calls to a client. The supported range is 100 to 60000 milliseconds. The default setting is 1000 milliseconds.

Min. Keep-Alive Time: Specify the minimum amount of time between keep-alive messages sent from the server to the client. The supported range is 100 to 60000 milliseconds. The default setting is 1000 milliseconds.

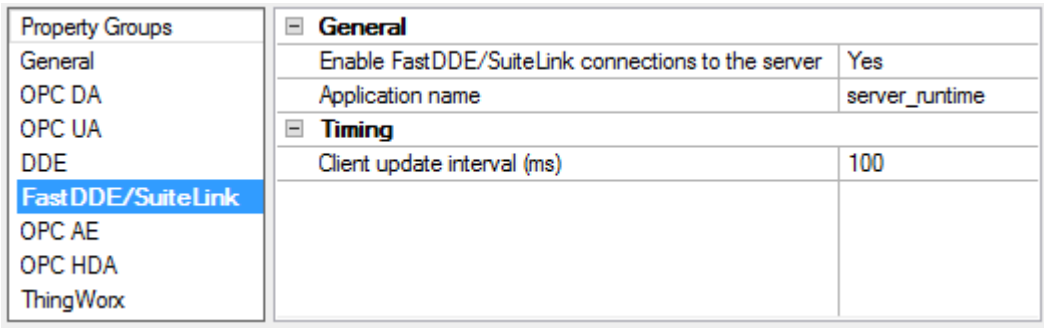
The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — FastDDE / SuiteLink

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

For the FastDDE interface to connect with the server, the Runtime must be allowed to interact with the desktop. For more information, refer to ***How To... Allow Desktop Interactions***.

Note: For proper FastDDE / SuiteLink operation (and for this tab to be available in Project Properties), the Wonderware FS2000 Common Components or the InTouch Runtime Component version 8.0 or higher must be installed on the PC.



Enable FastDDE / SuiteLink connections to the server: This property enables or disables support of the client / server protocols. When a Wonderware product is installed on the PC, this setting is available to enable. If the FastDDE / SuiteLink operation is disabled, the server does not respond to any request for FastDDE or SuiteLink data.

Tip: For better performance and security, it is recommended that this setting be disabled if the server is only used for OPC connectivity.

Application Name: icon to open the application's name. The default setting is "server_runtime".

Note: This name may be customized to suit specific end-user needs. For example, users that select "Remove and Redirect" during the installation must change this setting to "servermain" for certain FactorySuite applications to work without modification.

Client Update Interval (ms): icon to open how often new data is sent to FastDDE / SuiteLink client applications. The range is 20 to 32000 milliseconds. The default setting is 100 milliseconds. The timer allows 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.

Notes:

1. The update rate applies to how often data is sent to the client application, not how often data is read from the device. The scan rate can be used to adjust how fast or slow the server acquires data from an attached device. *For more information, refer to* ***Tag Properties — General***.
2. The server Runtime may have to be reinitialized for changes to take effect.

The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — iFIX PDB Settings

The iFIX PDB Settings dialog contains properties that allow users to adjust the behavior between the processing of the iFIX process database (PDB) tags and the server tags. To access, click **Edit | Project Properties**.

Note: The iFIX PDB Settings are only displayed in Project Properties if iFIX is installed on the computer.

In some cases, the Process Mode must be set to System Service for the iFIX PDB interface to work with the Runtime. For more information, refer to ***Process Modes***.

Property Groups		
General	**General**	
iFIX PDB Settings	Enable connectivity to iFIX PDB	Yes
	Enable latched data	No
	Enable update per poll	No
	Use iFIX startup configuration file	Yes
	Use unconfirmed updates	No
	Timing	
	PDB-to-server request timeout (s)	5
	Deactive tags on PDB read inactivity	Yes
	Inactivity timeout (s)	15

Note: It is recommended that users keep the default values for each field. Users should also ensure that the settings meet the application's requirements.

General

Enable connectivity to iFIX PDB: Enable or disable support of the client/server protocols. If the iFIX PDB operation is disabled, the server does not respond to any request for iFIX PDB data. For better performance and security when the server is only being used for OPC connectivity, disable this property.

Enable latched data: Normally, the iFIX application's data links display a series of question marks (such as "????") if a communication failure has occurred. Users may want to have a value displayed at all times, however. By enabling latched data, the last value successfully read is preserved on the screen. The default setting is enabled.

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

Enable update per poll: When enabled, the server delivers the current value, quality, and timestamp to iFIX every time that the driver polls the device. When disabled, the server only delivers an update to iFIX when it determines the value or the quality has changed. The default setting is disabled.

Note: This setting is dynamic, meaning that the server immediately begins to deliver updates to the iFIX client at the device scan rate after the option is applied.

Use iFIX startup configuration file: Enable to create this file through iFIX to contains all items accessed by the iFIX client. It automatically starts scanning items before iFIX requests item data. The default setting is enabled.

See Also: *Project Startup for iFIX Applications*

Use unconfirmed updates Controls how the server updates local cache for iFIX following writes via the NIO interface. With the default setting (disabled), the server does not update local cache until the value has been confirmed via a read. For the majority of applications, the default setting provides the best user experience from the standpoint of data integrity. For applications leveraging iFIX Easy Database Access (EDA), users may wish to enable unconfirmed updates to update the local cache for iFIX immediately with the attempted write value.

Note: From a data integrity perspective, use of unconfirmed updates can result in a false indication of write success and inaccurate data displayed in iFIX. Another consequence of using unconfirmed updates is that the data displayed in iFIX can "flicker" due to the temporary unconfirmed update (write value attempted) followed by a confirmed update (actual value read for the item).

Timing

PDB-to server request timeout(s): Specify the amount of time that the iFIX PDB waits for a response from an add, remove, read, or write request before timing out. Once timed out, the request is discarded on behalf of the server. A timeout can occur if the server is busy processing other requests or if the server has lost communications with iFIX PDB. In the case of lost communications, the iFIX PDB automatically re-establishes communications with the server so that successive timeouts do not occur. The valid range is 5 to 60 seconds. The default setting is 5 seconds.

Deactivate tags on PDB read inactivity: Direct the server to automatically deactivate tags that have not been read by iFIX for the time period specified. This reduces unnecessary polling of the process hardware. When enabled, the server reads its list of tags every 15 seconds and deactivates any that are idle. If iFIX has not performed a read request of a tag for the time period specified, the tag is considered idle. 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 reactivates the tag and resumes polling the hardware. The default setting is disabled. Once this feature is enabled, however, it becomes applied to all projects. Users may specify an idle time in a range from 15 to 607999 (15 seconds to 1 week).

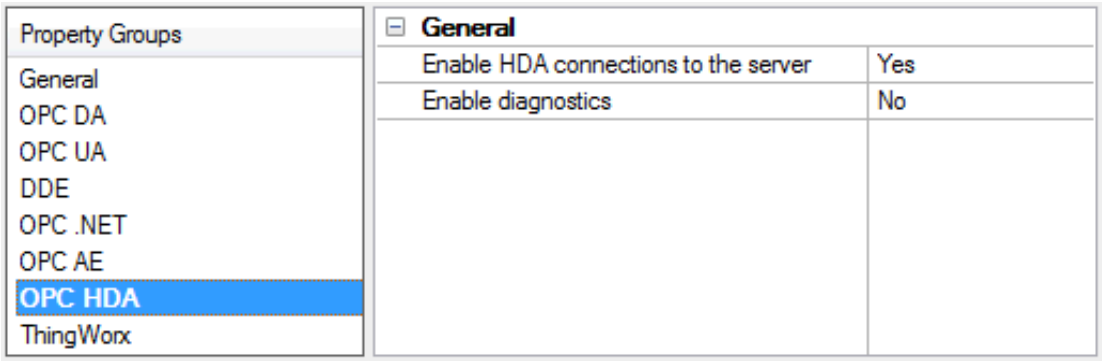
- This feature is meant to be used with Register tags only and can cause non-register tags to go off scan. To avoid this situation when using this feature, set the inactivity timer greater than the longest scan time configured in the iFIX database.

Inactivity timeout(s): Specify the amount of time that the iFIX PDB waits for activity before timing out. In the case of lost communications, the iFIX PDB automatically re-establishes communications with the server so that successive timeouts do not occur. The valid range is 5 to 60 seconds. The default setting is 5 seconds.

- The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — OPC HDA

To access the OPC HDA server settings through the Configuration, click **Edit** | **Project Properties** and expand the **OPC HDA** group.



Enable HDA connections to the server: When enabled, HDA clients can connect to the HDA server that is exposed by this server. When disabled, client HDA connections are disabled. These settings may be applied without restarting the Runtime; however, although the server does not drop connected clients, it does not accept new client connections either. The default setting is enabled.

Enable Diagnostics: When enabled, this option allows OPC HDA data to be captured and logged to the Event Log service for storage. The default setting is disabled.

Note: Enabling diagnostics has negative effect on the server runtime performance. For more information on event logging, refer to **OPC Diagnostics Viewer**.

The **Defaults** button restores the settings to the default / pre-set values.

Project Properties — ThingWorx

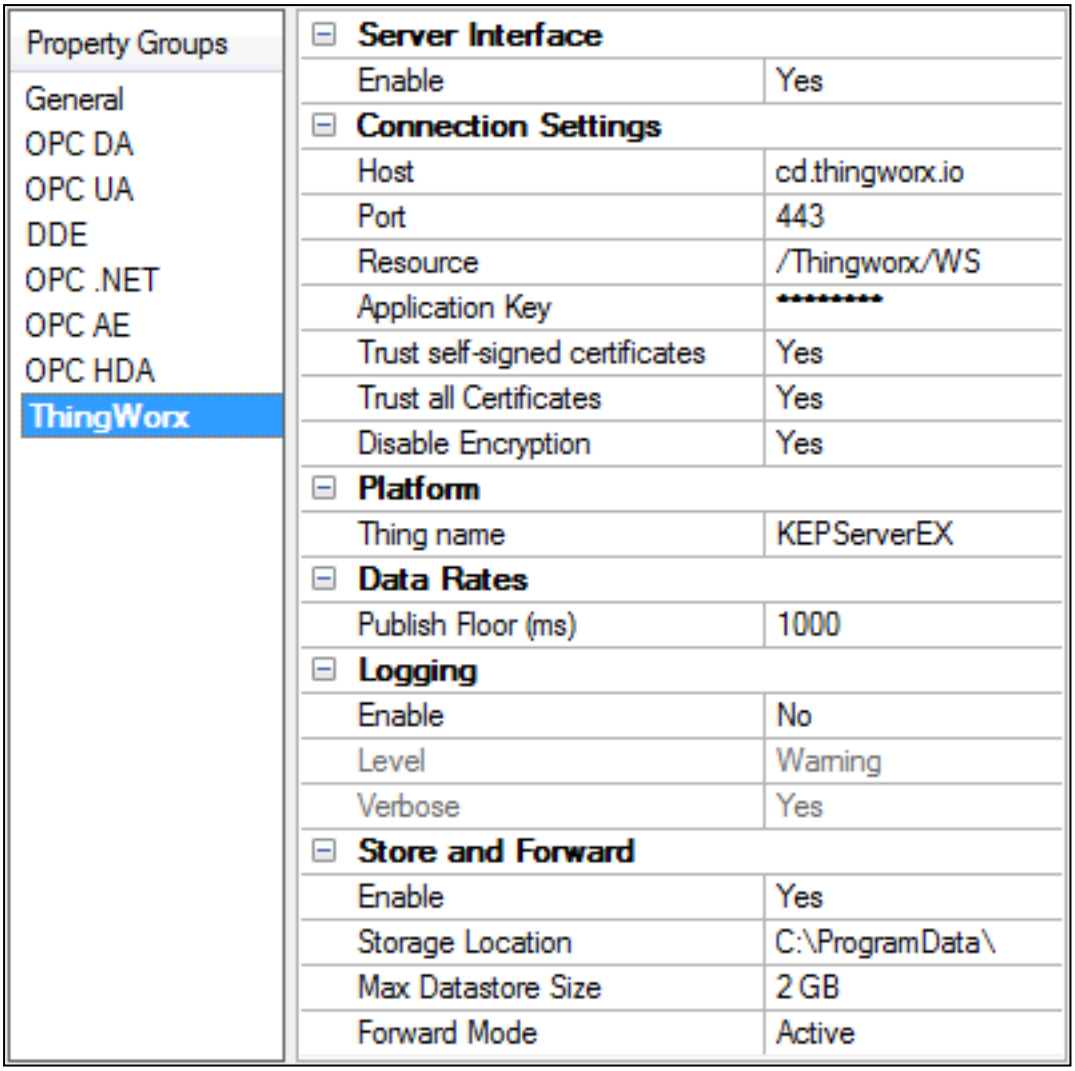
Support for the ThingWorx Native Interface simplifies the task of connecting with a ThingWorx Platform, while simultaneously allowing OPC and other connectivity as needed.

Before configuring the ThingWorx Native Interface, create a Thing in the ThingWorx platform with the "Industrial Gateway" Thing Template. A Thing name which represents its data source is recommended. Once the Thing is created, configure the OPC server to connect to the ThingWorx platform using the Thing name. The new connection will auto-bind to this Thing.

Once the connection to the ThingWorx platform is made, use the Industrial Connections selection from the left hand menu to select tags from the newly created server instance. These tags may be selected and bound to new Things directly in the ThingWorx Composer.
Refer to the ThingWorx Composer documentation for more information.

Cautions:

- Any tags with an array data type must be configured with the Always push type in the ThingWorx Platform. A push threshold set to value change will fail to publish updates to the platform.
- While most of the native interfaces function in a client server configuration, the ThingWorx Native Interface acts more like a client, as it creates an outbound connection to the ThingWorx platform. This allows the ThingWorx Native Interface to connect to a remote ThingWorx Platform using standard ports and protocols without the need to create unusual firewall or routing rules. As long as the ThingWorx Composer is reachable in a browser from the machine hosting the OPC server, then the server should be able to pass data to that platform through the Native interface.
- As noted in ThingWorx documentation, configuration of a ThingWorx Application Key is crucial to providing a secured environment. The Application Key should provide the appropriate privileges to allow the proper exchange of data between the server instance and the ThingWorx Platform.



Server Interface

Enable: Select Yes for the ThingWorx Native interface to attempt connection with the information provided.

Legacy Mode: When connecting with the ThingWorx NextGen Composer (v7.4 or newer), Legacy Mode should be Disabled (default). To connect with ThingWorx versions 7.3x or older, select Enable to force the use of the RemoteKEPServerEXThing template shape. To properly use this mode, import the RemoteKEPServerEXThing extension on the ThingWorx instance. Then the proper ThingShapes and service definitions work with this native interface.

Legacy Mode is ONLY used for existing KEPServerEX V6.0 ThingWorx native interface projects and ThingWorx Composer (v7.3x or older).

Connection Settings

Host: Specify the IP address or DNS name of the ThingWorx server.

Port: Specify the number of the TCP port used by the ThingWorx server.

Resource: Specify the URL endpoint on the ThingWorx server.

Application key: Enter or paste in the authentication string for connecting to the ThingWorx server.
Caution: Do NOT set this property using the Config API Service with enabled HTTP in production mode; it would compromise security.

Trust self-signed certificates: Select No for maximum security. Select Yes to accept self-signed certificates during development.
Caution: Do NOT set this to Yes in a production environment as it would compromise security.

Trust all certificates: Select No for maximum security. Select Yes and the TLS library does not validate the server certificate.
Caution: Do NOT set this to Yes in a production environment as it would compromise security.

Disable encryption: Indicate if connections to a non-SSL-secured ThingWorx platform are allowed.
Caution: Do NOT set this to Yes in a production environment as it would compromise security.

Platform

Thing name: Enter the name of the entity (remote thing) on the ThingWorx server that represents this data source. Use the OPC server template to create the remote thing.
Note: The Thing Name must match the name of the Industrial Gateway thing exactly (case sensitive).

Data Rates

Publish floor: Specify the minimum rate at which updates are sent to the platform. Zero sends updates as often as possible.

Logging

Enable: Select Yes to activate advanced logging of the ThingWorx native interface. This logging is sent to the server event log. This logging may cause the event log to fill up quickly, it is recommended that the logging remain disabled when not troubleshooting.

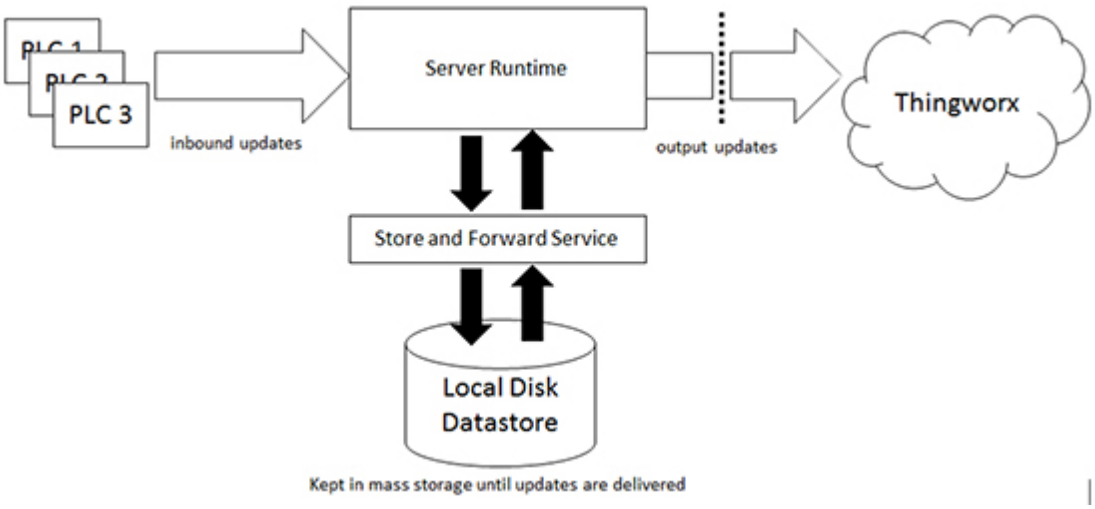
Level: Select the severity of logging to be sent to the event log. Trace includes all messages from the native ThingWorx interface.

Verbose: Select **Yes** to make the error messages as detailed as possible.

***See Also**: Event Log, Event Log Options*

Store and Forward

The ThingWorx Native Interface supports a Store and Forward datastore to persist property updates when the industrial server loses connectivity to the ThingWorx Platform. When enabled, Store and Forward persists all incoming property updates to disk until the ThingWorx Native Interface receives confirmation from the platform that the update has been received. If connection to the platform is lost, all updates are stored and maintained on disk until either the disk where updates are being stored comes within 500 MB of being full or the size of stored updates exceeds the maximum size specified - whichever comes first. Once the datastore or disk is full, incoming updates are dropped until enough space is available to store the incoming data.



***See Also**: Fill Rate Example*

Store and Forward Properties

Enable: Select **Yes** to save data to a local disk directory to avoid data loss during connection interruption or heavy data transfers. Enabling this setting allows data to be queued, then pushed forward once a connection is established and data receipt has been confirmed.

Storage Location: Enter or browse to the fully qualified path to the directory where data should be cached.
Note: The ThingWorx Native Interface queues updates in memory when the Store and Forward datastore cannot be initialized. The server automatically retries until a datastore can be initialized. *Refer to the event log for specific failure information.*

Max. Datastore Size: Select the maximum number of megabytes or gigabytes the data is allowed to reach before purging. The available datastore sizes range from 128 MB to 16 GB.

Forward Mode: Select a method to determine which updates are sent to ThingWorx when the connection is restored. In situations that require active monitoring of production data without any data loss when disconnected from the platform, it is possible to store and forward upon reconnect or to schedule forwarding the stored updates for a time when production is not being actively monitored (for example, during production downtime). Options include Active and On Hold:

- **Active Mode** - When the Forward Mode is set to Active, stored property updates are sent in the order they were received until the ThingWorx Platform has received all updates. Updates are then sent to the platform in real time. Property updates can be delayed due to the first In, first out nature of property update forwarding when many updates are collected during a ThingWorx platform disconnect.

- **On Hold Mode** - When the Forward Mode is set to On Hold, only the latest updates are sent to the platform after recovering from a disconnect. This ensures that ThingWorx applications that are actively monitoring production and get the freshest data available. When production is not being actively monitored, the mode can be set to Active to start forwarding the older updates that were stored while the server was disconnected from the platform. The industrial server buffers up to 25,000 property updates in memory before storing them to disk. Once the 25,000 update limit is reached, the property updates are pushed to disk and held until the Forward Mode is set to Active. This allows the industrial server to prioritize the most recent 25,000 updates when the connection to the ThingWorx

platform is restored, hold on to updates so they they're not lost, and forward them later. New updates are dropped when the datastore size limit is reached, or the disk is filled past the 500MB limit, whichever occurs first. The in-memory buffer is only typically filled when the connection to the ThingWorx platform is lost; however, this can also occur when property updates are collected at a rate faster than can be forwarded to the platform.

Store and Forward Considerations

- Store and Forward configuration is currently supported in the industrial server's configuration tool or with through the Configuration API.
- Store and Forward is disabled by default and must be enabled in industrial server's Project Properties or through the Configuration API.
- It is not necessary to configure Store and Forward from the ThingWorx Platform. However, to store the forwarded updates to the ThingWorx Platform, it is necessary to configure a Value Stream and enable logging for any properties for which a history is desired.
- When the datastore path configuration (defined in Storage Location setting) is modified, the existing datastore remains on disk. If the datastore path configuration is restored, updates associated with the current project are forwarded to the platform.
- Changes to Store and Forward properties do not require the platform connection to be reinitialized. The Thingworx interface continues collecting updates while applying the changes.

The Store and Forward path is validated both at configuration and runtime, and must comply with the following:

- Must be between 3 and 256 characters
- Must not contain any characters or symbols forbidden by the system
- Must be an absolute path (beginning with a drive letter)
- Must not refer to a network resource (mapped drive* or UNC share)
- Must not refer to removable media such as a USB drive*
  ** refers to items which are only validated at runtime*

Store and Forward Status and Monitoring can be accessed in the following ways:

- The industrial server's **Store and Forward Tags**
- The industrial server's Event Log

Store and Forward Operational Considerations

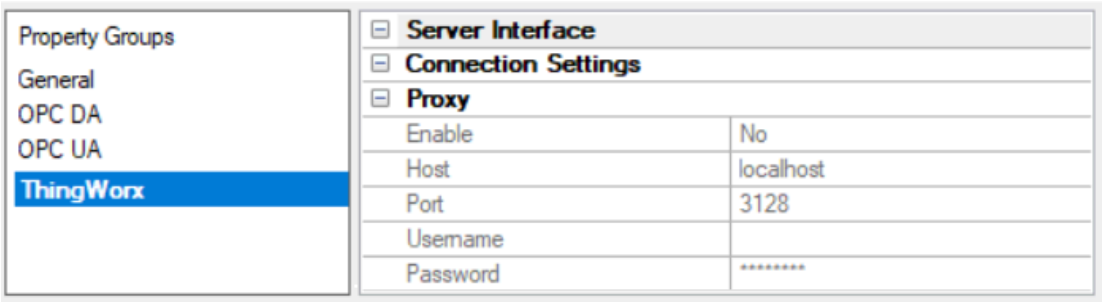
- The reliability requirements of Store and Forward introduce a small decrease in performance when enabled as all updates are routed through a disk buffer before being sent to the ThingWorx Platform and the ThingWorx Native Interface waits to receive confirmation that the platform has received the most recent set of updates before sending the next set.
- Stored updates persist across server restarts.
- Make sure all stored updates are forwarded before a software upgrade because updates cannot be preserved across major / minor server upgrades.

Proxy Properties

The server leverages the Thingworx CSDK to allow communicating with the ThingWorx platform through a proxy server. The following authentication options are supported:

- No authentication
- Basic authentication

- Digest authentication
- NTLM



Enable: Select **Yes** to connect to the ThingWorx platform through a proxy server.

Host: Specify the IP address or DNS name of the proxy server to connect.

Port: Specify the number of the TCP port used to connect to the proxy server.

Username: Specify the user account name to connect to the proxy server and authenticate.

Password: Enter the password authentication string for connecting to the ThingWorx server as the user specified.

Caution: Do NOT set this property using the Config API Service with enabled HTTP in production mode; it would compromise security.

Operation — Legacy Mode

Legacy Mode is ONLY used for existing KEPServerEX V6.0 ThingWorx native interface projects and ThingWorx Composer (v7.3x or older).

Once the interface is configured with valid information and enabled, KEPServerEX establishes the connection with the ThingWorx platform. A new "Thing" must be created on the ThingWorx platform that is identical to the Thing Name used in the project properties. The RemoteKEPServerEXThing Thing extension must be imported into the ThingWorx platform and used to create the new thing being integrated. Once created on the platform, the below services may be called via the platform.

See the ThingWorx help guide for instructions on importing an extension.

Note: The RemoteKEPServerEXThing Extension may be found in the ThingWorx marketplace online or in the following folder:

- For 64-bit Windows: C:\Program Files (x86)\Kepware\KEPServerEX 5\Utilities\KEPServerEX Extension for the ThingWorx IoT Platform
- For 32-bit Windows: C:\Program Files\Kepware\KEPServerEX 5\Utilities\KEPServerEX Extension for the ThingWorx IoT Platform

BrowseGroups: Returns a list of channels devices and tag groups. It can accept an input of a filter and a path. Filters are the same as OPC filters including char lists. Paths are channel and device lists such as “Channel1.Device1”.

BrowseItems: Returns a list of tags under a specific path. It can accept a filter and a path. Filters are the same as OPC filters, including char lists. Paths are channel and device lists, such as “Channel1.Device1”.

AddItems: Allows a tag to be subscribed to and added as a property under the Thing. An infotable is required to call this service. The infotable must contain the following information. ReadOnly: Boolean, ScanRateMS (optional): integer, Description (optional): String, BaseType: ThingWorx Data type, SourceType: KEPServer data type, Persistent: Boolean, Logged: Boolean, Source: Tag address (channel.device.tag), Name: Local name of the tag.

Note: Per ThingWorx restrictions; spaces, special characters, and leading numbers are not allowed in the name field. It is acceptable to include hyphens, underscores, and numbers within or at the end of the name.

RemoveItems: Removes the subscription from a tag. An infotable is required to call this service. The infotable must contain the following information. Name: Local name of the tag. Optionally, enable ForceRemove Boolean to unbind the tag from a property without deleting the property.

GetConfiguration: Returns an infotable containing the scan rate in milliseconds, the server description, and the publish floor in milliseconds.

SetConfiguration: Sets the scan rate in milliseconds, the server description, and the publish floor in milliseconds. Any values left blank retain their current setting.

Notes:

1. When using the date data type, values coming from KEPServerEX are interpreted as UTC. Allow for the proper time zone offset.
2. Adding items to the server is synchronous and is completed quickly. Autobinding properties in the platform can take some time and happens in the background after the items have been added. An event is fired when the autobinding process is complete with the results of the binding processes.
3. Calling RemoveItem only removes the binding from that property. Once RemoveItem is called, re-bind a different tag to that property, programmatically or through the Composer, or delete that property in the Composer. These properties appear in the Composer with a blank "Remote Property Name" until they are re-bound or deleted.
4. When adding multiple items at once. if two or more items are configured to use the same ThingWorx name, the entire addItem call will fail. Please make sure all properties have unique names.
5. The commands in the Example may be performed via cURL or other POST/PUT/GET utility. These are examples only; refer to the ThingWorx documentation for interacting with all of the services available.

The **Defaults** button restores the settings to the default / pre-set values.

Store and Forward — Fill Rate Example

The **Max Datastore Size** and data type of the updates being stored need to be considered to determine maximum update count and fill rate. The table below describes update count limits and fill rates for several data types scenarios assuming a maximum datastore size of 128 MB and 1 update / second.

Data Type	Maximum Update Count	Fill Rate (bytes / second)
Word / Short	5817792	22
DWord / Long / Float	5333076	24

Data Type	Maximum Update Count	Fill Rate (bytes / second)
Double	4571321	28
String (length = 10)	3764743	34

Using the following equation and information from the table above the fill rate for a given project can be determined by summing the fill rates that correspond to the tag data types of the project:

Overall Fill Rate =
ScanRate(seconds) *
PropertyCount(Bool) * FillRate (Bool)+
PropertyCount(Word) * FillRate (Word) +
PropertyCount(Word) * FillRate (Short) +
PropertyCount(DWord) * FillRate (DWord) +
PropertyCount(Word) * FillRate (Long) +
PropertyCount(Word) * FillRate (Float) +
PropertyCount(Double)* FillRate (Double)+
PropertyCount(String) * FillRate (String)

The table below describes the fill rate and offline time before data loss for a sample project consisting of 500 Word properties, 500 DWord properties, 10 String properties, and 100 Double properties for several scan rates assuming a maximum datastore size of 128 MB.

Per-Property Scan Rate (milliseconds)	Fill Rate (bytes / second)	Offline Time (minutes)
10000	2614	816
1000	26140	81
250	104560	20

Store and Forward — System Tags

System tags provide datastore status information and allow server clients to manage the updates. These system tags are only available to server clients when Store and Forward is enabled. The tags are located under the _ThingWorx group folder at the same level as the _System folder in the client browsing tree.

Tag	Class	Datatype	Description
_StoreAndForwardEnabled	Read / Write	Boolean	This tag allows Store and Forward to be turned On or Off. When this tag is set False, Store and Forward is disabled. When Store and Forward is disabled all Datastore related system tags report a default value equivalent to 0. **Note**: • The configuration is not always indicative of the enable / disable state of Store and Forward. Use the _StoreAndForwardEnabledStatus

Tag	Class	Datatype	Description
			system tag to get the configuration in use. For example, when an error occurs that prevents Store and Forward, the _StoreAndForwardEnabledStatus returns 0.
_StoreAndForwardEnabledStatus	Read / Write	Boolean	This tag indicates whether or not the interface is using Store and Forward.
_DatastoreDiskFull	Read Only	Boolean	This tag indicates whether the disk in use by the datastore has been filled past the 500 MB threshold required for updates to be stored.
_DatastoreFull	Read Only	Boolean	This tag indicates whether the datastore has reached the configured **Max Datastore Size** that can be used to store updates.
_StoredUpdateCount	Read Only	DWord	This tag indicates the number of updates in the datastore. **Notes**: • A non-zero value does not indicate that the ThingWorx connection has been lost because updates are always routed through the datastore when Store and Forward is enabled. • During steady-state operation this number is expected to fluctuate; however, the stored update count should not increase over time. This behavior indicates that more data is being collected than can be delivered to the ThingWorx platform.
_DeleteStoredData	Read / Write	Boolean	This tag can be used to delete the contents of a datastore. Writing any value to this tag deletes all stored updates in the Store and Forward datastore.
_DatastoreCurrentSizeMB	Read Only	Double	This tag reports the amount of space (in MiB) used by all updates currently on disk
_DatastoreRemainingSpaceMB	Read Only	Double	This tag reports the amount of space (in MiB) remaining in the datastore available to store updates. This is based on the Max Datastore Size property, and not available disk space. *For disk space remaining, see the _DatastoreUsableDiskSpace tag.*
_DatastoreUseableDiskSpaceMB	Read Only	Double	This tag reports the amount of space (in MiB) available to store updates on the disk

Tag	Class	Datatype	Description
			where the datastore is located. Store and Forward uses a safety buffer of 500MiB so as to not fill the entire disk. This system tag takes this safety buffer into account for its calculation. This tag does not reflect the amount of space remaining in the datastore as specified by the user. *See _DatastoreSizeRemaining for that information.*
_DatastoreAttachError	Read Only	Boolean	This tag indicates an error has occurred that prevents use of Store and Forward. When the tag value is True an error has occurred. Refer to the server event log for information regarding this error. *See Possible Cause/Solutions to resolve the error that prevents the Store and Forward datastore from being used.*
_DroppedUpdates	Read Only	Long	This tag reports the total number of dropped updates since the ThingWorx interface started. When the value reaches 2,147,483,647 that value will rollover to 0. The value resets to 0 when the ThingWorx connection is reinitialized.
_ForwardMode	Read/Write	DWord	This tag reports the current Forward Mode configuration of the ThingWorx Native Interface. The tag supports writes to change the configured mode. Valid values include 0 for Active and 1 for On Hold. All other write values are ignored. **Note**: • The configuration is not always indicative of the Forward Mode in use. Use the __ForwardModeStatus system tag to get the mode in use. For example, when an error occurs that prevents Store and Forward, the __ForwardModeStutus returns a blank.
_ForwardModeStatus	Read Only	String	This tag reports the current Forward Mode in use by the native interface. Possible values include Active and On Hold. The system tag returns a blank string when Store and Forward is not in use.

***See Also**: ThingWorx Interface Users for controlling access to the ThingWorx platform and related data transfer.*

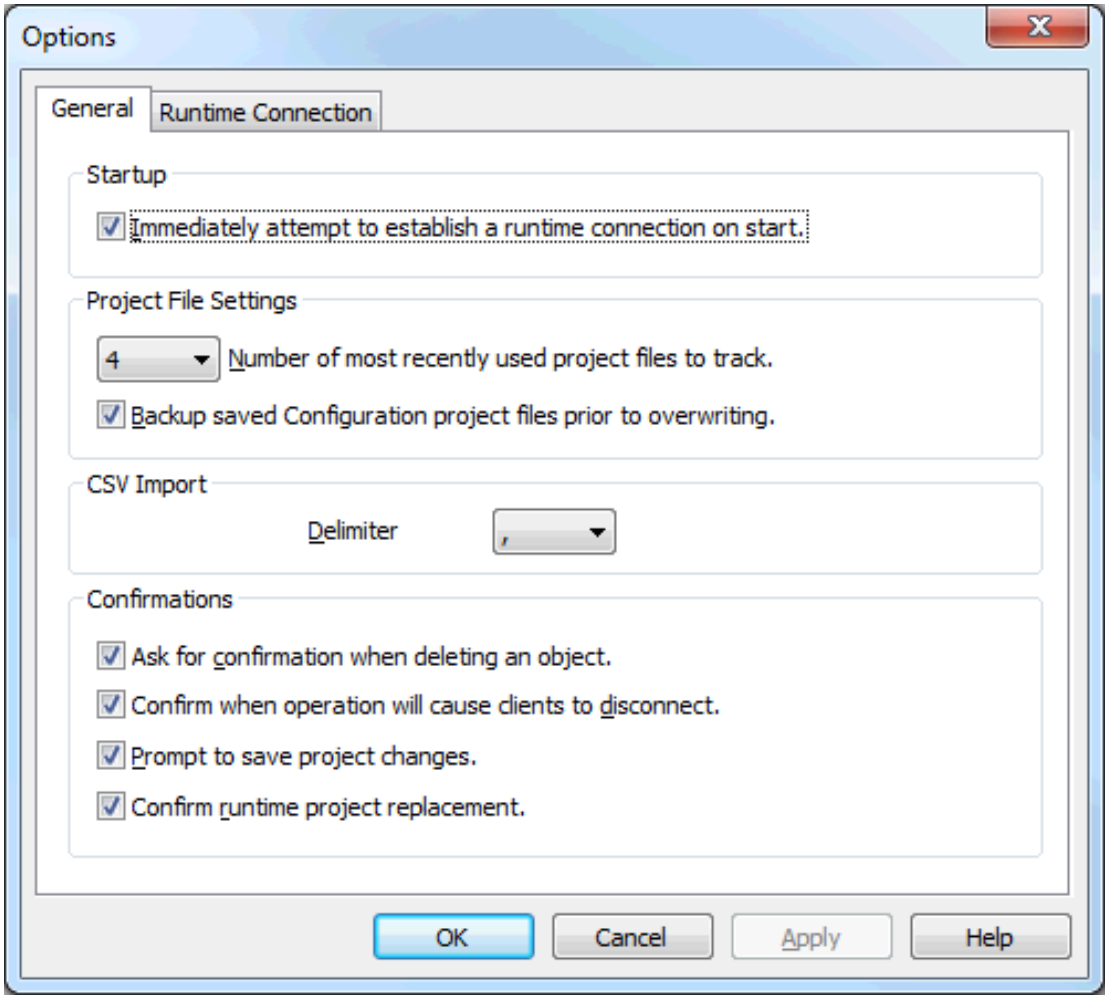
Server Options

To access the Server Options groups from the configuration, click **Tools | Options**. These settings are configured on an individual basis. For more information, select a link from the list below.

Options - General
Options - Runtime Connection

Options — General

This dialog is used to specify general server options (such as when to establish a connection with the Runtime, when to back up saved Configuration project files, and what conditions invoke warning pop-ups).



Immediately attempt to establish a Runtime connection on start: icon to open whether the configuration tool connects to the Runtime when started. When disabled, users must connect manually. The default setting is enabled.

Number of recently used project files to track: Select the number of project files presented in the **MRU (Most Recently Used)** list of projects. The valid range is 1 to 16. The default setting is 8.

Backup saved Configuration project files prior to overwriting: When enabled, the system automatically makes a backup copy of the last saved Configuration project before it is overwritten with a new project file. The backup file's name and location are displayed in the Event Log.

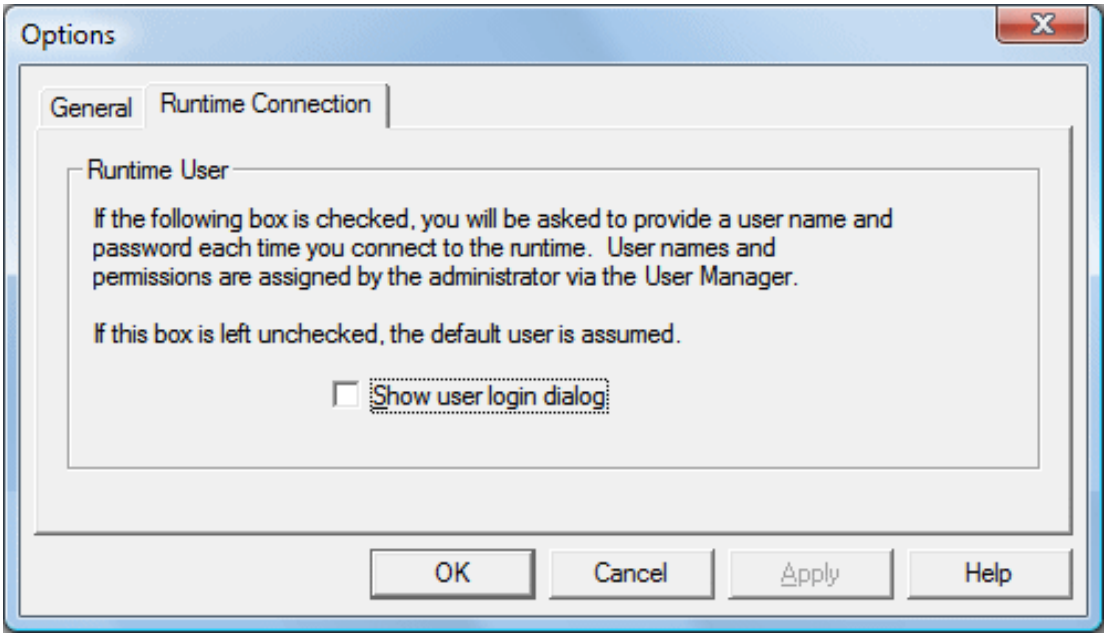
CSV Import: The **Delimiter** setting specifies the Comma Separated Variable (CSV) that the server uses to import and export tag data in a CSV file. Options include comma and semicolon. The default setting is comma. *For more information, refer to* ***Tag Management***.

Confirmations: Select the conditions that force the Configuration to present warning pop-ups to an operator.

- **Ask for confirmation when deleting an object**: When enabled, all Configuration delete operations invoke a warning popup that requires confirmation before the delete operation can be completed.
- **Confirm when operation will cause clients to disconnect**: When enabled, all Configuration operations that would cause client applications to be disconnected from the server invoke a warning popup. This popup requires confirmation before the disconnect sequence can be initiated.
- **Prompt to save project changes**: When enabled, the Configuration invokes a popup if the server is being shut down while the project has outstanding changes.
- **Confirm Runtime project replacement**: When enabled, this option warns that the project can be opened and edited offline while the Configuration is connected to the Runtime.

Options — Runtime Connection

This dialog is used to specify how connections to the Runtime are managed.



Show user login dialog: When enabled, a valid user name and password are required before the Configuration can be connected to the Runtime for project editing. The default setting is disabled.

Note: User names and permissions are assigned by the administrator. *For more information, refer to* ***Settings - User Manager***.

Components and Concepts

For more information on a specific server component, select a link from the list below.

What is a Channel?

What is a Channel?

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

Before adding devices to a project, users must define the channel to be used when communicating with 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.

Adding a Channel

Channels are added using the channel wizard, which guide users through the channel definition process. To start, users are prompted for a logical name to assign the channel. This name must be unique among all channels and devices defined in the project. *For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*

Users are prompted for the device driver to be used. A list box is presented that displays all of the device drivers currently installed in the system. All serial drivers can be used with multiple channels in the same project.

Note: For hardware card drivers, refer to the driver's help documentation to determine the ability to use with multiple channels in a single project. For information on how to determine the number of supported channels, refer to **Server Summary Information**.

Users are prompted for the specific communication parameters to be used. Multiple channels cannot share identical communication parameters; for example, two serial drivers cannot use COM1. For the correct communication parameters of a particular device, refer to both the manufacturer's and the driver's help documentation.

Note: 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).

The channel wizard finishes with a summary of the new channel.

Removing a Channel

To remove a channel from the project, select the desired channel and press the **Delete** key. Alternatively, select **Edit | Delete** from the Edit menu or toolbar.

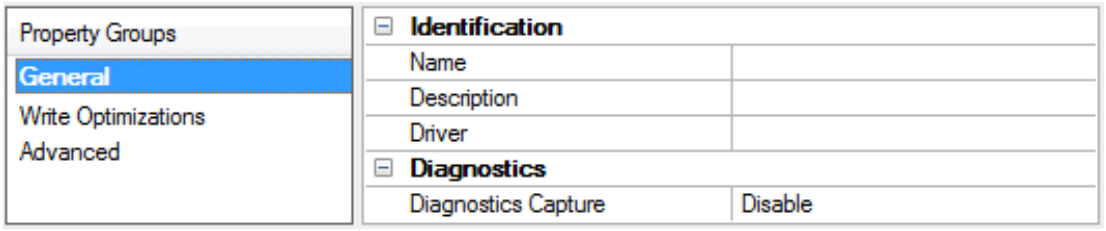
Displaying Channel Properties

To display the channel properties of a specific channel, select the channel and click **Edit | Properties** from the Edit menu or toolbar.

See Also: **Channel Properties — General**

Channel Properties — General

This server supports the use of simultaneous multiple communications drivers. Each protocol or driver used in a server project is called a channel. A server project may consist of many channels with the same communications driver or with unique communications drivers. A channel acts as the basic building block of an OPC link. This group is used to specify general channel properties, such as the identification attributes and operating mode.



Identification

Name: User-defined identity of this channel. In each server project, each channel name must be unique. Although names can be up to 256 characters, some client applications have a limited display window when browsing the OPC server's tag space. The channel name is part of the OPC browser information. The property is required for creating a channel.

For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.

Description: User-defined information about this channel.

Many of these properties, including Description, have an associated system tag.

Driver: Selected protocol / driver for this channel. This property specifies the device driver that was selected during channel creation. It is a disabled setting in the channel properties. The property is required for creating a channel.

Note: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-acquire using the old channel name, the item is not accepted. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to server features.

Diagnostics

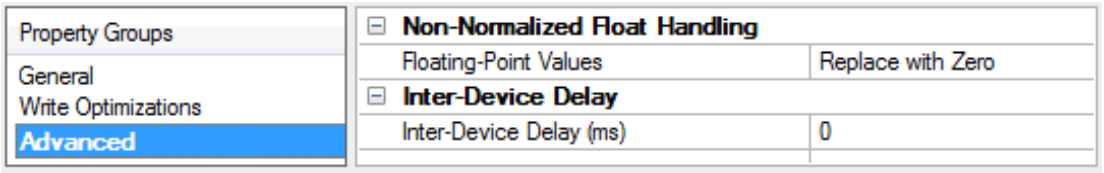
Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.

Note: This property is not available if the driver does not support diagnostics.

For more information, refer to "Communication Diagnostics" in the server help.

Channel Properties — Advanced

This group is used to specify advanced channel properties. Not all drivers support all properties; so the Advanced group does not appear for those devices.



Non-Normalized Float Handling: A non-normalized value is defined as Infinity, Not-a-Number (NaN), or as a Denormalized Number. The default is Replace with Zero. Drivers that have native float handling may default to Unmodified. Non-normalized float handling allows users to specify how a driver handles non-normalized IEEE-754 floating point data. Descriptions of the options are as follows:

- **Replace with Zero**: This option allows a driver to replace non-normalized IEEE-754 floating point values with zero before being transferred to clients.
- **Unmodified**: This option allows a driver to transfer IEEE-754 denormalized, normalized, non-number, and infinity values to clients without any conversion or changes.

Note: This property is not available if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM data is not affected by this setting.

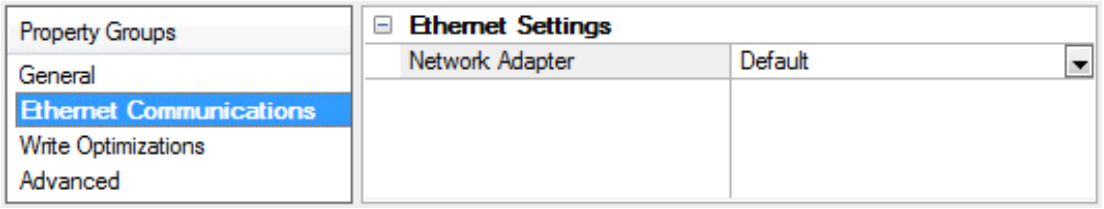
For more information on the floating point values, refer to "How To ... Work with Non-Normalized Floating Point Values" in the server help.

Inter-Device Delay: Specify the amount of time the communications channel waits to send new requests to the next device after data is received from the current device on the same channel. Zero (0) disables the delay.

Note: This property is not available for all drivers, models, and dependent settings.

Channel Properties — Ethernet Communications

Ethernet Communication can be used to communicate with devices.



Ethernet Settings

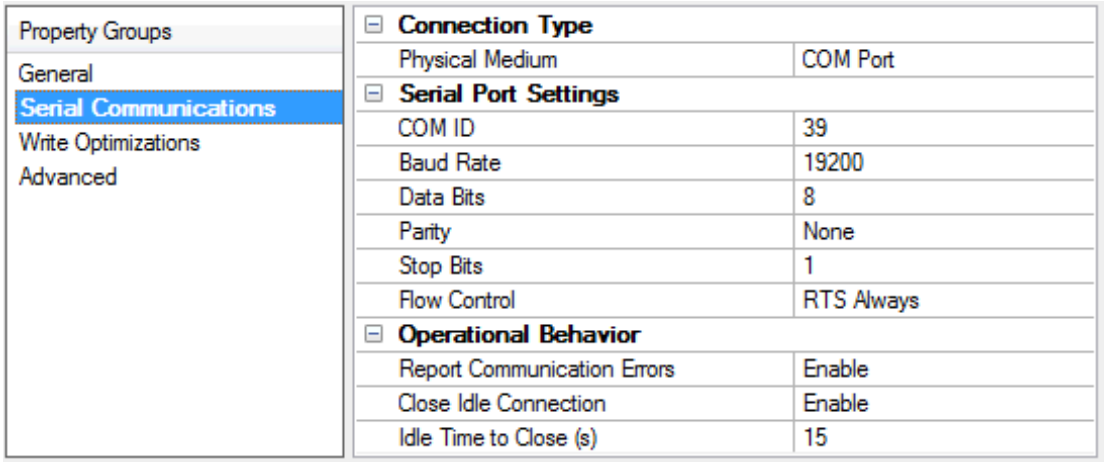
Network Adapter: Specify the network adapter to bind. When left blank or Default is selected, the operating system selects the default adapter.

Channel Properties — Serial Communications

Serial communication properties are available to serial drivers and vary depending on the driver, connection type, and options selected. Below is a superset of the possible properties.

Click to jump to one of the sections: Connection Type, Serial Port Settings or Ethernet Settings, and Operational Behavior.

Note: With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features, as changes made to these properties can temporarily disrupt communications.



Connection Type

Physical Medium: Choose the type of hardware device for data communications. Options include COM Port, None, Modem, and Ethernet Encapsulation. The default is COM Port.

- **None**: Select None to indicate there is no physical connection, which displays the Operation with no Communications section.
- **COM Port**: Select Com Port to display and configure the Serial Port Settings section.
- **Modem**: Select Modem if phone lines are used for communications, which are configured in the Modem Settings section.
- **Ethernet Encap.**: Select if Ethernet Encapsulation is used for communications, which displays the Ethernet Settings section.
- **Shared**: Verify the connection is correctly identified as sharing the current configuration with another channel. This is a read-only property.

Serial Port Settings

COM ID: Specify the Communications ID to be used when communicating with devices assigned to the channel. The valid range is 1 to 999. The default is 1.

Baud Rate: Specify the baud rate to be used to configure the selected communications port.

Data Bits: Specify the number of data bits per data word. Options include 5, 6, 7, or 8.

Parity: Specify the type of parity for the data. Options include Odd, Even, or None.

Stop Bits: Specify the number of stop bits per data word. Options include 1 or 2.

Flow Control: Select how the RTS and DTR control lines are utilized. Flow control is required to communicate with some serial devices. Options are:

- **None**: This option does not toggle or assert control lines.
- **DTR**: This option asserts the DTR line when the communications port is opened and remains on.
- **RTS**: This option specifies that the RTS line is high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line is low. This is normally used with RS232/RS485 converter hardware.
- **RTS, DTR**: This option is a combination of DTR and RTS.
- **RTS Always**: This option asserts the RTS line when the communication port is opened and remains on.
- **RTS Manual**: This option asserts the RTS line based on the timing properties entered for RTS Line Control. It is only available when the driver supports manual RTS line control (or when the properties are shared and at least one of the channels belongs to a driver that provides this support).
  RTS Manual adds an **RTS Line Control** property with options as follows:
  - **Raise**: This property specifies the amount of time that the RTS line is raised prior to data transmission. The valid range is 0 to 9999 milliseconds. The default is 10 milliseconds.
  - **Drop**: This property specifies the amount of time that the RTS line remains high after data transmission. The valid range is 0 to 9999 milliseconds. The default is 10 milliseconds.
  - **Poll Delay**: This property specifies the amount of time that polling for communications is delayed. The valid range is 0 to 9999. The default is 10 milliseconds.

Tip: When using two-wire RS-485, "echoes" may occur on the communication lines. Since this communication does not support echo suppression, it is recommended that echoes be disabled or a RS-485 converter be used.

Operational Behavior

- **Report Communication Errors**: Enable or disable reporting of low-level communications errors. When enabled, low-level errors are posted to the Event Log as they occur. When disabled, these same errors are not posted even though normal request failures are. The default is Enable.
- **Close Idle Connection**: Choose to close the connection when there are no longer any tags being referenced by a client on the channel. The default is Enable.
- **Idle Time to Close**: Specify the amount of time that the server waits once all tags have been removed before closing the COM port. The default is 15 seconds.

Ethernet Settings

Note: Not all serial drivers support Ethernet Encapsulation. If this group does not appear, the functionality is not supported.

Ethernet Encapsulation provides communication with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port that converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted, users can connect standard devices that support serial communications to the terminal server. The terminal server's serial port must be properly configured to match the requirements of the serial device to which it is attached. *For more information, refer to "How To... Use Ethernet Encapsulation" in the server help.*

- **Network Adapter**: Indicate a network adapter to bind for Ethernet devices in this channel. Choose a network adapter to bind to or allow the OS to select the default.
  Specific drivers may display additional Ethernet Encapsulation properties. For more information, refer to Channel Properties — Ethernet Encapsulation.

Modem Settings

- **Modem**: Specify the installed modem to be used for communications.
- **Connect Timeout**: Specify the amount of time to wait for connections to be established before failing a read or write. The default is 60 seconds.
- **Modem Properties**: Configure the modem hardware. When clicked, it opens vendor-specific modem properties.
- **Auto-Dial**: Enables the automatic dialing of entries in the Phonebook. The default is Disable. *For more information, refer to "Modem Auto-Dial" in the server help.*
- **Report Communication Errors**: Enable or disable reporting of low-level communications errors. When enabled, low-level errors are posted to the Event Log as they occur. When disabled, these same errors are not posted even though normal request failures are. The default is Enable.
- **Close Idle Connection**: Choose to close the modem connection when there are no longer any tags being referenced by a client on the channel. The default is Enable.
- **Idle Time to Close**: Specify the amount of time that the server waits once all tags have been removed before closing the modem connection. The default is 15 seconds.

Operation with no Communications

- **Read Processing**: Select the action to be taken when an explicit device read is requested. Options include Ignore and Fail. Ignore does nothing; Fail provides the client with an update that indicates failure. The default setting is Ignore.

Channel Properties — Ethernet Encapsulation

Ethernet Encapsulation can be used over wireless network connections (such as 802.11b and CDPD packet networks) and has also been developed to support a wide range of serial devices. With a terminal server device, users can place RS-232 and RS-485 devices throughout the plant while still allowing a single localized PC to access the remotely-mounted devices. Ethernet Encapsulation also allows an individual network IP address to be assigned to devices as needed. Multiple terminal servers provide users access to hundreds of serial devices from a single PC. One channel can be defined to use the local PC serial port while another channel can be defined to use Ethernet Encapsulation.

Note: These properties are only available to serial drivers. The properties displayed depend on the selected communications driver and supported functionality.

Network Adapter: This property specifies the network adapter.

Device Address: This property specifies 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 has its own IP address.

Port: This property configures the Ethernet port that used when connecting to a remote terminal server. The valid range is 1 to 65535, with some numbers reserved. The default is 2101.

Protocol: This property specifies TCP/IP or UDP communications, and depends on the nature of the terminal server being used. The default is TCP/IP. For more information on the protocol available, refer to the terminal server's help documentation.

Important: The Ethernet Encapsulation mode is completely transparent to the actual serial communications driver. Users must configure the remaining device properties as if they were connecting to the device directly on the local PC serial port.

Connect Timeout: This property specifies the amount of time that is required to establish a socket connection for a remote device to be adjusted. In many cases, the connection time to a device can take longer

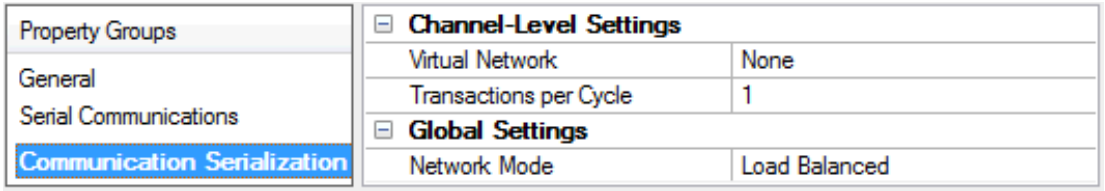
than a normal communications request to that same device. The valid range is 1 to 999 seconds. The default is 3 seconds.

Note: With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties.

Channel Properties — Communication Serialization

The server's multi-threading architecture allows channels to communicate with devices in parallel. Although this is efficient, communication can be serialized in cases with physical network restrictions (such as Ethernet radios). Communication serialization limits communication to one channel at a time within a virtual network.

The term "virtual network" describes a collection of channels and associated devices that use the same pipeline for communications. For example, the pipeline of an Ethernet radio is the master radio. All channels using the same master radio associate with the same virtual network. Channels are allowed to communicate each in turn, in a "round-robin" manner. By default, a channel can process one transaction before handing communications off to another channel. A transaction can include one or more tags. If the controlling channel contains a device that is not responding to a request, the channel cannot release control until the transaction times out. This results in data update delays for the other channels in the virtual network.



Channel-Level Settings

Virtual Network This property specifies the channel's mode of communication serialization. Options include None and Network 1 - Network 500. The default is None. Descriptions of the options are as follows:

- **None**: This option disables communication serialization for the channel.
- **Network 1 - Network 500**: This option specifies the virtual network to which the channel is assigned.

Transactions per Cycle This property specifies the number of single blocked/non-blocked read/write transactions that can occur on the channel. When a channel is given the opportunity to communicate, this is the number of transactions attempted. The valid range is 1 to 99. The default is 1.

Global Settings

- **Network Mode**: This property is used to control how channel communication is delegated. In **Load Balanced** mode, each channel is given the opportunity to communicate in turn, one at a time. In **Priority** mode, channels are given the opportunity to communicate according to the following rules (highest to lowest priority):

- Channels with pending writes have the highest priority.
- Channels with pending explicit reads (through internal plug-ins or external client interfaces) are prioritized based on the read's priority.
- Scanned reads and other periodic events (driver specific).

The default is Load Balanced and affects *all* virtual networks and channels.

Devices that rely on unsolicited responses should not be placed in a virtual network. In situations where communications must be serialized, it is recommended that Auto-Demotion be enabled.

Due to differences in the way that drivers read and write data (such as in single, blocked, or non-blocked transactions); the application's Transactions per cycle property may need to be adjusted. When doing so, consider the following factors:

- How many tags must be read from each channel?
- How often is data written to each channel?
- Is the channel using a serial or Ethernet driver?
- Does the driver read tags in separate requests, or are multiple tags read in a block?
- Have the device's Timing properties (such as Request timeout and Fail after *x* successive timeouts) been optimized for the virtual network's communication medium?

Channel Properties — Network Interface

With Ethernet Encapsulation, virtually all drivers currently available support some form of Ethernet communications. Some form of a network interface is used, whether for a natively Ethernet-based driver or a serial driver configured for Ethernet Encapsulation. In most cases, that interface takes the form of a Network Interface Card (NIC). For a PC that has networking installed, this usually means that a single NIC is installed that provides a connection to either the IT or plant floor network (or both).

This configuration works well for typical network configurations and loading. Problems may arise if data needs to be received from an Ethernet device at a regular interval, however. If the plant floor network is mixed with the IT network, a large batch file transfer could completely disrupt the interval of the plant floor data. The most common way to deal with this issue is to install a second NIC in the PC. One NIC can be used for accessing the IT network while the other NIC accesses the plant floor data. Although this may sound reasonable, problems may occur when trying to separate the networks. When using multiple NICs, users must determine the bind order. The bind order determines what NIC is used to access different portions of the Ethernet network. In many cases, bind settings can be managed using the operating system's tools.

When there is a clear separation between the types of protocols and services that are used on each NIC card, the bind order can be created by the operating system. If there isn't a clear way to select a specific bind order, users may find that the Ethernet device connection is being routed to the wrong network. In this case, the network interface shown below can be used to select a specific NIC card to use with the Ethernet driver. The network interface selection can be used to select a specific NIC card based on either the NIC name or its currently assigned IP address. This list of available NICs includes either unique NIC cards or NICs that have multiple IP assigned to them. The selection displays any WAN connections are active (such as a dial up connection).

Note: This property is only available to Ethernet drivers.

By selecting a specific NIC interface, users can force the driver to send all Ethernet communication through the specified NIC. When a NIC is selected, the normal operating system bind order is bypassed completely. This ensures that users have control over how the network operates and eliminates any guesswork.

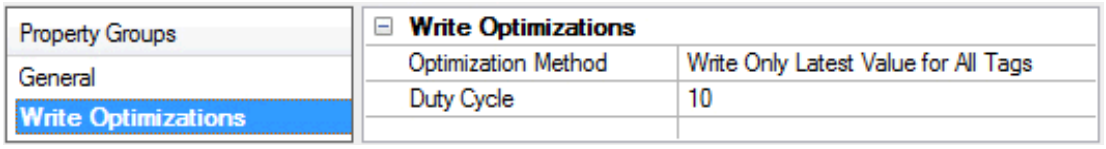
The selections displayed in the Network Adapter drop-down menu depend on the network configuration settings, the number of unique NICs installed in the PC, and the number of unique IPs assigned to the NICs. To force the operating system to create the bind order selection, select Default as the network adapter. This allows the driver to use the operating system's normal bind order to set the NIC.

Important: When unsure of which NIC to use, select the default condition. Furthermore, when an Ethernet-based device is being used and this feature is exposed through a product upgrade, select the default condition.

Note: With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties. Keep in mind that changes made to this property can temporarily disrupt communications.

Channel Properties — Write Optimizations

As with any server, writing data to the device may be the application's most important aspect. The server intends to ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties that can be used to meet specific needs or improve application responsiveness.



Write Optimizations

Optimization Method: Controls how write data is passed to the underlying communications driver. The options are:

- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags**: Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are needed 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. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.

  Note: This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

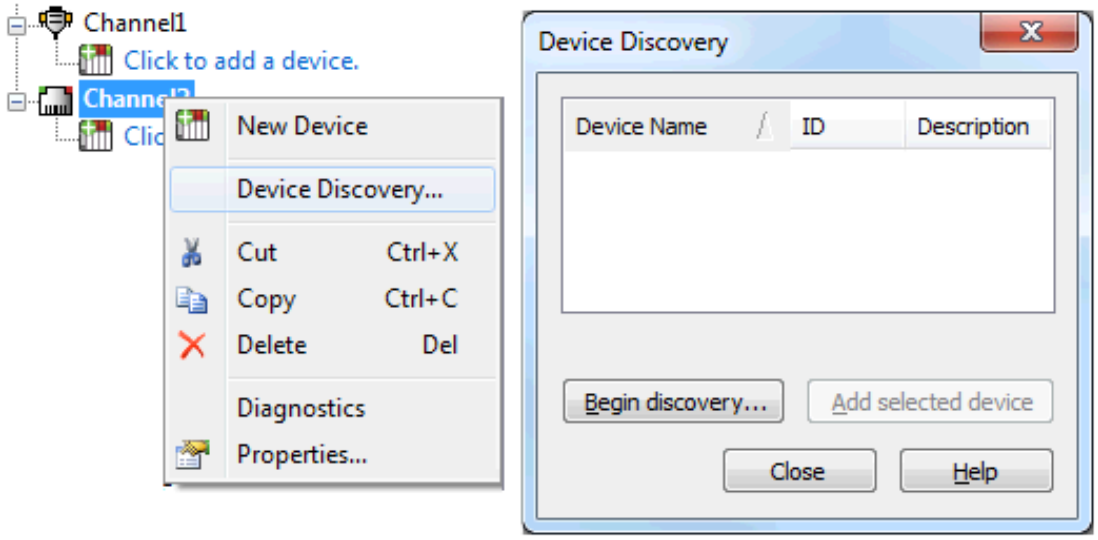
Duty Cycle: is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read

operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

Note: It is recommended that the application be characterized for compatibility with the write optimization enhancements before being used in a production environment.

Device Discovery Procedure

Device Discovery is available for drivers that support locating devices on the network. Once devices are found, they may be added to a channel. The maximum number of devices that can be discovered at once is 65535.



1. Select the channel in which devices should be discovered and added.
2. Right click on the channel node and select **Device Discovery...**
3. Click the **Begin discovery...** button to start the discovery process.
4. Specify the discovery properties, which are driver-specific, such as address range, timeout, discovery scope.
5. Click **OK**.
6. Devices discovered populate the dialog with the following information / headings **Name**, **ID**, **Description**.
7. If any discovered device is of interest, select that device and click **Add selected device...**.
8. Click **Close**.

What is a Device?

Devices represent the PLCs or other hardware with which the server communicates. The device driver that the channel is using restricts device selection.

Adding a Device

Devices can be added using the New Device Wizard both at the initial setup and afterward. To do so, click **Edit | New Device**. Users are prompted to enter the device name, which is user-defined and should be

logical for the device. This is the browser branch name used in OPC links to access the device's assigned tags. *For information on reserved characters, refer to* ***How To... Properly Name a Channel, Device, Tag, and Tag Group****.*

Users will also be prompted to enter a Network ID, which is a number or string that uniquely identifies the device on the device's network. Networked, multi-dropped devices must have a unique identifier so that the server's data requests are routed correctly. Devices that are not multi-dropped do not need an ID; this setting is not available.

Removing a Device

To remove a device from the project, select the desired device press **Delete**. Alternatively, click **Edit | Delete**.

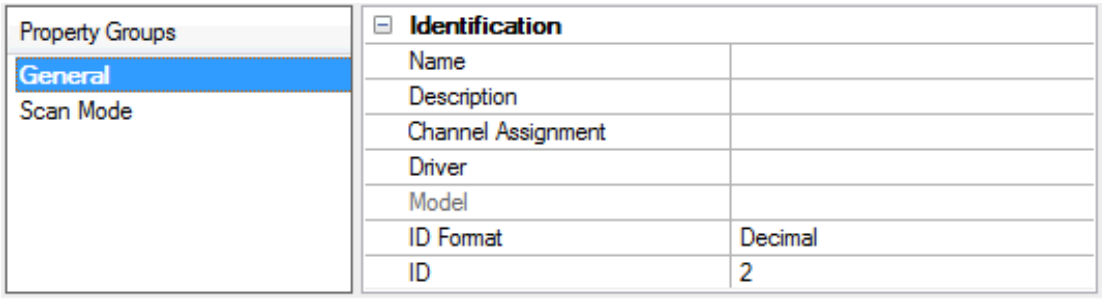
Displaying Device Properties

To display a device's properties, first select the device and click **Edit | Properties**.

For more information, refer to ***Device Properties****.*

Device Properties — General

A device represents a single target on a communications channel. If the driver supports multiple controllers, users must enter a device ID for each controller.



Identification

Name: This property specifies the name of the device. It is a logical user-defined name that can be up to 256 characters long, and may be used on multiple channels.

Note: Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".

For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.

Description: User-defined information about this device.

Many of these properties, including Description, have an associated system tag.

Channel Assignment: User-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

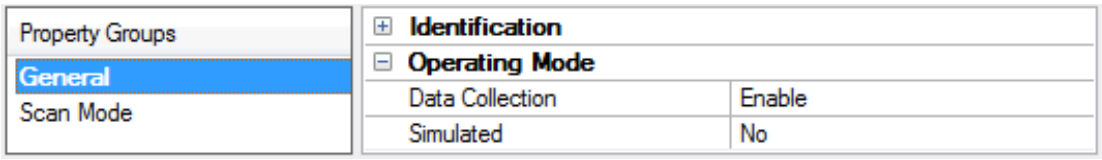
Model: This property specifies the specific type of device that is associated with this ID. The contents of the drop-down menu depends on the type of communications driver being used. Models that are not supported by a driver are disabled. If the communications driver supports multiple device models, the model selection can only be changed when there are no client applications connected to the device.

Note: If the communication driver supports multiple models, users should try to match the model selection to the physical device. If the device is not represented in the drop-down menu, select a model that conforms closest to the target device. Some drivers support a model selection called "Open," which allows users to communicate without knowing the specific details of the target device. For more information, refer to the driver help documentation.

ID: This property specifies the device's driver-specific station or node. The type of ID entered depends on the communications driver being used. For many communication drivers, the ID is a numeric value. Drivers that support a Numeric ID provide users with the option to enter a numeric value whose format can be changed to suit the needs of the application or the characteristics of the selected communications driver. The format is set by the driver by default. Options include Decimal, Octal, and Hexadecimal.

Note: If the driver is Ethernet-based or supports an unconventional station or node name, the device's TCP/IP address may be used as the device ID. TCP/IP addresses consist of four values that are separated by periods, with each value in the range of 0 to 255. Some device IDs are string based. There may be additional properties to configure within the ID field, depending on the driver. *For more information, refer to the driver's help documentation.*

Operating Mode



Data Collection: This property controls the device's active state. Although device communications are enabled by default, this property can be used to disable a physical device. Communications are not attempted when a device is disabled. From a client standpoint, the data is marked as invalid and write operations are not accepted. This property can be changed at any time through this property or the device system tags.

Simulated: This option places the device into Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client as valid data. While in Simulation Mode, the server treats all device data as reflective: whatever is written to the simulated device is read back and each OPC item is treated individually. The item's memory map is based on the group Update Rate. The data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

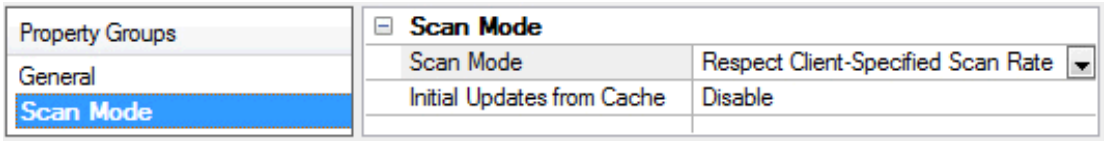
Notes:

1. This System tag (_Simulated) is read only and cannot be written to for runtime protection. The System tag allows this property to be monitored from the client.
2. In Simulation mode, the item's memory map is based on client update rate(s) (Group Update Rate for OPC clients or Scan Rate for native and DDE interfaces). This means that two clients that reference the same item with different update rates return different data.

Simulation Mode is for test and simulation purposes only. It should never be used in a production environment.

Device Properties — Scan Mode

The Scan Mode specifies the subscribed-client requested scan rate for tags that require device communications. Synchronous and asynchronous device reads and writes are processed as soon as possible; unaffected by the Scan Mode properties.



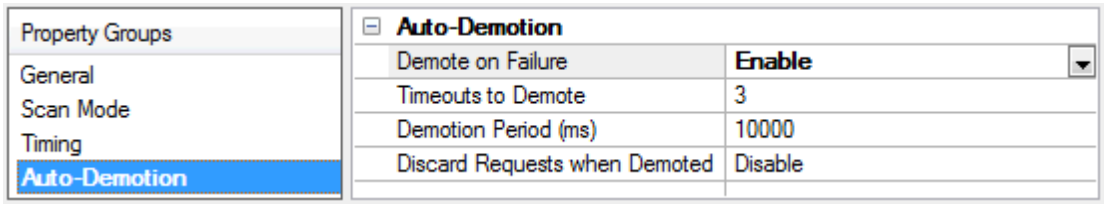
Scan Mode: Specifies how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

- **Respect Client-Specified Scan Rate**: This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate**: This mode specifies the value set as the maximum scan rate. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
  Note: When the server has an active client and items for the device and the scan rate value is increased, the changes take effect immediately. When the scan rate value is decreased, the changes do not take effect until all client applications have been disconnected.
- **Request All Data at Scan Rate**: This mode forces tags to be scanned at the specified rate for subscribed clients. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
- **Do Not Scan, Demand Poll Only**: This mode does not periodically poll tags that belong to the device nor perform a read to get an item's initial value once it becomes active. It is the client's responsibility to poll for updates, either by writing to the _DemandPoll tag or by issuing explicit device reads for individual items. *For more information, refer to "Device Demand Poll" in server help*.
- **Respect Tag-Specified Scan Rate**: This mode forces static tags to be scanned at the rate specified in their static configuration tag properties. Dynamic tags are scanned at the client-specified scan rate.

Initial Updates from Cache: When enabled, this option allows the server to provide the first updates for newly activated tag references from stored (cached) data. Cache updates can only be provided when the new item reference shares the same address, scan rate, data type, client access, and scaling properties. A device read is used for the initial update for the first client reference only. The default is disabled; any time a client activates a tag reference the server attempts to read the initial value from the device.

Device Properties — Auto-Demotion

The Auto-Demotion properties can temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device offline for a specific time period, the driver can continue to optimize its communications with other devices on the same channel. After the time period has been reached, the driver re-attempts to communicate with the non-responsive device. If the device is responsive, the device is placed on-scan; otherwise, it restarts its off-scan time period.



Demote on Failure: When enabled, the device is automatically taken off-scan until it is responding again.

Tip: Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

Timeouts to Demote: Specify how many successive cycles of request timeouts and retries occur before the device is placed off-scan. The valid range is 1 to 30 successive failures. The default is 3.

Demotion Period: Indicate how long the device should be placed off-scan when the timeouts value is reached. During this period, no read requests are sent to the device and all data associated with the read requests are set to bad quality. When this period expires, the driver places the device on-scan and allows for another attempt at communications. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds.

Discard Requests when Demoted: Select whether or not write requests should be attempted during the off-scan period. Disable to always send write requests regardless of the demotion period. Enable to discard writes; the server automatically fails any write request received from a client and does not post a message to the Event Log.

Device Properties — Communication Parameters

Ethernet Encapsulation mode has been designed to provide communication with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port. The terminal server converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted to a serial form, users can connect standard devices that support serial communications to the terminal server.

For more information, refer to "How to... Use Ethernet Encapsulation" in the server help.

Note: Because Ethernet Encapsulation mode is completely transparent to the actual serial communications driver, users should configure the remaining device properties as if they were connecting to the device directly on the local PC serial port.

IP Address: This property is used to enter the four-field IP address of the terminal server to which the 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; however, devices may have the same IP address if there are multiple devices multi-dropped from a single terminal server.

Port: This property is used to configure the Ethernet port to be used when connecting to a remote terminal server.

Protocol: This property is used to select either TCP/IP or UDP communications. The selection depends on the nature of the terminal server being used. The default protocol selection is TCP/IP. For more information on available protocols, refer to the terminal server's help documentation.

Notes:

1. With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties.
2. The valid IP Address range is greater than (>) 0.0.0.0 to less than (<) 255.255.255.255.

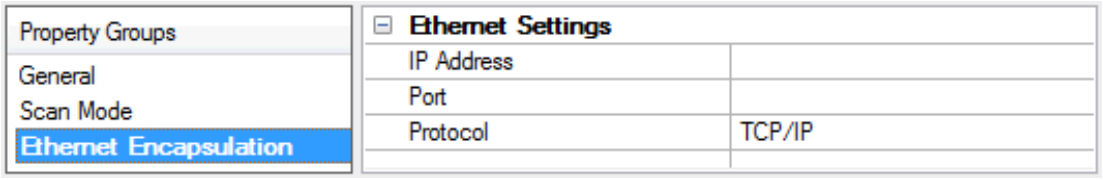
Device Properties — Ethernet Encapsulation

Ethernet Encapsulation is designed to provide communication with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port. The terminal server con-

verts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted to a serial form, users can connect standard devices that support serial communications to the terminal server.

For more information, refer to "How to... Use Ethernet Encapsulation" in server help.

Ethernet Encapsulation is transparent to the driver; configure the remaining properties as if connecting to the device directly on a local serial port.



IP Address: This property is used to enter the four-field IP address of the terminal server to which the 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; however, devices may have the same IP address if there are multiple devices multi-dropped from a single terminal server.

Port: This property is used to configure the Ethernet port to be used when connecting to a remote terminal server.

Protocol: This property is used to select either TCP/IP or UDP communications. The selection depends on the nature of the terminal server being used. The default protocol selection is TCP/IP. For more information on available protocols, refer to the terminal server's help documentation.

Notes

1. With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties.
2. The valid IP Address range is greater than (>) 0.0.0.0 to less than (<) 255.255.255.255.

Device Properties — Tag Generation

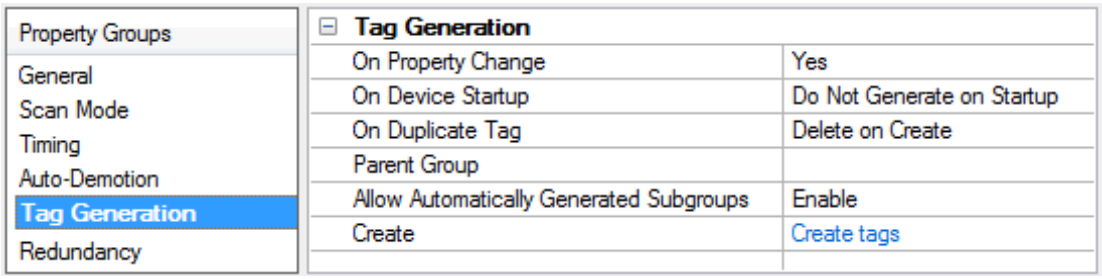
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the supporting driver) can be browsed from the clients.

Not all devices and drivers support full automatic tag database generation and not all support the same data types. Consult the data types descriptions or the supported data type lists for each driver for specifics.

If the target device supports its own local tag database, the driver reads the device's tag information and uses the data to generate tags within the server. If the device does not natively support named tags, the driver creates a list of tags based on driver-specific information. An example of these two conditions is as follows:

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

Note: Automatic tag database generation's mode of operation is completely configurable. *For more information, refer to the property descriptions below.*



On Property Change: If the device supports automatic tag generation when certain properties change, the **On Property Change** option is shown. It is set to **Yes** by default, but it can be set to **No** to control over when tag generation is performed. In this case, the **Create tags** action must be manually invoked to perform tag generation.

On Device Startup: This property specifies when OPC tags are automatically generated. Descriptions of the options are as follows:

- **Do Not Generate on Startup**: This option prevents the driver from adding any OPC tags to the tag space of the server. This is the default setting.
- **Always Generate on Startup**: This option causes the driver to evaluate the device for tag information. It also adds tags to the tag space of the server every time the server is launched.
- **Generate on First Startup**: This option causes the driver to evaluate the target device for tag information the first time the project is run. It also adds any OPC tags to the server tag space as needed.

Note: When the option to automatically generate OPC tags is selected, any tags that are added to the server's tag space must be saved with the project. Users can configure the project to automatically save from the **Tools** | **Options** menu.

On Duplicate Tag: When automatic tag database generation is enabled, the server needs to know what to do with the tags that it may have previously added or with tags that have been added or modified after the communications driver since their original creation. This setting controls how the server handles OPC tags that were automatically generated and currently exist in the project. It also prevents automatically generated tags from accumulating in the server.

For example, if a user changes the I/O modules in the rack with the server configured to **Always Generate 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. The options are:

- **Delete on Create**: This option deletes any tags that were previously added to the tag space before any new tags are added. This is the default setting.
- **Overwrite as Necessary**: This option instructs the server to only remove the tags that the communications driver is replacing with new tags. Any tags that are not being overwritten remain in the server's tag space.
- **Do not Overwrite**: This option prevents the server from removing any tags that were previously generated or already existed in the server. The communications driver can only add tags that are completely new.

- **Do not Overwrite, Log Error**: This option has the same effect as the prior option, and also posts an error message to the server's Event Log when a tag overwrite would have occurred.

Note: Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags. Users should avoid adding tags to the server using names that may match tags that are automatically generated by the driver.

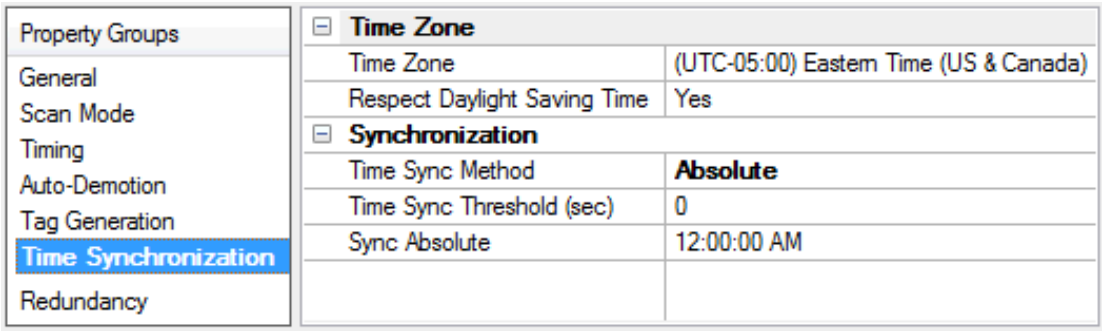
Parent Group: This property keeps automatically generated tags from mixing with tags that have been entered manually by specifying a group to be used for automatically generated tags. The name of the group can be up to 256 characters. This parent group provides a root branch to which all automatically generated tags are added.

Allow Automatically Generated Subgroups: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting tags are named with the address value. For example, the tag names are not retained during the generation process.
Note: If, as the server is generating tags, a tag is assigned the same name as an existing tag, the system automatically increments to the next highest number so that the tag name is not duplicated. For example, if the generation process creates a tag named "AI22" that already exists, it creates the tag as "AI23" instead.

Create: Initiates the creation of automatically generated OPC tags. If the device's configuration has been modified, **Create tags** forces the driver to reevaluate the device for possible tag changes. Its ability to be accessed from the System tags allows a client application to initiate tag database creation.
Note: **Create tags** is disabled if the Configuration edits a project offline.

Device Properties — Time Synchronization

This group is used to specify the device's time zone and time synchronization properties. It primarily applies to time stamped data or information from battery-powered devices at remote locations where the device time may deviate (causing issues with the time-stamped data). To prevent this problem from occurring, users can specify that the server synchronize the device time.



Note: Not all drivers and models support all options.

Time Zone: This property specifies the device's time zone. To ignore the time zone, select one of the first four options in the list (which do not have an offset). The default is the time zone of the local system.
Note: The driver uses this property both when synching the device time and when converting EFM timestamps from the device to UTC time.

Respect Daylight Saving Time: Select Yes to follow Daylight Saving Time offset when synching the device time. Select No to ignore Daylight Saving Time. Only time zones that observe Daylight Saving Time will be affected. The default is No (disabled).

Note: When enabled, the time of the device is adjusted by +1 hour for Daylight Saving Time (in the spring), and adjusted by -1 hour after Daylight Saving Time (in the fall).

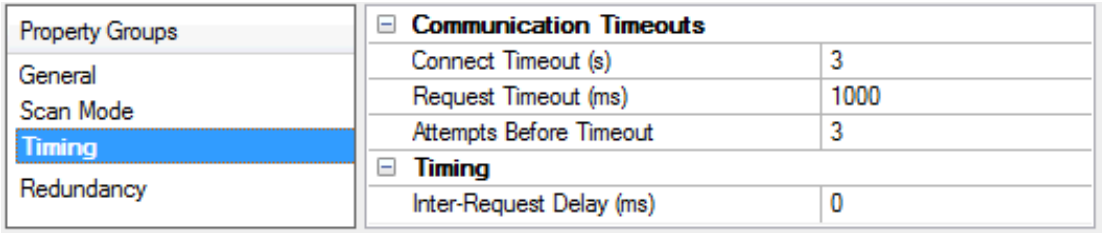
Time Sync Method: This property specifies the method of synchronization. Options include Disabled, Absolute, and Interval. The default is Disabled. Descriptions of the options are as follows:

- **Disabled**: No synchronization.
- **Absolute**: Synchronizes to an absolute time of day specified through the Time property (appears only when Absolute is selected).
- **Interval**: Synchronizes on startup and every number of minutes specified through the Sync Interval property (appears only when Interval is selected). The default is 60 minutes.
- **OnPoll**: Synchronizes when poll is completed (applicable only to EFM devices).

Time Sync Threshold: This property specifies the maximum allowable difference, in seconds, between the device time and the system time before syncing the device time to the system time. If the threshold is set to 0, a time synchronization occurs every time. The default is 0 seconds. The maximum allowable threshold is 600 seconds.

Device Properties — Timing

The device Timing properties allow the driver's response to error conditions to be tailored to fit the application's needs. In many cases, the environment requires changes to these properties for optimum performance. Factors such as electrically generated noise, modem delays, and poor physical connections can influence how many errors or timeouts a communications driver encounters. Timing properties are specific to each configured device.



Communications Timeouts

Connect Timeout: This property (which is used primarily by Ethernet based drivers) controls the amount of time required to establish a socket connection to a remote device. The device's connection time often takes longer than normal communications requests to that same device. The valid range is 1 to 30 seconds. The default is typically 3 seconds, but can vary depending on the driver's specific nature. If this setting is not supported by the driver, it is disabled.

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

Request Timeout: This property specifies an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9,999,999 milliseconds (167.6667 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

Attempts Before Timeout: This property specifies how many times the driver issues a communications request before considering the request to have failed and the device to be in error. The valid range is 1 to 10. The default is typically 3, but can vary depending on the driver's specific nature. The number of attempts

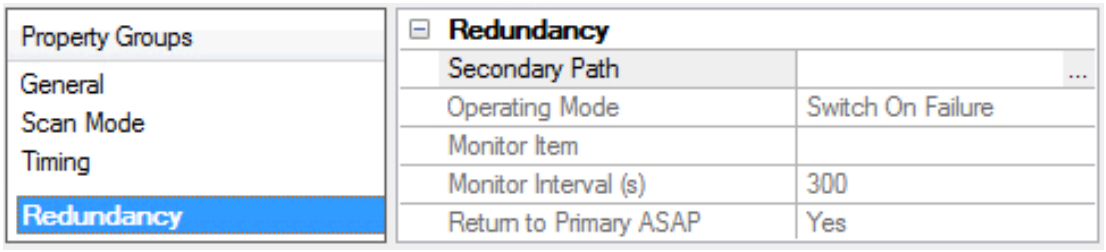
configured for an application depends largely on the communications environment. This property applies to both connection attempts and request attempts.

Timing

Inter-Request Delay: This property specifies how long the driver waits before sending the next request to the target device. It overrides the normal polling frequency of tags associated with the device, as well as one-time 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. Configuring a delay for a device affects communications with all other devices on the channel. It is recommended that users separate any device that requires an inter-request delay to a separate channel if possible. Other communications properties (such as communication serialization) can extend this delay. The valid range is 0 to 300,000 milliseconds; however, some drivers may limit the maximum value due to a function of their particular design. The default is 0, which indicates no delay between requests with the target device.

Note: Not all drivers support Inter-Request Delay. This setting does not appear if it is not available.

Device Properties — Redundancy



Redundancy is available with the Media-Level Redundancy Plug-In.

Consult the website, a sales representative, or the user manual for more information.

What is a Tag?

A tag represents addresses within the PLC or other hardware device with which the server communicates. The server allows both Dynamic tags and user-defined Static tags. Dynamic tags are entered directly in the OPC client and specify device data. User-defined Static tags are created in the server and support tag scaling. They can be browsed from OPC clients that support tag browsing.

Displaying Tag Properties

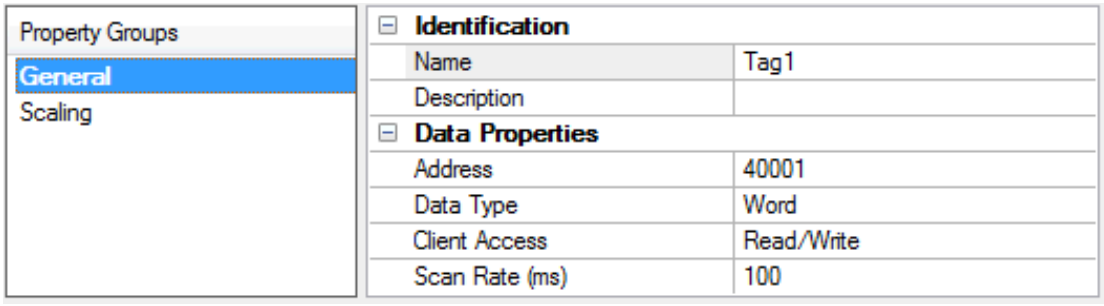
To invoke the tag properties for a specific tag, double-click on it in the Tag Selection pane of the server configuration.

Tag Name	Address	Data Type	Scan Rate	Scaling	Description
Tag1	40001	Word	100	None	
Tag2	40002	Word	100	None	
Tag3	40003	Word	100	None	
Tag4	40004	Float	100	None	
Tag5	40005	Word	100	None	
Tag6	40006	Word	100	Square Root	
Tag7	40007	Word	100	None	
Tag8	40008	Word	100	None	
Tag9	40009	Word	100	None	
Tag10	40010	Word	50	None	
Tag11	40011	Word	100	None	
Tag12	40012	Word	100	None	
Tag13	40013	Word	100	None	
Tag14	40014	Word	100	Linear	
Tag15	40015	Word	100	None	
Tag16	40016	Word	100	None	
Tag17	40017	Word	100	None	
Tag18	40018	LBCD	100	None	
Tag19	40019	Word	100	None	
Tag20	40020	Word	100	None	
Tag21	40021	Word	100	None	
Tag22	40022	Word	100	None	

Tag Properties — General

A tag represents addresses in the PLC or other hardware device with which the server communicates. The server allows both Dynamic tags and user-defined Static tags. Dynamic tags are entered directly in the OPC client and specify device data. User-defined Static tags are created in the server and support tag scaling. They can be browsed from OPC clients that support tag browsing.

For more information, refer to ***Dynamic Tags*** *and* ***Static User-Defined Tags****.*



Name: Enter a string to represent this tag. The tag name can be up to 256 characters in length. While using descriptive names is generally a good idea, 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. *For information on reserved characters, refer to* ***How To... Properly Name a Channel, Device, Tag, and Tag Group****.*

Tip: If the application is best suited for using blocks of tags with the same names, use tag groups to segregate the tags. *For more information, refer to* ***Tag Group Properties****.*

Description: Apply a comment to the tag. A string of up to 255 characters can be entered for the description. When using an OPC client that supports Data Access 2.0 tag properties, the description property is accessible from the tag's item Description properties.

Address: Enter the target tag's driver address. The address's format is based on the driver protocol. The address can be up to 128 characters.
Tip: For hints about how an address should be entered, click the browse (...) button. If the driver accepts the address as entered, no messages are displayed. A popup informs of any errors. Some errors are related to the data type selection and not the address string.

Data Type: 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 and the driver knows what format needs to be used when reading the device's data. 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 one, two, four, eight, or sixteen registers or possibly a Boolean value. The driver governs the data format being chosen.

- **Default** - Uses the driver default data type
- **Boolean** - Binary value of true or false
- **Char** - Signed 8-bit integer data
- **Byte** - Unsigned 8-bit integer data
- **Short** - Signed 16-bit integer data
- **Word** - Unsigned 16-bit integer data
- **Long** - Signed 32-bit integer data
- **DWord** - Unsigned 32-bit integer data
- **LLong** - Signed 64-bit integer data
- **QWord** - Unsigned 64-bit integer data
- **Float** - 32-bit real value IEEE-754 standard definition
- **Double** - 64-bit real value IEEE-754 standard definition
- **String** - Null-terminated Unicode string
- **BCD** - Two byte-packed BCD value range is 0-9999
- **LBCD** - Four byte-packed BCD value range is 0-99999999
- **Date -** See **Microsoft® Knowledge Base**.

Client Access: Specify whether the 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 affects how the tag appears in the browse space of an OPC client. Many OPC client applications allow filtering tags based on attributes. Changing the access method of this tag may change how and when the tag appears in the browse space of the OPC client.

Scan Rate: 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, users can tailor the bandwidth requirements of the server to suit the needs of

the application. If, for example, data that changes very slowly needs to be read, 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 milliseconds (ms), with a 10 ms increment. The default is 100 milliseconds.

- With the server's online full-time operation, these properties can be changed at any time. Changes made to tag properties take effect immediately; however, OPC clients that have already connected to this tag are not affected until they release and attempt to reacquire it. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties.

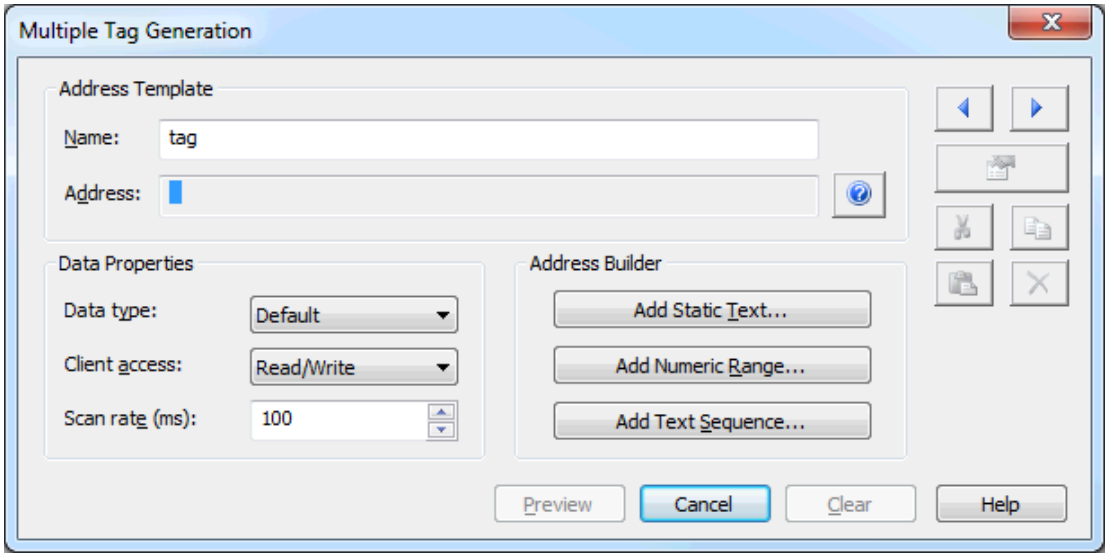
Multiple Tag Generation

The Multiple Tag Generation Tool dynamically creates multiple tags using user-defined driver nomenclature. It allows a variety of address formats (such as ranges utilizing decimal, hexadecimal, and octal number systems). To avoid overlapping data, the Tag Generator Tool also has the ability to increment by the user-defined data type.

For information on a specific dialog, select a link from the list below:

Add Numeric Range
Add Static Text
Add Text Sequence
Multiple Tag Generation Preview
Tag Name Properties

Multiple Tag Generation



Address Template

Name: Enter user-defined tag name.

Address: Verify the tag address, generated through options defined in the Address Builder section.

Data Properties

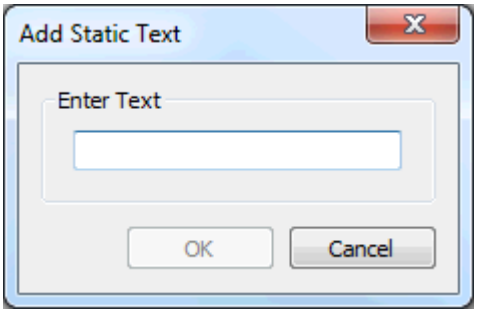
Data Type: Select data type to apply to all generated tags. Depending on the native interface supported by the driver, the data type may override the default increment of the Add Numeric Range property for the last element. The default setting is Default.

Client Access: Select the tag's permission settings from Read Only or Read / Write. The default setting is Read Only.

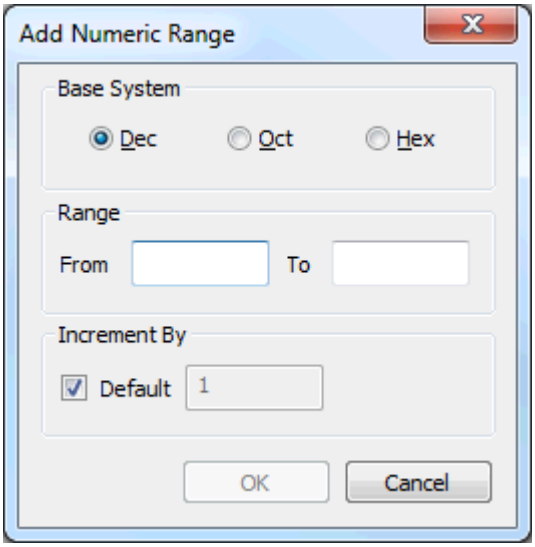
Scan Rate: Specify the frequency at which tags are scanned. The valid range is 10 to 99999990 milliseconds. The default setting is 100 milliseconds.

Address Builder

Add Static Text...: Click to launch the Add Static Text dialog where a single line of text can be entered.

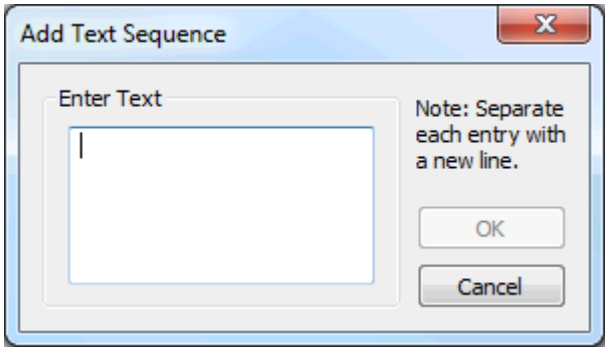


Add Numeric Range...: Click to launch the Add Numeric Range dialog.



- **Base System** Select the format of the base system: Decimal, Octal, or Hexadecimal. The default setting is Decimal.
- **Range** Enter the starting and ending values for the numeric range in the From and To fields.
- **Increment By** When not using Default (which increments by one), users can specify a custom increment value. The range increments according to the selected Base System.

Add Text Sequence...: Click to launch the Add Text Sequence dialog where multiple strings can be created. Each string is inserted independently of the other strings specified in the list.

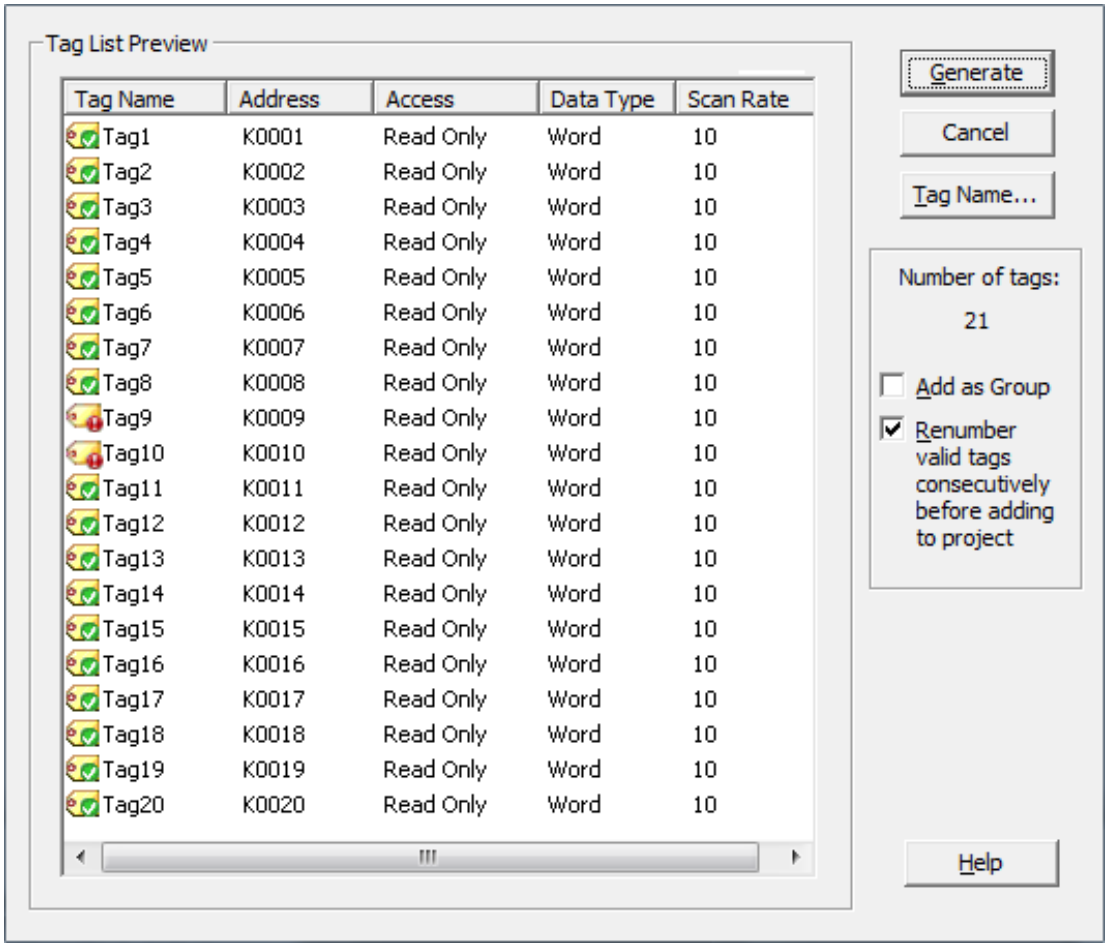


Tips

1. To enable the Edit icons to the right, highlight a section of the tag address syntax element.
2. The Hints icon opens the help file on Address Descriptions.

Preview: Click to generate a test view of the generated tags.

Multiple Tag Generation Preview



Generate: Click to send all valid tags to the server for insertion.

Cancel: Click to reject any changes made to the tags and return to the prior dialog.

Tag Name...: Click to invoke the Tag Name Properties dialog.

Add as Group: Enable to add the tags into a single organizing group. The default setting is disabled.

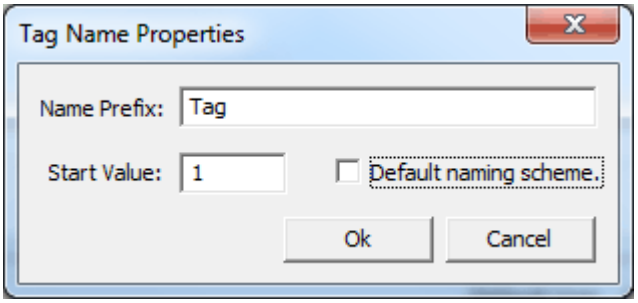
Renumber valid tags consecutively before adding to project: Enable to renumber the tags consecutively before adding to the project. The default setting is enabled.

Note: Tags shown with a green checkmark are valid. Tags shown with a red exclamation mark (!) are invalid.

Tag Name Properties

The Tag Generator Tool includes the option for a custom naming scheme, allowing users to specify both a name prefix and a numeric suffix to all the tags. The numeric suffix is automatically incremented for each tag, allowing users to create custom names for tags for better readability. Assigned tag names may be changed after generation. A default naming scheme is implemented to each generated tag if the user does not define a custom name through the Tag Name Properties dialog.

Note: Users who change the naming scheme in the Generation dialog before returning to the Tag Duplication dialog to make changes to the addressing syntax can choose to save the naming scheme for the next time the tag list is generated.



Name Prefix: Enter a custom name prefix (letters to pre-pend to the tag name).

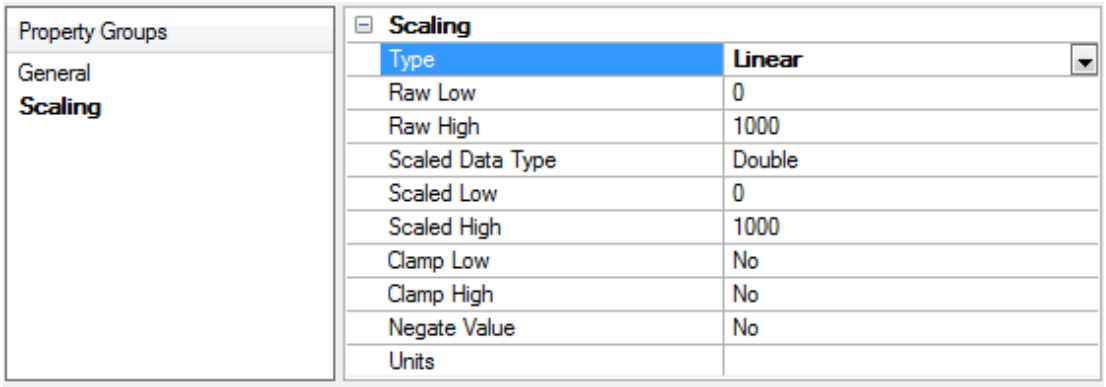
Start Value: Specify the numeric first value to increment for each tag.

Default naming scheme: When enabled, the default naming scheme is used. The default setting is disabled.

See Also: *Generating Multiple Tags*

Tag Properties — Scaling

This server supports tag Scaling, which allows raw data from the device to be scaled to an appropriate range for the application.



Type: Select the method of scaling raw values. Select **Linear**, **Square Root**, or **None** to disable. The formulas for scaling types are shown below.

Type	Formula for Scaled Value
Linear	(((ScaledHigh - ScaledLow)/(RawHigh - RawLow))*(RawValue - RawLow)) + ScaledLow
Square root	(Square root ((RawValue - RawLow)/(RawHigh - RawLow))*(ScaledHigh - ScaledLow)) + ScaledLow

Raw Low: Specify the lower end of the range of data from the device. The valid range depends on the raw tag data type. For example, if the raw value is Short, the valid range of the raw value would be from -32768 to 32767.

Raw High: Specify the upper end of the range of data from the device. The Raw High value must be greater than the Raw Low value. The valid range depends on the raw tag data type.

Scaled Data Type: Select the data type for the tag being scaled. The data type can be set to any valid OPC data type, including a raw data type, such as Short, to an engineering value with a data type of Long. The default scaled data type is Double.

Scaled Low: Specify the lower end of the range of valid resulting scaled data values. The valid range depends on the tag data type.

Scaled High: Specify the upper end of the range of valid resulting scaled data values. The valid range depends on the tag data type.

Clamp Low: Select **Yes** to prevent resulting data from exceeding the lower end of the range specified. Select **No** to allow data to fall outside of the established range.

Clamp High: Select **Yes** to prevent resulting data from exceeding the upper end of the range specified. Select **No** to allow data to fall outside of the established range.

Negate Value: Select **Yes** to force the resulting value to be negated before being passed to the client. Select **No** to pass the value to the client unmodified.

The server supports the OPC tag properties available in the 2.0 Data Access specifications. If the OPC client being used supports these properties, it can automatically configure the range of objects (such as user input objects or displays) by using the Scaling settings. Utilize the User Manager to restrict access rights to server features to prevent any unauthorized operator from changing these properties.

Dynamic Tags

Dynamic tag addressing is a second method of defining tags that allows users to define tags only in the client application. As such, instead of creating a tag item in the client that addresses another tag item created in the server, users only need to create tag items in the client that directly accesses the device driver's addresses. On client connect, the server creates a virtual tag for that location and starts scanning for data automatically.

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

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

If the data type is omitted, the driver chooses a default data type based on the device and address being referenced. The default data types for all locations are documented in each individual driver's help documentation. If the data type specified is not valid for the device location, the server rejects the tag and an error posts in the Event Log.

OPC Client Using Dynamic Addressing Example

Scan the 16-bit location "R0001" on the Simulator device. The following Dynamic tag examples assume that the project created is part of the example.

1. Start the OPC client application and connect to the server.
2. Using the Simulator Driver, create a channel and name it "Channel1." Then, make a device and name it "Device1."
3. In the client application, define an item name as "Channel1.Device1.R0001@Short."
4. The client project automatically starts 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.

Note: When utilizing 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 is used if it is supported by the communications driver. The @[Data Type] modifier can be useful when ensuring that a communications driver interprets a piece of data exactly as needed.

Non-OPC Client Example

Non-OPC clients can override the update rate on a per-tag basis by appending @[Update Rate].

For example, appending:
<DDE service name>|_ddedata!Device1.R0001@500 overrides just the update rate.
<DDE service name>|_ddedata!Device1.R0001@500,Short overrides both update rate and data type.

Tips:

1. The server creates a special Boolean tag for every device in a project that can be used by a client to determine whether a device is functioning properly. To use this tag, specify the item in the link as "Error." If the device is communicating properly, the tag's value is zero; otherwise, it is one.
2. If the 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 references the address pointed to by the user-defined tag.
3. Static tags must be used to scale data in the server.

See Also:
Static Tags (User-Defined)
Designing a Project: Adding User-Defined Tags

Static Tags (User-Defined)

The most common method that uses the server to get data from the device to the client application has two requirements. Users must first define a set of tags in the server using the assigned tag name as the item of each link between the client and the server. The primary benefit to using this method is that all user-defined tags are available for browsing within most OPC clients. Before deciding whether or not to create Static tags, ensure that the client can browse or import tags from the server.

Tip: User-defined tags support scaling.

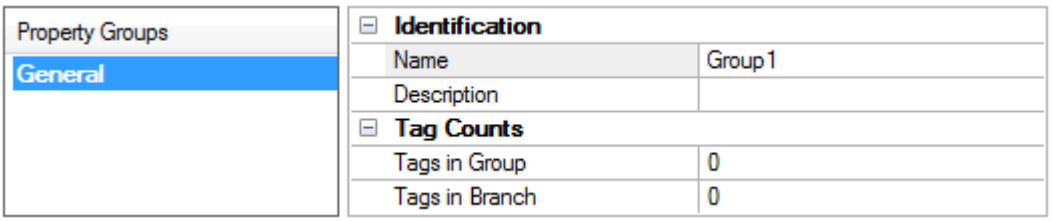
What is a Tag Group?

This server allows tag groups to be added to the project. Tag groups are used to tailor the layout of OPC data into logical groupings that fit the application's needs. Tag groups allow multiple sets of identical tags to be added under the same device: this can be convenient when a single device handles a number of similar machine segments.

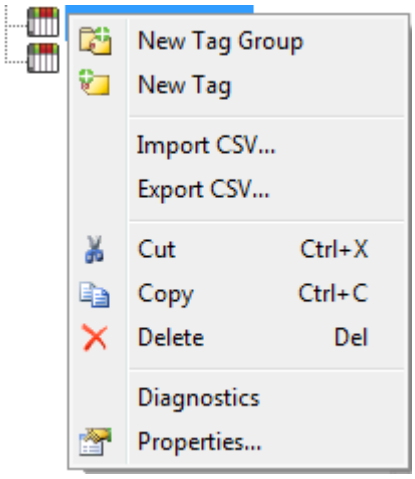
Tag Group Properties

From an OPC client standpoint, tag groups allow users to segregate OPC data into smaller tag lists, making finding specific tags easier.

The following image used the supplied OPC Quick Client to create Cell1 and Cell2 tag groups and simplify the OPC client browsing.



To add a new tag group to the project, right-click on either an existing device or tag group branch and select **New Tag Group** from the context menu. Alternatively, click on either an existing device or tag group branch and click the New Tag Group icon on the toolbar.



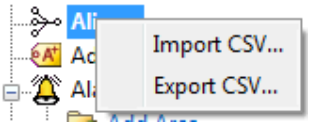
Tag groups can be added at any level from the device-level down, and multiple tag groups can be nested together to fit the application's needs. As seen in the OPC Quick Client dialog above, the fully qualified OPC item path is "Channel1.Device1.Machine1.Cell1.Tag1". For this OPC item, "Machine1" and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

Note: With the server's online full-time operation, these properties can be changed at any time. Any changes made to the tag groups take effect immediately. If the name is changed, OPC clients that have already used that tag group as part of an OPC item request are not affected until they release the item and attempt to reacquire it. New tag groups added to the project immediately allows browsing from an OPC client. Utilize the User Manager to restrict access rights to server features to prevent operators from changing the properties.

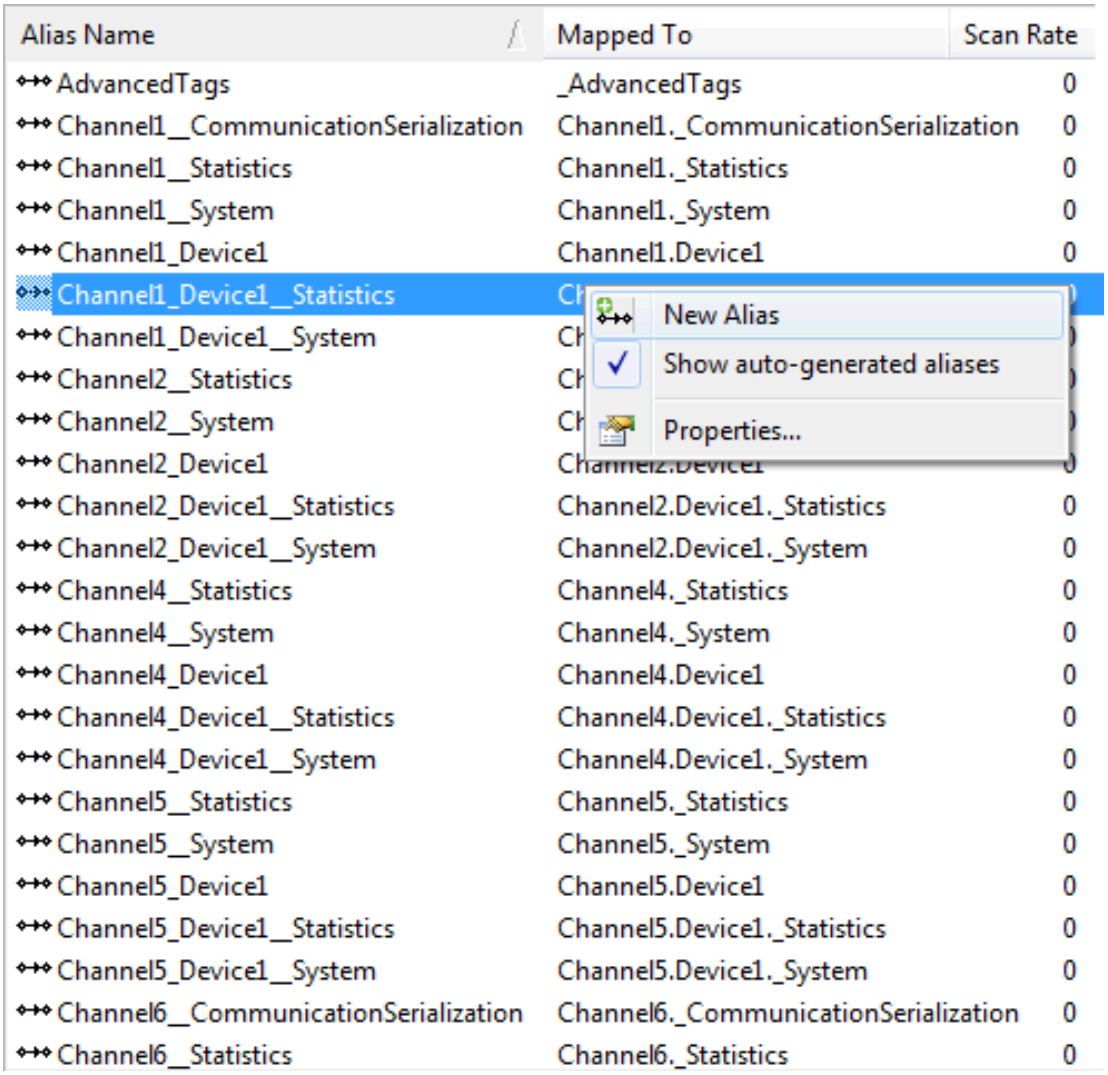
What is the Alias Map?

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. Although the latest version of the server automatically creates the alias map, users can add their own alias map entries to compliment those created by the server. Users can also filter the server created aliases so that the only ones visible are their own.

Alias map elements can be exported and imported by right-clicking on the target alias in the tree view pane.



Alias map elements can be added, edited, and deleted by right-clicking on the target alias in the detail pane.



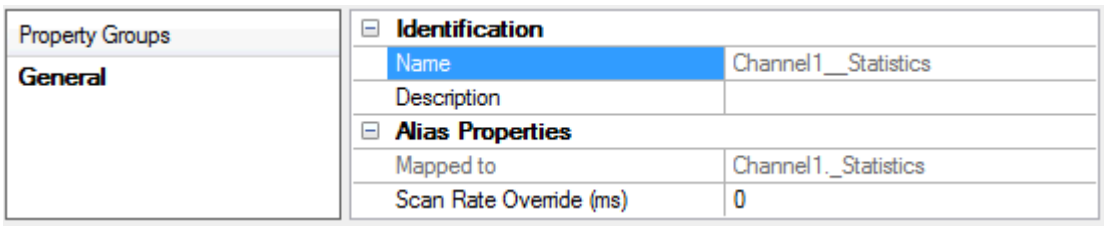
Note: When enabled, the **Show auto-generated aliases** displays those alias maps created by the server automatically.

See Also: *How to... Create and Use an Alias*

Alias Properties

The Alias Map allows a way to assign alias names to complex tag references that can be used in client applications.

An alias is constructed by entering an alias name and clicking on the desired device name or group name.



Name: Specify the alias name, which can be up to 256 characters long. It must be unique in the alias map. For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.

Description: Enter a description of this alias to clarify data sources and reports (optional).

Mapped to: Specify or browse to the location of the alias. Because the alias map does not allow tag items to be browsed from the alias table, create a short nickname that replaces the address that leads up to the tag. This makes it easier to address items in a client application that does not support tag browsing.

Scan Rate Override: Specify an update rate to be applied to all non-OPC tags accessed using this alias map entry. The valid range is 0 to 99999990 milliseconds. The default is 0 milliseconds.

Tip: This setting is equivalent to the topic update rate found in many DDE-only servers.

Note: When set to 0 milliseconds, the server observes the scan rate set at the individual tag level.

What is the Event Log?

The Event Log displays the date, time, and source of an error, warning, information, or security event. For more information, select a link from the list below.

Event Log Options
Event Log Settings

Event Log

Users can specify the type of events displayed in the Event Log. There are currently four types of events that can be recorded: Error Events, Warning Events, Information Events, and Security Events. Descriptions of the events are as follows:

Information: Messages that provide status and data requiring no interaction or correction, such as successful connection or data collection.

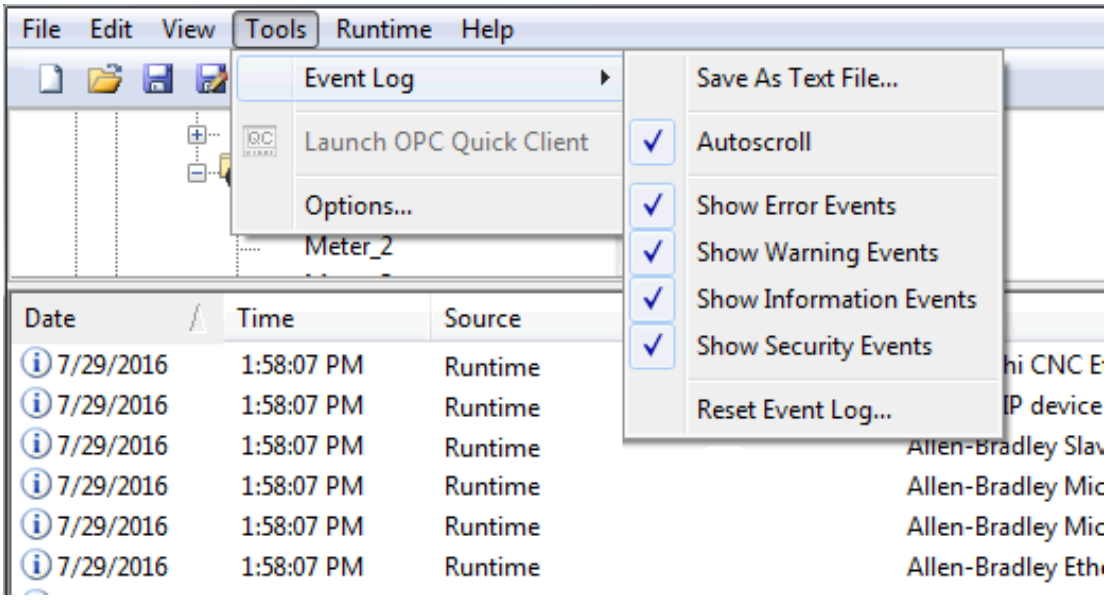
Security: Messages that call attention to conditions that are not best practices from a security perspective, such as running the software as the default user versus a logged-in user with valid credentials.

Warning: Messages that indicate an issue that does not require interaction, but may result in unexpected results, such as a device not responding.

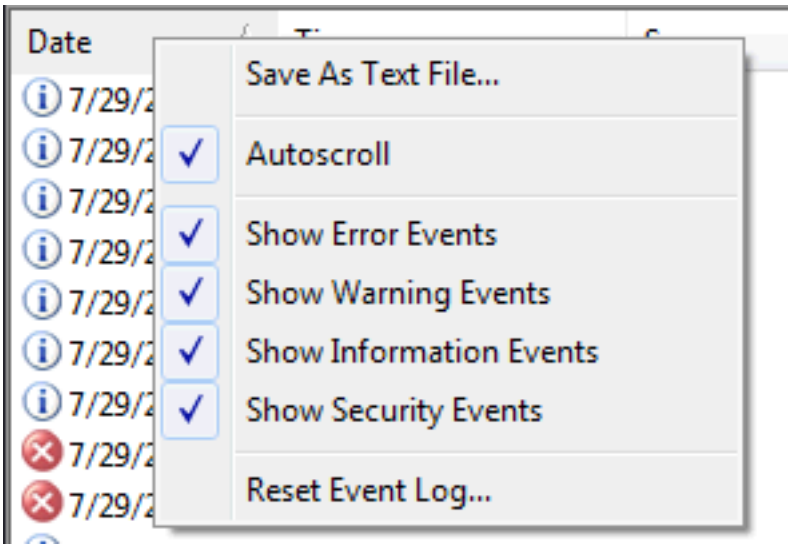
Error: Messages that alert the user to failures or problems that, generally, should be researched and corrected for best results.

Note: To access the event types in the Configuration client, click **Tools | Event Log**. Alternatively, right-click anywhere in the Event Log display.

Tools menu



Right Click



Note: The Event Log system would be useless if there was no mechanism to protect its contents. If operators could change these properties or reset the log, the purpose would be lost. Utilize the User Manager to limit the functions an operator can access and prevent these actions from occurring.

See Also: *Settings - Event Log*

Tag Management

The server's user-defined tag management features can create a tag database structure to fit each application's specific nature. Users can define multiple tag groups to segregate tag data on a device-by-device basis, and can also easily add large numbers of tags through drag and drop editing. CSV import and export also allow tag editing in any application. Like all other server features, new tags can be added to the application at any time.

Automatic Tag Database Generation

The OPC server's ability to automatically generate tags for select communication drivers brings OPC technology one step closer to Plug and Play operation. Tag information can be read directly from a device, and tags can also be generated from stored tag data. In either case, users no longer need to manually enter OPC tags into the server.

System Tags

System tags provide general error feedback to client applications, allow the operation control over when a device is actively collecting data, and also permit a channel or device's standard properties to be changed from an OPC client application. The number of System tags available at the channel or device level depends on the nature of the driver being used.

Note: System tags can be grouped according to their purpose as both status and control or property manipulation.

See Also: SAF System Tags

Property Tags

Property tags are additional tags that can be accessed by any Data Access client by appending the property name to any fully qualified tag address. When using an OPC client that supports item browsing, users can browse tag properties by turning on **Include tag properties when a client browses the server** under OPC DA settings. *For more information, refer to* ***Project Properties — OPC DA****.*

Statistics Tags

Statistics tags provide feedback to client applications regarding the operation of the channel communications in the server. When diagnostics are enabled, seven built-in Statistics tags are available. *For more information, refer to* ***OPC Diagnostic Viewer****.*

Modem Tags

Modem tags configure modem properties and monitor modem status. They are only available when the **Connection Type** in **Channel Properties** is set to **Modem**. *For more information, refer to* ***Channel Properties — Serial Communications****.*

Communication Serialization Tags

Driver communications normally occur simultaneously across multiple channels, yielding higher data throughput. In some applications, however, it is required that only one channel be allowed to communicate at a time. Communication Serialization provides this support. Communication Serialization tags are used to configure and monitor a channel's serialization status. Both the feature and its tags are only available to specific drivers. *For more information, refer to the driver's help documentation.*

CSV Import and Export

This server can import and export tag data in a Comma-Separated Variable (CSV) file to quickly create tags in an application. The CSV functions are only available when a device or tag group is selected.

Note: For information on which character to specify as the variable, refer to Options - General.

To jump to a specific section, select a link from the list below.

Exporting a Server Tag List
Importing a Server Tag List into the Server
Using Other Characters as the Delimiter

Creating a Template

The easiest way to create and import CSV file is to create a template. For more information, refer to the instructions below.

1. To start, click **File | Export CSV**. Define the channels and devices for the project.
2. Define a tag for each device.
3. Export each device or tag group as a CSV file.
4. Use this template in a spreadsheet application that supports CSV files and modify the file as desired.

Note: The resulting CSV file can be saved to disk and re-imported into the server under the same (or new) device or tag group.

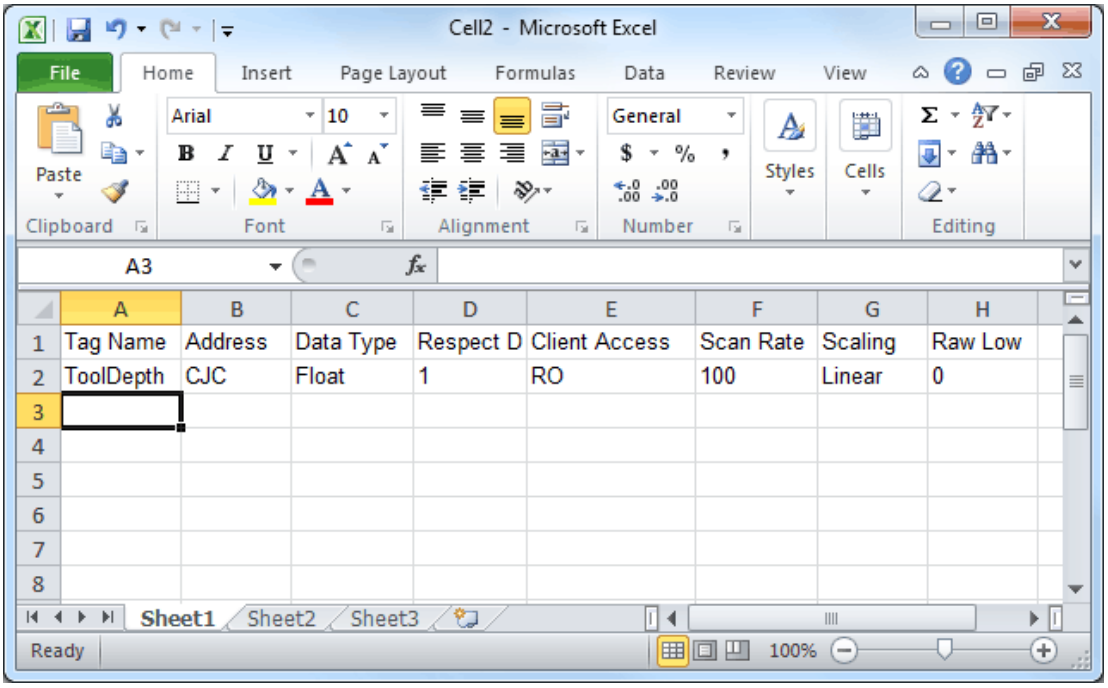
Exporting a Server Tag List

Exporting a server tag list generates a .CSV 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**: The name of the tag as referenced in an OPC client.
  The tag name may contain a group name prefix separated from the tag name with a period. For example, a tag name of "Group1.Tag1" creates a group named "Group1" that contains "Tag1".
- **Address**: The device location referenced by the tag.
- **Data Type**: The data type used for the tag as shown in the server tag's data type drop-down list.
- **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 (read only and read / write).
- **Scan Rate**: The rate in milliseconds at which the tag address is scanned when used with most non-OPC clients.
- **Scaling**: Scaling mode (None, Linear, and 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**: Forces the resulting scaled value to stay within the limit of Scaled Low (1, 0).
- **Clamp High**: Forces the resulting scaled value to stay within the limit of Scaled High (1, 0).
- **Eng. Units**: Units string.
- **Description**: The description of the tag.
- **Negate Value**: Negates the resulting value before being passed to the client when scaling is applied (1, 0).

Note: Each tag record contains the data for each field.

Microsoft Excel is an excellent tool for editing large groups of tags outside 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 shown in the image below.



Importing a CSV Tag List into the Server

Once the tag list has been edited, it can be re-imported into the server by clicking **File | Import CSV**. This option is only available when a device or tag group is selected.

Using Other Characters as the Delimiter

When utilizing a CSV file that does not use a comma or semi-colon delimiter, users should do one of the following:

- Save the project in XML. Then, perform mass configuration on the XML file instead of using CSV.
- Perform a search-and-replace on the delimiter in the CSV file and replace the delimiter with a comma or semicolon. The delimiter being used by the OPC server (either comma or semicolon) must be set to the replacement character.

See Also: Options - General

System Tags

System tags provide general error feedback to client applications, allow operational control when a device is actively collecting data, and allow a channel or device's standard properties to be changed by an OPC client application when needed.

The number of system tags available at both the channel level and device level depends on the nature of the driver being used. In addition, application-level system tags allow client applications to monitor the server's status. System tags can also be grouped according to their purpose as both status and control or property manipulation. Descriptions are as follows:

- **Status Tags** Status tags are read-only tags that provide data on server operation.
- **Parameter Control Tags**: Parameter control tags can be used to modify the server application's operational characteristics. This provides a great deal of flexibility in the OPC applications. By using the property control tags, users can implement redundancy by switching communications links or changing the device ID of a target device. Users can also provide access to the 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

Syntax Example: <*Channel Name*>.<*Device Name*>._*System*._*ActiveTagCount*

Tag	Class	Description
_ActiveTagCount	Status Tag	The _ActiveTagCount tag indicates the number of tags that are currently active in the server. This is a read-only tag.
_ClientCount	Status Tag	The _ClientCount tag indicates the number of clients that are currently connected to the server. This is a read-only tag.
_Date	Status Tag	The _Date tag 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.
_DateTime	Status Tag	The _DateTime tag 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	Status Tag	The _DateTimeLocal tag 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.
_Date_Day	Status Tag	The _Date_Day tag indicates the current day of the month of the system on which the server is running. This is a read-only tag.
_Date_DayOfWeek	Status Tag	The _Date_DayOfWeek tag indicates the current day of the week of the system on which the server is running. The format of the string is a number from 0 (Sunday) to 6 (Saturday). This is a read-only tag.
_Date_Month	Status Tag	The _Date_Month tag indicates the current month of the system on which the server is running. The format of the string is a number (such as "9"

Tag	Class	Description
		instead of "September"). This is a read-only tag.
_Date_Year2	Status Tag	The _Date_Year2 tag indicates the last two digits of the current year of the system on which the server is running. This is a read-only tag.
_Date_Year4	Status Tag	The _Date_Year4 tag indicates the current year of the system on which the server is running. This is a read-only tag.
_ExpiredFeatures	Status Tag	The _ExpiredFeatures tag provides a list of all server features whose time-limited usage has expired. These features are no longer operational. This is a read-only tag.
_FullProjectName	Status Tag	The _FullProjectName tag indicates the fully qualified path and file name to the currently loaded project. This is a read-only tag.
_IsDemo	Status Tag	The _IsDemo tag is no longer available as the runtime will not enter Time Limited mode in version 6.0 or higher. See the _TimeLimitedFeatures, _LicensedFeatures, and _ExpiredFeatures tags to monitor the status of server features.
_LicensedFeatures	Status Tag	The _LicensedFeatures tag provides a list of all server features in use that have a valid license. These features are not subject to a time limit and will continue normal operation after any time-limited features expire. This is a read-only tag.
_OpcClientNames	Status Tag	The _OpcClientNames tag is a String Array that lists the names of all OPC clients that connect to the server and register their name through the IOPCCommon::SetClientName method. This is a read-only tag.
_ProductName	Status Tag	The _ProductName tag indicates the name of the underlying communication server. This is a read-only tag.
_ProductVersion	Status Tag	The _ProductVersion tag indicates the version of the underlying communication server. This is a read-only tag.
_ProjectName	Status Tag	The _ProjectName tag indicates the currently loaded project file name and does not include path information. This is a read-only tag.

Tag	Class	Description
_ProjectTitle	Status Tag	The _ProjectTitle tag is a String tag that indicates the title of the project that is currently loaded. This is a read-only tag.
_Time	Status Tag	The _Time tag 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.
_Time_Hour	Status Tag	The _Time_Hour tag indicates the current hour of the system on which the server is running. This is a read-only tag.
_Time_Hour24	Status Tag	The _Time_Hour24 tag indicates the current hour of the system on which the server is running in a 24 hour format. This is a read-only tag.
_Time_Minute	Status Tag	The _Time_Minute tag indicates the current minute of the system on which the server is running. This is a read-only tag.
_Time_PM	Status Tag	The _Time_PM tag indicates the current AM/PM status of the system on which the server is running. This is a Boolean tag: 0 (False) indicates AM, and 1 (True) indicates PM. This is a read-only tag.
_Time_Second	Status Tag	The _Time_Second tag indicates the current second of the system on which the server is running. This is a read-only tag.
_TimeLimitedFeatures	Status Tag	The _TimeLimitedFeatures tag provides a list of all server features that are time-limited and the time remaining (in seconds). When the time remaining expires, the feature will cease operation. This is a read-only tag.
_TotalTagCount	Status Tag	The _TotalTagCount tag 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 Serial Port Drivers

Syntax Example: <*Channel name*>._System._BaudRate

Tag	Class	Description
_AvailableNetworkAdapters	Status Tag	The _AvailableNetworkAdapters tag 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 that are active, such as a dial-up 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 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.
_BaudRate	Parameter Control Tag	The _BaudRate tag allows the baud rate of the driver to be changed at will. The _BaudRate tag is defined as a long value and 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. This is a read / write tag.
_ComId	Parameter Control Tag	The _ComId tag allows the comm port selection for the driver to be changed at will. 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, users will also need to set the IP number of the remote terminal server. This is done at the device-level and is shown below. This is a read / write tag.
_DataBits	Parameter Control Tag	The _DataBits tag allows the data bits of the driver to be changed at will. The _DataBits tag is defined as a signed 8-bit value. Valid data bits selections are 5, 6, 7 and 8. This is a read / write tag.
_Description	Status Tag	The _Description tag indicates the current user-defined text description for the channel it is referencing. This is a read-only tag.
_EnableDiagnostics	Parameter Control Tag	The _EnableDiagnostics tag 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

Tag	Class	Description
		be available. For more information, refer to **Statistics Tags**. This is a read / write tag.
_EncapsulationPort	Parameter Control Tag	The _EncapsulationPort tag controls the destination port for Ethernet connections. The valid range is 0 to 65535. This is a read / write tag.
_EncapsulationProtocol	Parameter Control Tag	The _EncapsulationProtocol tag controls the protocol used for Ethernet connections. Options include TCP/IP and UDP. This is a read / write tag.
_FloatHandlingType	Parameter Control Tag	The _FloatHandlingType tag allows the current channel-level float handling to be changed. It exists in the channel-level _System folder. For more information, refer to **Channel Properties — Advanced**. This is a read / write tag.
_FlowControl	Parameter Control Tag	The _FlowControl tag allows the flow control setting of the driver to be changed at will. 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. This is a read / write tag.
_InterDeviceDelayMS	Parameter Control Tag	The _InterDeviceDelayMS tag specifies the amount of time that the channel will delay sending a request to the next device after the data has been received from the current device on the same channel. The valid range is 0 to 60000 milliseconds. The default setting is 0. **Note**: This tag is only available on channels that use protocols that utilize the Inter-Device Delay. This is a read / write tag.
_NetworkAdapter	Parameter Control Tag	The _NetworkAdapter tag allows the current NIC adapter in use by the driver to be changed at will. 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 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.

Tag	Class	Description
		Note: When changing the NIC selection the driver is forced to break all current device connections and reconnect. This is a read / write tag.
_Parity	Parameter Control Tag	The _Parity tag allows the parity of the driver to be changed at will. 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. This is a read / write tag.
_ReportComErrors	Parameter Control Tag	The _ReportComErrors tag 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 the environment contains a lot of electrical noise, this feature can be disabled to prevent the Event Log from filling with error messages. This is a read / write tag.
_RtsLineDrop	Parameter Control Tag	The _RtsLineDrop tag 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 to 9999 milliseconds. The Manual RTS mode has been designed for use with radio modems. This is a read / write tag.
_RtsLinePollDelay	Parameter Control Tag	The _RtsLinePollDelay tag 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 to 9999 milliseconds. The Manual RTS mode has been designed for use with radio modems. This is a read / write tag.
_RtsLineRaise	Parameter Control Tag	The _RtsLineRaise tag 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 _

Tag	Class	Description
		RtsLineRaise is defined as a long value. The valid range is 0 to 9999 milliseconds. The Manual RTS mode has been designed for use with radio modems. This is a read / write tag.
_SharedConnection	Status Tag	The _SharedConnection tag indicates that the port settings are being shared with another channel. This is a read-only tag.
_StopBits	Parameter Control Tag	The _StopBits tag allows the stop bits of the driver to be changed at will. The _StopBits tag is defined as a signed 8-bit value. Valid data bit selections are 1 and 2. This is a read / write tag.
_UnsolicitedEncapsulationPort	Parameter Control Tag	The _UnsolicitedEncapsulationPort tag controls the Ethernet port that has been opened to allow connections. The valid range is 0 to 65535. This is a read / write tag.
_UnsolicitedEncapsulationProtocol	Parameter Control Tag	The _UnsolicitedEncapsulationProtocol tag controls the Ethernet protocol used to connect to the Unsolicited Encapsulation Port. Options include TCP/IP and UDP. This is a read / write tag.
_WriteOptimizationDutyCycle	Parameter Control Tag	The _WriteOptimizationDutyCycle tag allows the duty cycle of the write to read ratio to be changed at will. 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. For more information, refer to **Channel Properties — Write Optimizations**. This is a read / write tag.

Channel-Level System Tags for Ethernet Drivers

Syntax Example: <*Channel name*>._*System*._*NetworkAdapter*

Tag	Class	Description
_AvailableNetworkAdapters	Status Tag	The _AvailableNetworkAdapters tag lists the available NICs and includes both unique NIC cards and NICs that have multiple IPs assigned to them. Additionally this tag also displays any WAN connections that are active, such as a dial-up 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 contains all of the NIC names and their IP assignments. A semi-colon separates each unique NIC to allow the names to

Tag	Class	Description
		be parsed within an OPC application. For a serial driver, this tag is only used if Ethernet Encapsulation is selected. This is a read-only tag.
_Description	Status Tag	The _Description tag indicates the current user-defined text description for the channel it is referencing. This is a read-only tag.
_EnableDiagnostics	Parameter Control Tag	The _EnableDiagnostics tag 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. For more information, refer to **Statistics Tags**. This is a read / write tag.
_EncapsulationPort	Parameter Control Tag	The _EncapsulationPort tag controls the port used for Ethernet connections. The valid range is 0 to 65535. This is a read / write tag.
_EncapsulationProtocol prop	Parameter Control Tag	The _EncapsulationProtocol tag controls the protocol used for Ethernet connections. Options include TCP/IP and UDP. This is a read / write tag.
_FloatHandlingType	Parameter Control Tag	The _FloatHandlingType tag allows the current channel-level float handling to be changed. It exists in the channel-level _System folder. For more information, refer to **Channel Properties — Advanced**. This is a read / write tag.
_InterDeviceDelayMS	Parameter Control Tag	The _InterDeviceDelayMS tag specifies the amount of time that the channel will delay sending a request to the next device after the data has been received from the current device on the same channel. The valid range is 0 to 60000 milliseconds. The default setting is 0. **Note**: This tag is only available on channels that use protocols that utilize the Inter-Device Delay. This tag is a read / write tag.
_NetworkAdapter	Parameter Control Tag	The _NetworkAdapter tag allows the current NIC adapter in use by the driver to be changed at will. As a string tag, the name of the newly desired NIC adapter must be writ-

Tag	Class	Description
		ten to this tag in string format. The string written must match the exact description of the desired NIC 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 is forced to break all current device connections and reconnect. This is a read / write tag.
_UnsolicitedEncapsulationPort	Parameter Control Tag	The _UnsolicitedEncapsulationPort tag controls the Ethernet port that has been opened to allow connections. The valid range is 0 to 65535. This is a read / write tag.
_UnsolicitedEncapsulationProtocol	Parameter Control Tag	The _UnsolicitedEncapsulationProtocol tag controls the Ethernet protocol used to connect to the Unsolicited Encapsulation Port. Options include TCP/IP and UDP. This is a read / write tag.
_WriteOptimizationDutyCycle	Parameter Control Tag	The _WriteOptimizationDutyCycle tag allows the duty cycle of the write to read ratio to be changed at will. 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. For more information, refer to **Channel Properties — Write Optimizations**. This is a read / write tag.

Device-Level System Tags for both Serial and Ethernet Drivers

Syntax Example: <*Channel Name*>.<*Device Name*>._*System*._*Error*

Tag	Class	Description
_AutoCreateTagDatabase	Parameter Control Tag	The _AutoCreateTagDatabase tag is a Boolean tag that is used 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. This is a read / write tag.
_AutoDemoted	Status Tag	The _AutoDemoted tag is a Boolean tag that returns the current auto-demoted state of the device. When False, the device

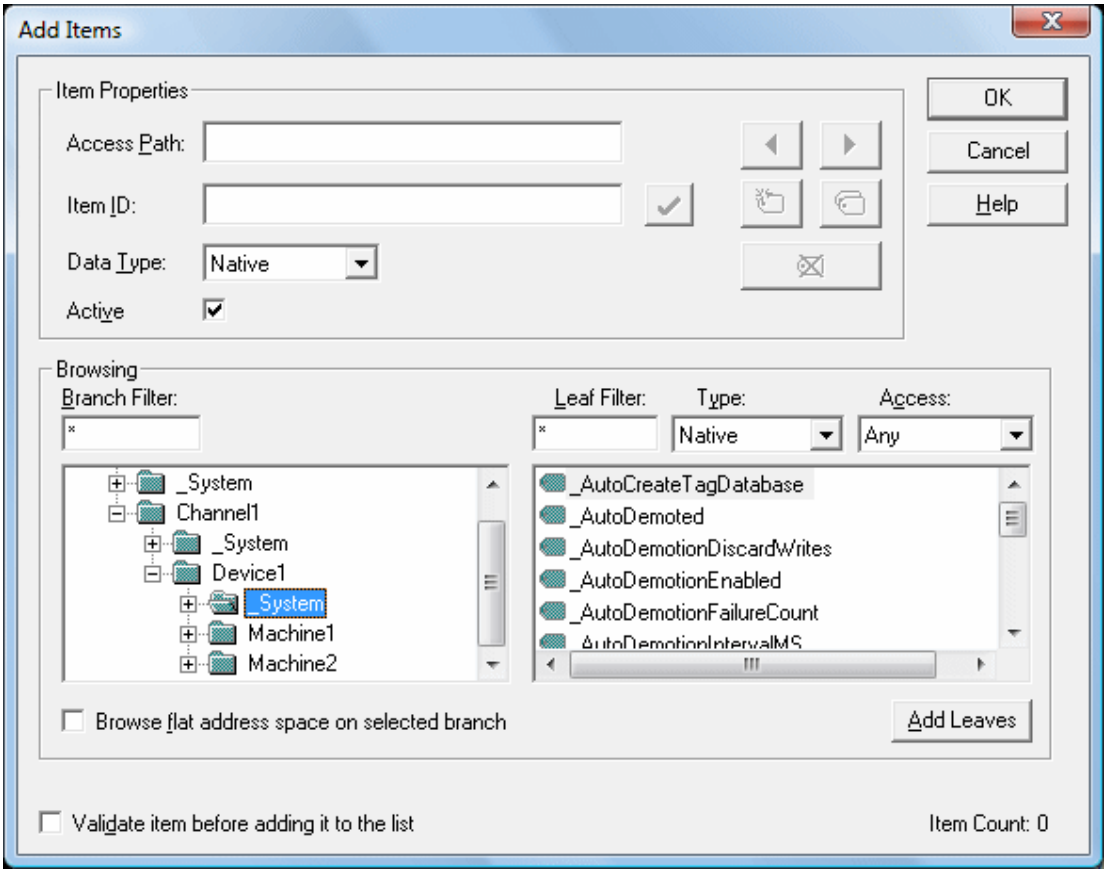
Tag	Class	Description
		is not demoted and is being scanned by the driver. When set True, the device is in demoted and not being scanned by the driver. This is a read-only tag.
_AutoDemotionDiscardWrites	Parameter Control Tag	The _AutoDemotionDiscardWrites tag is a Boolean tag that specifies whether or not write requests should be discarded during the demotion period. When this tag is set to False, all writes requests are performed regardless of the _AutoDemoted state. When this tag is set to True, all writes are discarded during the demotion period. This is a read / write tag.
_AutoDemotionEnabled	Parameter Control Tag	The _AutoDemotionEnabled tag is a Boolean tag that allows the device to be automatically demoted for a specific time period when the device is unresponsive. When this tag is set False, the device will never be demoted. When this tag is set True, the device is demoted when the _AutoDemotedFailureCount has been reached. This is a read / write tag.
_AutoDemotedFailureCount	Parameter Control Tag	The _AutoDemotedFailureCount tag specifies how many successive failures it takes to demote a device. The _AutoDemotedFailureCount is defined as a long data type. The valid range is 1 to 30. This tag can only be written to if _AutoDemotionEnabled is set to True. This is a read / write tag.
_AutoDemotionIntervalMS	Parameter Control Tag	The _AutoDemotionIntervalMS tag specifics how long, in milliseconds, a device is demoted before re-attempting to communicate with the device. The _AutoDemotionIntervalMS is defined as a long data type. The valid range is 100 to 3600000 milliseconds. This tag can only be written to if _ AutoDemotionEnabled is set to True. This is a read / write tag.
_ConnectTimeout	Parameter Control Tag	The _ConnectTimeout tag allows the timeout associated with making an IP connection to a device to be changed at will. This tag is available when either a native Ethernet driver is in use or a serial driver is in Ethernet Encapsulation mode. The _ ConnectTimeout is defined as a Long data type. The valid range is 1 to 30 seconds. This is a read / write tag.
_DemandPoll	Status / Control Tag	The _DemandPoll tag issues a device read to all the active client items associated with the device. This is the equivalent of a client performing an asynchronous device read for those

Tag	Class	Description
		items. It takes priority over any scheduled reads that are supposed to occur for items that are being actively scanned. The _DemandPoll tag becomes True (1) when written to. It returns to False (0) when the final active tag signals that the read requests have completed. Subsequent writes to the _DemandPoll tag will fail until the tag value returns to False. The demand poll respects the read / write duty cycle for the channel. This is a read / write tag.
_Description	Status Tag	The _Description tag indicates the current user-defined text description for the device it is referencing. This is a read-only tag.
_DeviceId	Parameter Control Tag	The _DeviceId tag allows the ID of the device to be changed at will. The data format of the _DeviceId depends on the type of device. For most serial devices this tag is a Long data type. For Ethernet drivers the _DeviceId is 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. This is a read / write tag.
_Enabled	Parameter Control Tag	The _Enabled 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 is 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. This is a read / write tag.
_EncapsulationIp	Parameter Control Tag	The _EncapsulationIp tag allows the IP of a remote terminal server to be specified and changed at will. This tag is only available on serial drivers that support **Device Properties — Ethernet Encapsulation** mode. The _EncapsulationIp 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. This is a read / write tag.
_EncapsulationPort	Parameter Control Tag	The _EncapsulationPort tag allows the port number of the remote terminal server to be specified and changed. The _EncapsulationPort is defined as a long data type. The valid range is 0 to 65535. The port number entered in this tag must

Tag	Class	Description
		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. This is a read / write tag.
_EncapsulationProtocol	Parameter Control Tag	The _EncapsulationProtocol tag allows the IP protocol used for Ethernet Encapsulation to be specified and changed. The _EncapsulationProtocol 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. This is a read / write tag.
_Error	Status Tag	The _Error 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. **Note**: For more information, refer to **Device Properties — Timing**. This is a read-only tag.
_FailedConnection	Status Tag	The _FailedConnection tag specifies that the connection failed. It is only available to specific drivers. This is a read-only tag.
_InterRequestDelay	Parameter Control Tag	The _InterRequestDelay tag allows the time interval between device transactions to be changed at will. The _InterRequestDelay is defined as a Long data type. The valid range is 0 to 30000 milliseconds. This tag only applies to drivers that support this feature. This is a read / write tag.
_RequestAttempts	Parameter Control Tag	The _RequestAttempts tag allows the number of communication attempts to be changed. The _RequestAttempts is defined as a Long value. The valid range is 1 to 10 attempts. This tag applies to all drivers equally. This is a read / write tag.
_RequestTimeout	Parameter Control Tag	The _RequestTimeout tag allows the timeout associated with a data request to be changed at will. The _RequestTimeout tag is defined as a Long value. The valid range is 100 to 30000 milliseconds. This tag applies to all drivers equally. This is a read / write tag.

Tag	Class	Description
_NoError	Status Tag	The _NoError 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. **Note:** For more information, refer to **Device Properties — Timing**. This is a read-only tag.
_ScanMode	Status Tag	The _ScanMode tag allows clients to dictate the method used for updates. It is defined as a String value, and corresponds to the user-specified Scan Mode setting (located in device properties). "Respect client specified scan rate" has a value of "UseClientRate," "Request data no faster than *x*" has a value of "UseFloorRate," and "Request all data at *x*" has a value of "ForceAllToFloorRate." The default setting is "Respect client specified scan rate." This is a read-only tag.
_ScanRateMs	Status Tag	The _ScanRateMs tag corresponds to the _ScanMode tag, and is used when the Scan Mode is set to Request Data No Faster than Scan Rate or Request All Data at Scan Rate. This tag is defined as a DWord tag. The default setting is 1000 milliseconds. This is a read-only tag.
_SecondsInError	Status Tag	The _SecondsInError tag is a DWord tag that displays the number of seconds since the device entered an error state. This tag displays 0 when the device is not in an error state. This is a read-only tag.
_Simulated	Parameter Control Tag	The _Simulated 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 returns good data for this device, but does not attempt to communicate with the actual physical device. When tag is read as False, communication with the physical device is active. Changing the tag value allows clients to enable / disable simulated mode. This is a read/write tag.

When using an OPC client, the System tags are found under the _System branch of the server browse space for a given device. The following image 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 "<DDE service name>|_ddedata!Channel1.Device1._System._Error".

The _Enabled tag provides a very flexible means of controlling the 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.

See Also:
Property Tags
Modem Tags
Statistics Tags
Store and Forward Tags

Property Tags

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. For more information, refer to Tag Properties — General.

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".

Supported Property Tag Names

Tag Name	Description
_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.
_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
Modem Tags
System Tags

Statistics Tags

Statistics tags are used to provide feedback to client applications regarding the operation of the channel communications in the server. Statistics tags are only available when diagnostics are enabled. *For more information, refer to Channel Diagnostics and OPC Diagnostics Viewer.*

Syntax Example: <*Channel Name*>.*_Statistics._FailedReads*

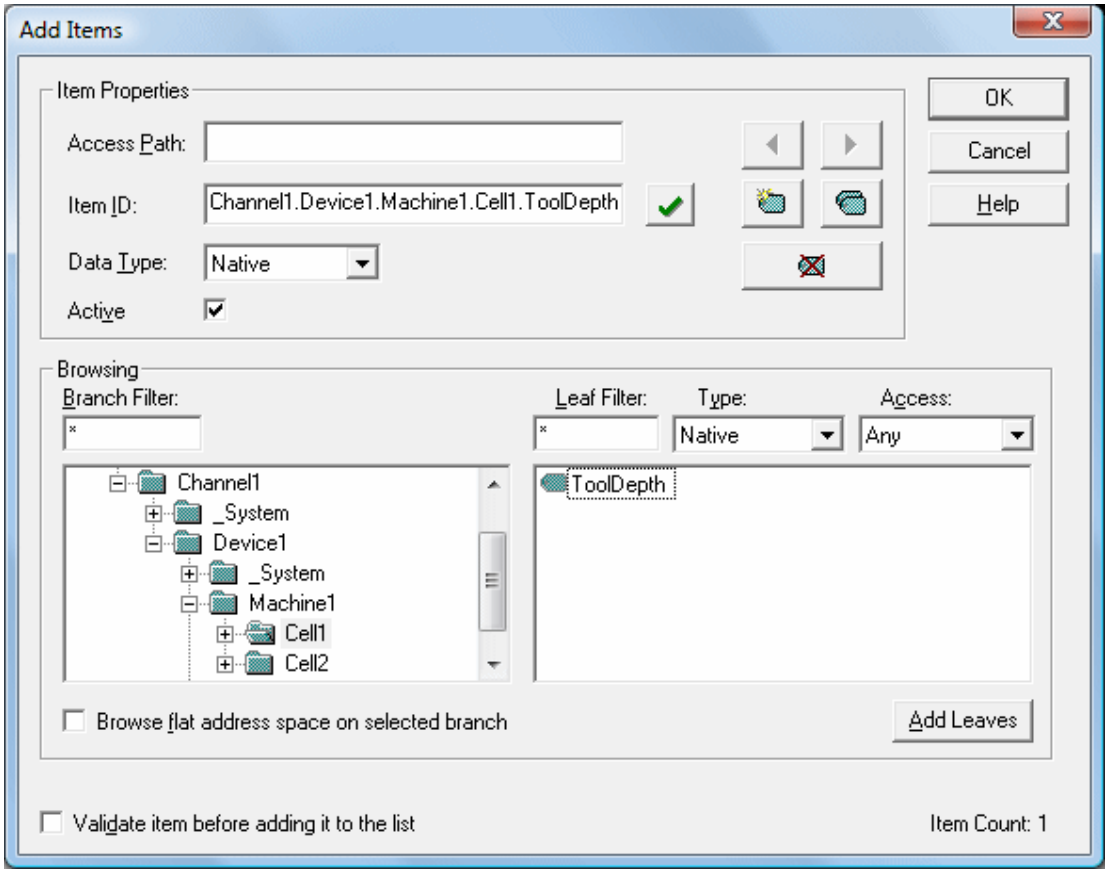
Supported Statistics Tag Names

Tag Name	Description
_SuccessfulReads	The _SuccessfulReads 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 _Reset tag was invoked. This tag is formatted as unsigned 32-bit integer and will eventually rollover. This tag is read only.
_SuccessfulWrites	The _SuccessfulWrites 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 _Reset tag was invoked. This tag is formatted as an unsigned 32-bit integer and will eventually rollover. This tag is read only.
_FailedReads	The _FailedReads 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 _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 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.
_MaxPendingReads	The _MaxPendingReads tag contains a count of the maximum number of pending read requests for the channel since the start of the application (or the _Reset tag) was invoked. This tag is formatted as an unsigned 32-bit integer. The tag is read only.
_MaxPendingWrites	The _MaxPendingWrites tag contains a count of the maximum number of pending write requests for the channel since the start of the application (or the _Reset tag) was invoked. This tag is formatted as an unsigned 32-bit integer. The tag is read only.
_NextReadPriority	The _NextReadPriority is a channel-level system tag that reflects the priority level of the next read in the channel's pending read queue. Possible values are -1: No pending reads. 0: The next read is a result of a schedule-level demand poll or explicit read from a client. 1 - *n*: The next read is a result of scheduled read. This tag is read only.

Tag Name	Description
_PendingReads	The _PendingReads tag contains a count of the current pending read requests for the channel. This tag is formatted as an unsigned 32-bit integer. The tag is read only.
_PendingWrites	The _PendingWrites tag contains a count of the current pending write requests for the channel. This tag is formatted as an unsigned 32-bit integer. This tag is read only.

* This statistical item is not updated in simulation mode (*See Device Properties*).

Statistics tags are only available when diagnostics are enabled. 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 image is taken from the OPC Quick Client, and shows how a Diagnostics tag appears to an OPC client.



The _Statistics branch (located beneath the channel branch) only appears when diagnostics are enabled for the channel. To reference a Diagnostics tag from a DDE application, given the above example and the DDE defaults, the link would appear as: "<DDE service name>|_ddedata!Channel1._Statistics._SuccessfulReads".

The Diagnostics tag's value can also be viewed in the server by using the Communication Diagnostics Viewer. If Diagnostics Capture is enabled under Channel Properties, right-click on that channel and select **Diagnostics**.

See Also:
System Tags
Property Tags

Modem Tags

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

Syntax Example: <*Channel Name*>.<*Device Name*>._*Modem*._*Dial*

Supported Modem Tag Names

Tag Name	Description	Access
_Dial	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) generates an error.	Read / Write
_DialNumber	The 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.	Read Only
_Hangup	Writing any value to this tag hangs up the current connection. The Hangup tag ends the current connection when an external device has called the server. Writes to the Hangup tag are ignored if the Status <= 3 (Idle), meaning that there is no currently open connection.	Read / Write
_LastEvent	Whenever the Status changes, the reason for the change is set in this tag as a number. *For a list of event numbers and meanings, refer to* ***Last Event Values***.	Read Only
_Mode	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 are 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).	Read / Write
_PhoneNumber	This is the current phone number to be dialed. Users can write to this value at any time, but the change is only effective if Status is <= 3 (Idle). If users write to the phone number while the status is greater than 3, the number is queued. As soon as the status drops to 3 or less, the new number is transferred to the tag. The queue is of size 1, so only the last phone number written is retained. The phone number must be in canonical format to apply the dialing preferences. If the canonical format is used, the resulting number to 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	Read / Write

Tag Name	Description	Access
	Note: The country code for the U.S. is 1. If the number is not in canonical form, dialing preferences are not applied. The number is dialed exactly as it is entered. Users can also enter a Phone-book tag name instead of a phone number. In this case, the current value of the Phonebook tag is used.	
_Status	This is the current status of the modem assigned to a channel. For a list of status values and meanings, refer to **Status Values**.	Read Only
_StringLastEvent	This contains a textual representation of the LastEvent tag value. For a list of event numbers and meanings, refer to **Last Event String Values**.	Read Only
_StringStatus	This contains a textual representation of the Status tag value. For a list of event numbers and meanings, refer to **Status String Values**.	Read Only

Status Values

The five lowest bits of the 32-bit status variable are currently being used.

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

When read as an integer, the value of the Status tag is always 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

Status String Values

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

Last Event Values

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

Last Event String Values

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

Communication Serialization Tags

Syntax Example: <*Channel Name*>.*_CommunicationSerialization._VirtualNetwork*

Tag	Description
_NetworkOwner	The _NetworkOwner tag indicates if the channel currently owns control of communications on the network. The frequency of

Tag	Description
Class: Status Tag	change reflects how often the channel is granted control. This tag is read only.
_Registered Class: Status Tag	The _Registered tag indicates whether the channel is currently registered to a virtual network. After setting the _VirtualNetwork, the channel unregisters from the network it is currently registered to (indicated in _RegisteredTo) when it is capable of doing so. In other words, if the channel owns control during the switch, it cannot unregister until it has released control. Upon unregistering, the channel registers with new virtual network. This tag is FALSE if _VirtualNetwork is None. This tag is read only.
_RegisteredTo Class: Status Tag	The _RegisteredTo tag indicates the virtual network to which the channel is currently registered. After setting the _VirtualNetwork, the channel unregisters from the network it is currently registered to when it is capable of doing so. In other words, if the channel owns control during the switch, it cannot unregister until it has released control. Upon unregistering, the channel registers with new virtual network. This tag indicates if there are delays switching networks as _VirtualNetwork and _RegisteredTo could differ for a period of time. This tag is N/A if _VirtualNetwork is None. This tag is read only.
_StatisticAvgNetworkOwnershipTimeSec Class: Status Tag	The _StatisticAvgNetworkOwnershipTimeSec tag indicates how long on average the channel holds ownership of control since the start of the application (or since the last time _StatisticsReset was written to). This tag helps identify busy channels/bottlenecks. This tag is formatted as a 32-bit floating point and may eventually rollover. This tag is read only.
_StatisticNetworkOwnershipCount Class: Status Tag	The _StatisticNetworkOwnershipCount tag indicates the number of times the channel has been granted control of communications since the start of the application (or since the last time _StatisticsReset was written to). This tag is formatted as an unsigned 32-bit integer and may eventually rollover. This tag is read only.
_StatisticNetworkOwnershipTimeSec Class: Status Tag	The _StatisticNetworkOwnershipTimeSec tag indicates how long in seconds the channel has held ownership since the start of the application (or since the last time _StatisticsReset was written to). This tag is formatted as a 32-bit floating point and may eventually rollover. This tag is read only.

Tag	Description
_StatisticsReset	The _StatisticsReset tag can be used to reset all the statistic counters. The _StatisticsReset tag is formatted as a Boolean tag. Writing a non-zero value to the _StatisticsReset tag causes the statistics counters to be reset. This tag is read / write.
_TransactionsPerCycle	The _TransactionsPerCycle tag indicates the number of read / write transactions that occur on the channel when taking turns with other channels in a virtual network. It allows the channel-level setting to be changed from a client application. This tag is formatted as a signed 32-bit integer (Long). The valid range is 1 to 99. The default setting is 1. This tag is read / write.
_VirtualNetwork Class: Parameter Tag	The _VirtualNetwork tag allows the virtual network selection for the channel to be changed on the fly. As a string tag, the desired virtual network must be written to the tag as a string value using the following possible selections: None, Network 1, Network 2, ---, Network 500. To disable communication serialization, select None. This tag is read / write.

Communications Management

Auto-Demotion

The Auto-Demotion properties 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 offline, 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 re-attempts to communicate with the non-responsive device. If the device is responsive, the device is placed on-scan; otherwise, it restarts its off-scan time period.

For more information, refer to ***Device Properties — Auto-Demotion***.

Network Interface Selection

An NIC card can be 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 its currently assigned IP address. The list of available NICs includes both unique NIC cards and NICs that have multiple IPs assigned to them. The selection displays any WAN connections that may be active (such as a dial-up connection).

Ethernet Encapsulation

The Ethernet Encapsulation mode has been designed to provide communications with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port: the terminal server converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted to a serial form, users can connect standard devices that support serial communications to the terminal server. Using a terminal server device allows users to place RS-232 and RS-485 devices throughout the plant operations while still allowing a single localized PC to access the remotely mounted devices. Furthermore, 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 via the Ethernet network.

*For more information, refer to **How Do I...** and **Device Properties — Ethernet Encapsulation**.*

Modem Support

This server supports the use of modems to connect to remote devices, which 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.

Note: Not all serial drivers support the use of modems. *To determine modem support, refer to the specific driver's help documentation.*

When accessing the **modem systems tags**, the channel name can be used as either a base group or topic name. To be available, modems must be configured with the operating system through the Control Panel settings.

Once the modem has been properly installed, it can be enabled by selecting **Modem** as the Physical Medium in the **channel properties**.

For specific setup information, refer to the Windows and modem documentation.

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, an external modem (which can be configured to dial using specific Baud Rate and Parity settings) is strongly recommended. To determine the best modem for a specific application, refer to Technical Support. For examples on how to use a modem in a project, refer to **Using a Modem in the Server Project**.

Using a Modem in the Server Project

Modems convert serial data from the RS-232 port into signal levels that can be transmitted over the phone line. To do this, they break down each byte of the serial data into bits that are used to generate the signal transmitted. Most modems can convert up to 10 bits of information for every byte of data that is sent. Devices must be able to use 10 bits or less to communicate through a modem. To determine the number of bits being used by a specific device, refer to the formula below.

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

For example, the Modbus RTU Driver is configured to use 8 Data Bits, Even Parity, 1 Stop Bit, and 1 Start Bit. When plugged into the formula, it would be 1 + 8 + 1 + 1, which equals 11 bits. A normal modem could not transmit data to this Modbus device. If Parity is changed to None, it would be 1 + 8 + 0 + 1, which equals 10 Bits. A normal modem could transmit data to this Modbus device.

Some drivers cannot be configured to use a 10-bit or less data format, and so cannot use standard modems. Instead, they require modems that can handle transmitting 11 data bits. For drivers that fall into this category, consult the device's manufacturer for recommendations on an appropriate modem vendor. Modem operation is available for all serial drivers, regardless of driver support for modem operation.

Configuring the Initiating Modem

This 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 provided by the modem's manufacturer for the Windows OS must be installed before the server can use the modem in a project. The Windows Control Panel can be used to install new modems.

For information regarding modem installation and setup, refer to both the Windows and the modem's documentation.

Once the modem has been properly installed, users can begin using it in a server project. The receiving end, or the device modem, must be properly configured before it can be used. Users must confirm that the receiving modem matches the profile provided by the driver.

Cables

Before the project can be used, the cable connection must be configured between the receiving modem and the device. Three cables are required: the existing device communication cable for direct connection, a NULL modem adapter, and a NULL modem cable. A NULL modem cable is connected to the modem, and all pins are connected to the same pins on both ends of the cable. The device communication cable is used to connect to the target device, and usually has pins 2 and 3 reversed. Because the cable being used to talk to the device for the direct connection is working by this point, it can be used on the receiving modem by attaching a NULL modem adapter. Similarly, a PC modem cable runs from the PC to the initiating modem. With the cables in place, a modem can now be used in the application.

Note: NULL modem adapters can be found at most computer stores.

Example: Server-side Modem Configuration

After the modems have been configured and installed, they can be used with the server.

1. To start, load the direct connect project and double-click on the channel name. In **Channel Properties**, open the **Serial Communications** group.
2. In the **Physical Medium** drop-down menu, select **Modem**.
3. In **Modem Settings**, select a modem that is available on the computer.

   Note: Users are not able to select Modem from the Physical Medium drop-down menu if there are none available on the computer. If this occurs, exit the server and attempt to reinstall the modem using the Modem Configuration tools supplied by the operating system.
4. To configure the initiating modem's characteristics, use the properties in **Modem Settings**. *For more information, refer to **Channel Properties — Serial Communications**.*
5. Once finished, click **Apply**. Then, click **OK** to save and exit the Channel Properties.

Using a Modem in an Application

Once modem operation has been enabled, a list of pre-defined tags are available to data clients. These **Modem tags** control and monitor an attached modem, and are contained under the channel name (which has become an active OPC access path through which the Modem tags are accessed). Because the server knows very little about what the application needs for modem control, it does not imply any type of control. By using the predefined Modem tags, users can apply the application's scripting capabilities to control how the server uses the selected modem.

Phonebook

A Phonebook is a collection of Phonebook tags (Phone Numbers) that can be used in place of specifying a telephone number written to the "_PhoneNumber" tag in the Modem system tags. The Phonebook is automatically created for any channel that has the **Physical Medium** set to **Modem**. The data associated with a Phonebook tag is a phone number to be dialed by the server. The act of a client writing to a Phonebook tag causes the server to dial the phone number associated with that tag.

Data Type	Privilege
String	Read / Write

Phonebook tags are created by creating new entries in the Phonebook. To add a new Phonebook entry click on the Phonebook node in the project tree and then click New Phone Number icon.

This opens the Phone Number property editor.

Name: Specify the name of the phone number entry. It will be part of the OPC browse data in the "_Phonebook" system tag group. It can be up to 256 characters in length. While using descriptive names is generally a good idea, some OPC client applications may have a limited display window when browsing the tag space of an OPC server. The Name of a phone number must be unique within a Phonebook.

Number: Specify the phone number to be dialed when the associated Phonebook tag is invoked from an OPC client application. A string of up to 64 digits can be entered.

Description: Enter text to attach a comment to the phone number entry. It can be up to 255 characters in length.

Note: With the server's online full-time operation, these parameters can be changed at any time. Changes made to properties take effect immediately; however, OPC clients that have already connected to this tag are not affected until they release and reacquire the tag.

Auto-Dial Priority

When Auto-Dial has been enabled for the channel, the initial connection request begins by attempting to dial the first entry encountered in the Phonebook. If that attempt is unsuccessful, the next number in the phonebook is attempted and so on. This sequence continues until a modem connection is established or the client releases all references to data that can be supplied by the channel. The order priority that Auto-Dial uses to dial is user defined and can be changed by selecting a Phonebook entry and clicking one of the Priority Change icons as shown below. They can also be changed by opening the context menu for the selected entry.

Example

For a Phonebook entry created and the name set to "Site1":

Syntax Example: <*Channel Name*>._*Phonebook.Site1*

Auto-Dial

Auto-Dial automates the actions required of a client application when modem use is specified within the server project. Without Auto-Dial , these actions (which include connecting, disconnecting, and assigning phone numbers) would be performed by an external client application through the use of channel-level Modem tags. For example, to begin the process of establishing a connection, the client would write a dial string to "<*Channel Name*>._Modem._PhoneNumber" and write a value to "<*Channel Name*>._Modem._Dial". When data from the remote device is no longer needed, the client would end the call by writing to "<*Channel Name*>._Modem._Hangup".

Auto-Dial relieves the client of these responsibilities by automatically dialing phone numbers defined in the Phonebook when attempting to establish a connection. The connection is automatically dropped when there

are no client references to tags that rely on the modem connection. To access the Auto-Dial property, click **Channel Properties | Serial Communications**.

For more information, refer to ***Channel Properties — Serial Communications***.

Modem Connection and Disconnection

The process of establishing a modem connection begins when a client connects to the server Runtime and requests data from a device connection to a channel on which Auto-Dial is enabled. The initial connection request begins by attempting to dial the first phone number encountered in the phonebook. If that attempt is unsuccessful, the next number in the phonebook is attempted and so on. This sequence continues until a modem connection is established or the client releases all references to data that can be supplied by the channel.

Note: When re-establishing a connection, the phonebook entry that last produced a successful connection is used. If no previous phonebook entry was successful (or if the entry has since been deleted), the user-defined sequence of phone numbers is used. The number used for re-dialing is not preserved during server reinitialization or restart.

See Also: ***Phonebook***

Timing

Timing settings (such as how long to wait for a connection before proceeding to the next phone number) are determined by the TAPI modem configuration and not by any specific Modem Auto-Dial setting.

Note: Some drivers do not allow the serial port to close once it has opened. Connections established using these drivers do not experience disconnection until all client references have been released (unless the TAPI settings are configured to disconnect after a period of idle time).

Client Access

Modem tags may be used to exert client-level control over the modem. If Modem Auto-Dialing is enabled, however, write access to the Modem tags is restricted so that only one form of access is possible. The Modem tags' values are updated just as they would if the client were in control of the modem.

Changing the Auto-Dial Settings from the Configuration

The runtime reacts to changes in settings according to the following rules:

- If Auto-Dial is enabled after the client has already dialed the modem and established a connection, the change is ignored until the modem is disconnected. If the client is still requesting data from the channel at the time of disconnection, the initial connection sequence begins.
- If Auto-Dial is enabled while no modem connection exists and data is being requested from the channel by the client, the initial connection sequence begins.
- If Auto-Dial is disabled while an existing auto-dial connection exists, no action is taken and the connection is dropped.

See Also: ***Channel Properties — Serial Communications***

Designing a Project

The following examples use the Simulator Driver supplied with the server to demonstrate the process of creating, configuring, and running a project. The Simulator Driver is a memory-based driver that provides both static and changing data for demonstration purposes. Because it does not support the range of configuration options found in other communication drivers, some examples may use images from other

drivers to demonstrate specific product features. For more information on a specific topic, select a link from the list below.

*For information on software and hardware requirements, refer to **System Requirements**.*

Running the Server

This server can be run as both a service and as a desktop application. When running in the default setting as a service, the server is online at all times. When running as a desktop application, the OPC client can automatically invoke the server when it attempts to connect and collect data. For either process to work correctly, users must first create and configure a project. On start, the server automatically loads the most recently used project.

Initially, users must manually invoke the server. To do so, either double-click the desktop icon or select **Configuration** from the Administration menu located in the System Tray. The interface's appearance depends on the changes made by the user.

Once the server is running, a project may be created.

*For more information on the server elements, refer to **Basic Server Components**. For more information on the user interface, refer to **Navigating the Configuration**.*

Starting a New Project

Users must configure the server to determine what content is provided during operation. A server project includes the definition of channels, devices, 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, saved, and loaded.

Some configuration options are global and applied to all projects. These global options are configured in the **Tools | Options** dialog, which includes both General Options and Runtime Connection Options. These settings are stored in a Windows INI file called "settings.ini," which is stored in the Application Data directory selected during installation. Although global options are usually stored in the Windows registry, the INI file supports the copying of these global settings from one machine to another.

The software opens initially with a default project open. That file can be edited, saved, and closed like any other file.

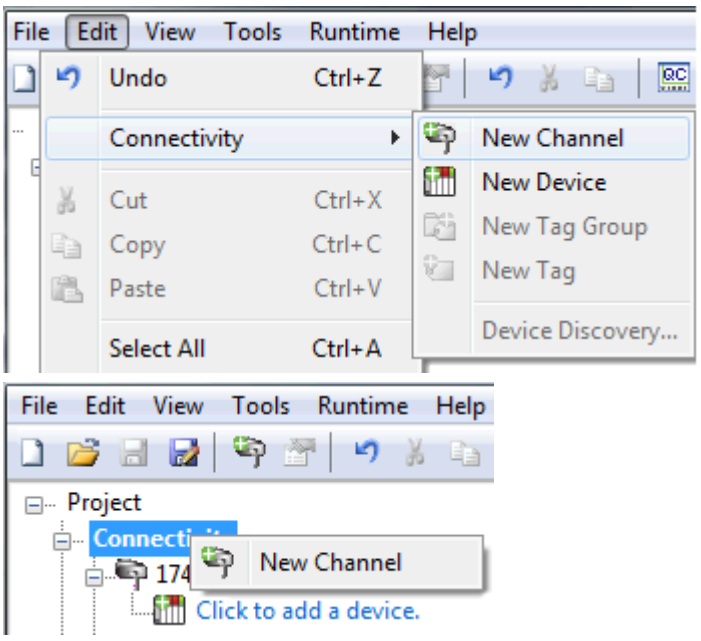
1. To define a new project, choose **File | New**.
2. If prompted to close, save, or edit offline.
3. Choose **File | Save As**.
4. Enter a password to secure the encrypted project file.
5. Choose the location in which to store the file.
6. Click **Save**.
7. Begin configuring the project file by **Adding a Channel**.

***See Also**: Options - General, Saving a Project*

Adding and Configuring a Channel

When creating a new project, users must first determine the communications driver that is required by the application: this is referred to as a channel in the server. A number of channels can be defined within a single project, depending on the driver or drivers installed. For more information, refer to the instructions below.

1. To start, add a new channel to the project by:
   clicking **Edit | Connectivity | New Channel** - OR -
   clicking the **New Channel** icon on the toolbar - OR -
   right-clicking on the **Connectivity** node in the tree and choosing **New Channel**



2. In the **channel wizard**, leave the channel name at its default setting "Channel1". Then, click **Next**.
3. In **Device Driver**, select the communications driver to be applied to the channel. Then, click **Next**. In this example, the Simulator Driver is used.

4. For the Simulator Driver, the next page is **Channel Summary**. Other devices may have additional channel wizard pages that allow the configuration of other properties (such as communications port, baud rate, and parity). *For more information, refer to* ***Channel Properties — Serial Communication***.

5. Once complete, click **Finish**.

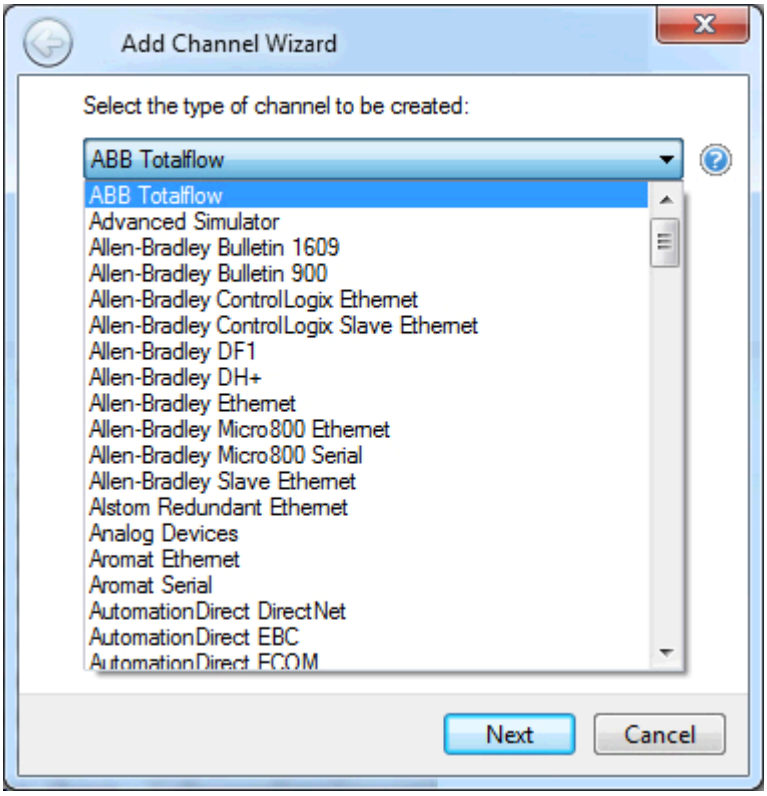
See Also: ***How to... Optimize the Server Project*** , ***Server Summary Information***

Channel Creation Wizard

The Channel Creation Wizard steps through the process of configuring a channel (defined by the protocol being used). Once a channel is defined, its properties and settings are used by all devices assigned to that channel. The specific properties are dependent on the protocol or driver selected.

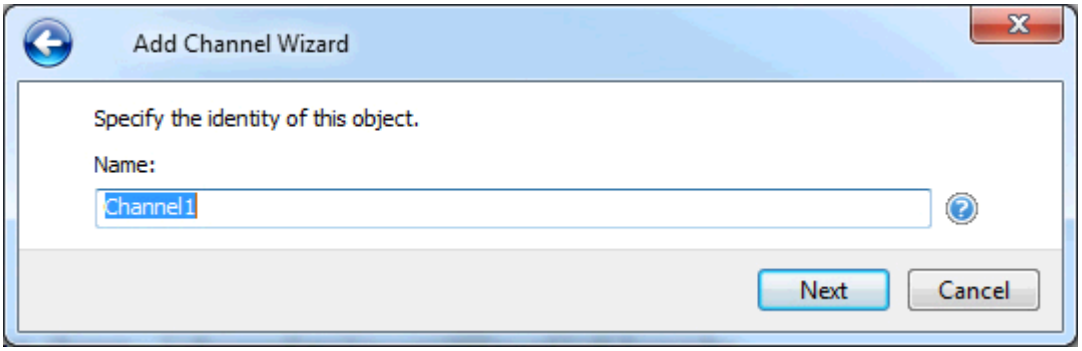
1. In the tree view, right-click on the **Connectivity** node and select **New Channel** (or choose **Edit | Connectivity | New Channel**).

2. Select type of channel to be created from the drop-down list of available drivers.



3. Click **Next**.

4. Enter a name for the channel to help identify it (used in tag paths, event log messages, and aliasing).



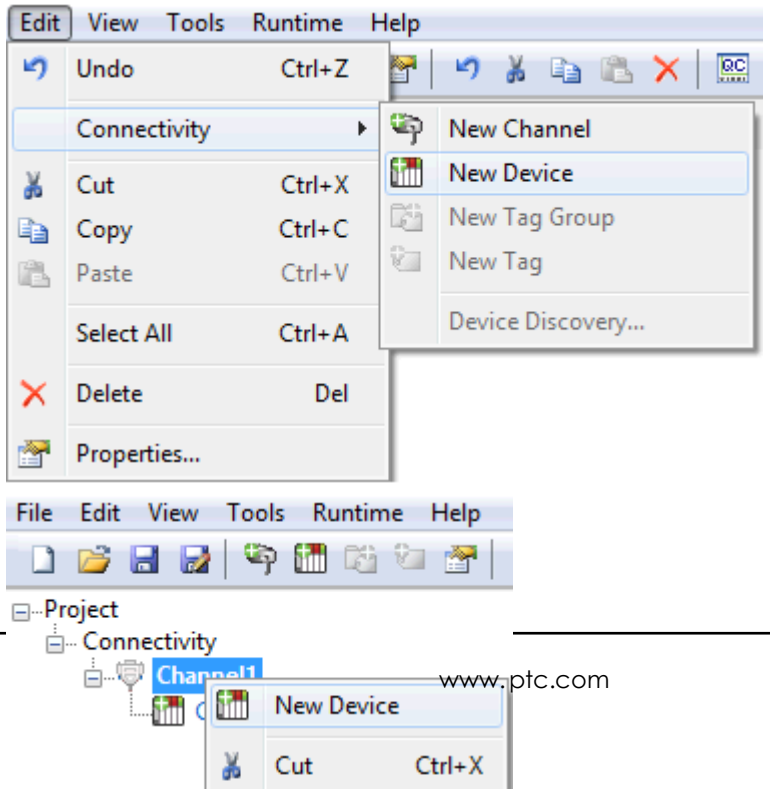
5. Click **Next**.
6. Configure the channel properties according to the options and environment.
7. Review the summary for the new channel and choose **Back** to make changes or **Finish** to close.

Adding and Configuring a Device

Once a channel has been defined, a device can be added. The device identifies a communication link's physical node or station, and can be thought of as a way to frame the connection's definition to a specific point of interest in the application. In this respect, a device is the correct term for describing the connection to a database object. As such, "device" refers to a specific device on a network, support multiple device nodes, and allows users to simulate networked devices.

Note: In this example, the Simulator Driver is used. The options in device wizard depend on the driver.

1. To start, select the channel to which the device will be added.
2. To start, add a new device to the project by:
   clicking **Edit | Connectivity | New Device** - OR -
   clicking the **New Device** icon on the toolbar - OR -
   right-clicking on the **Connectivity** node in the tree and choosing **New Device**



3. In the device wizard, leave the name at its default setting "Device1" and click **Next**.
4. In **Model**, select either an 8 or 16-bit register size for the device being simulated and click **Next**.

   Note: Other device drivers may require users to select a device model instead. For this example, the 16-bit register size is chosen.

5. In **ID**, select the device ID (which is the unique identifier required by the actual communications protocol). Then, click **Next**.

   Note: The device ID format and style depend on the communications driver being used. For the Simulator Driver, the device ID is a numeric value.

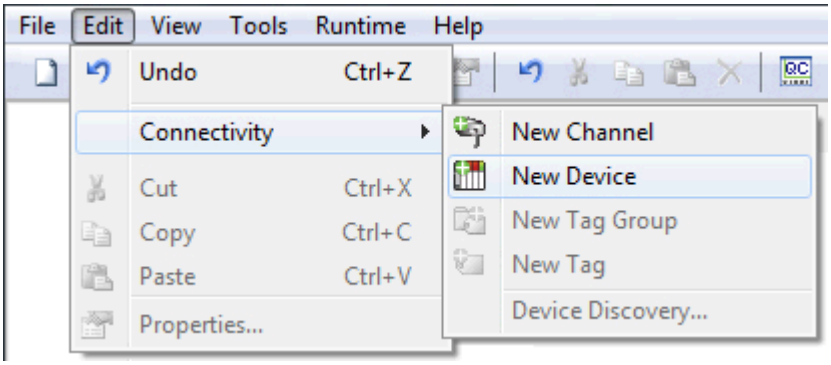
6. In **Scan Mode**, specify the device's scan rate. Then, click **Next**.
7. For the Simulator Driver, the next page is the **Device Summary**. Other drivers may have additional device wizard pages that allow the configuration of other properties (such as Timing). For more information, refer to Device Properties.
8. Once complete, click **Finish**.

Note: With the server's online full-time mode of operation, the server can start providing OPC data immediately. At this point, however, the configuration can potentially be lost because the project hasn't been saved. Before saving, users can add tags to the server. For more information, refer to Adding User-Defined Tags.

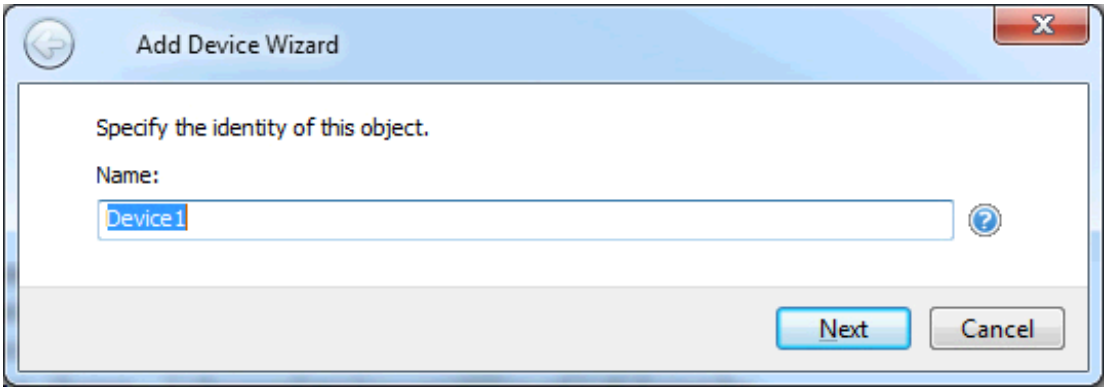
Device Creation Wizard

The Device Creation Wizard steps through the process of configuring a device for communication and data collection. The specific properties are dependent on the protocol or driver selected.

1. In the tree view, locate and select the channel to which device(s) are being added.
2. Right-click and select **New Device** or choose **Edit | Connectivity | New Device**).



3. Enter a name for the device to help identify it (used in tag paths, event log messages, and aliasing).



4. Click **Next**.
5. Configure the **device properties** according to the options and environment.
6. Review the summary for the new device and choose **Back** to make changes or **Finish** to close.

Adding User-Defined Tags (Example)

The server can get data from a device to the client application in two ways. The most common method requires that users define a set of tags in the server project and uses the name previously assigned to each tag as the item of each link between the client and the server. This method makes all user-defined tags available for browsing within OPC clients.

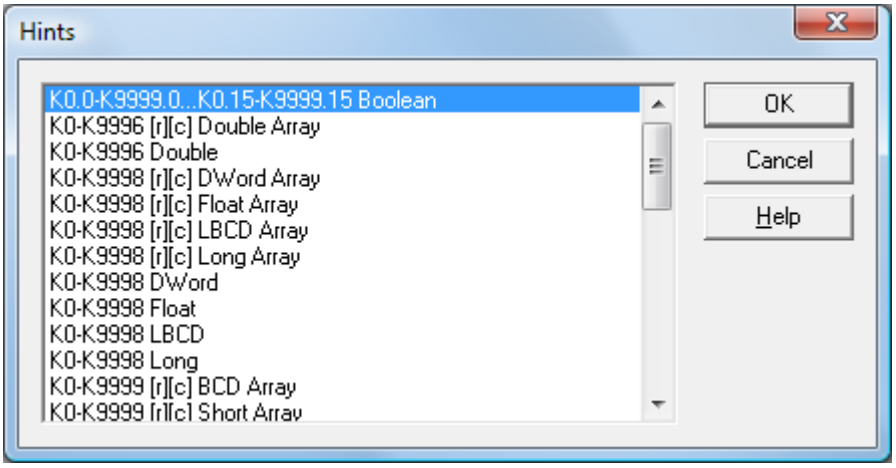
*User-defined tags support scaling. For more information, refer to **Adding Tag Scaling**.*
*Some situations support browsing for and selecting multiple tags. For more information, refer to **Browsing for Tags**.*

1. To start, select a device name from the Connectivity tree node. In this example, the selected device is "Device1".
2. Click **Edit | Connectivity | New Tag**. Alternatively, right-click on the device and select **New Tag**.
3. In **Tag Properties — General**, edit the properties to match the following:

    - **Tag Name** MyFirstTag
    - **Address** R000
    - **Description (Optional)** My First Simulator Tag
    - **Data Type** Word
    - **Client Access** read / write
    - **Scan Rate** 100 milliseconds. This property does not apply to OPC tags.

    *For more information, refer to **Tag Properties — General**.*

4. If necessary, use **Hints** to determine the driver's correct settings. To invoke Hints, click on the question mark icon in Tag Properties.



Note The Address, Data Type, and Client Access fields depend on the communications driver. For example, in the Simulator Driver, "R000" is a valid address that supports a data type of Word and has read / write access.

5. For additional information, click **Help**. This invokes the "Address Descriptions" topic in the driver's help documentation.
6. Commit the tag to the server by pressing **Apply**. The tag should now be visible in the server.
7. In this example, a second tag must be added for use in **Tag Properties — Scaling**. To do so, click the **New** icon in **Tag Properties — General**. This returns the properties to their default setting.
8. Enter the following:
   - **Tag Name** MySecondTag
   - **Address** K000
   - **Description** My First Scaled Tag
   - **Data Type** Short
   - **Client Access** read / write
9. Next, commit the new tag to the server by pressing **Apply**. The tag should now be visible in the server.

Error Messages

When entering tag information, users may be presented with an occasional error message from the server or driver. The server generates error messages when users attempt to add a tag using the same name as an existing tag. The communications driver generates errors for three possible reasons:

1. If there are any errors entered in the address's format or content (including in the range of a particular device-specific data item).
2. When the selected data type is not available for the address.
3. If the selected client access level is not available for the address.

*For more information on a specific error message, refer to **Error Descriptions**.*

Dynamic Tag Addressing

Dynamic tag addressing defines 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 only need to create a tag item in the client that directly accesses the device address. On client connect, the server creates a virtual tag for that location and start scanning for data automatically.

For more information, refer to ***Dynamic Tags***.

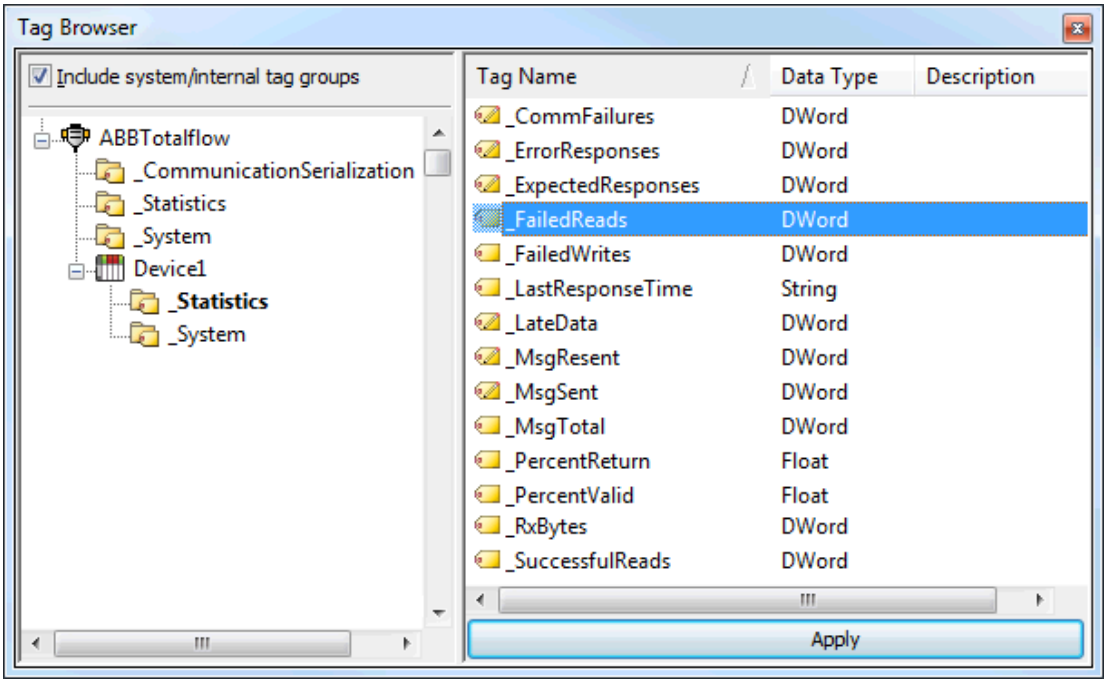
Tips:

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". This tag is zero if the device is communicating properly, or one if the device is not.
2. If the data type is omitted, the driver chooses a default data type based on the device and address being referenced. The default data types for all locations are documented in the driver's help documentation. If the data type specified is not valid for the device location, the server rejects the tag and an error posts in the Event Log.
3. 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 references the address pointed to by the user-defined tag. With the server's online full-time operation, users can start using this project in an OPC client at this time.

Browsing for Tags

The server supports browsing for available tags and, in some cases, selecting multiple tags to add to a project.

1. Access the Tag Browser dialog box.



2. If the **Include system / internal tag groups** is available, enable to enable making these groups available for selection.

3. If the **Branch level tag selection** is available, enable to enable selection of branch nodes in the tree view on the left (which selects all the associated tags in the right).
4. Navigate the tree in the left pane to locate the branch containing the tag(s) to add.
5. Unless **Branch level tag selection** is enabled, select the tag(s) in the right pane. Where adding multiple tags is supported, standard keyboard functions (Shift, Ctrl) work to select multiple tags.
6. Click **Apply**.

See Also: Adding User Tags

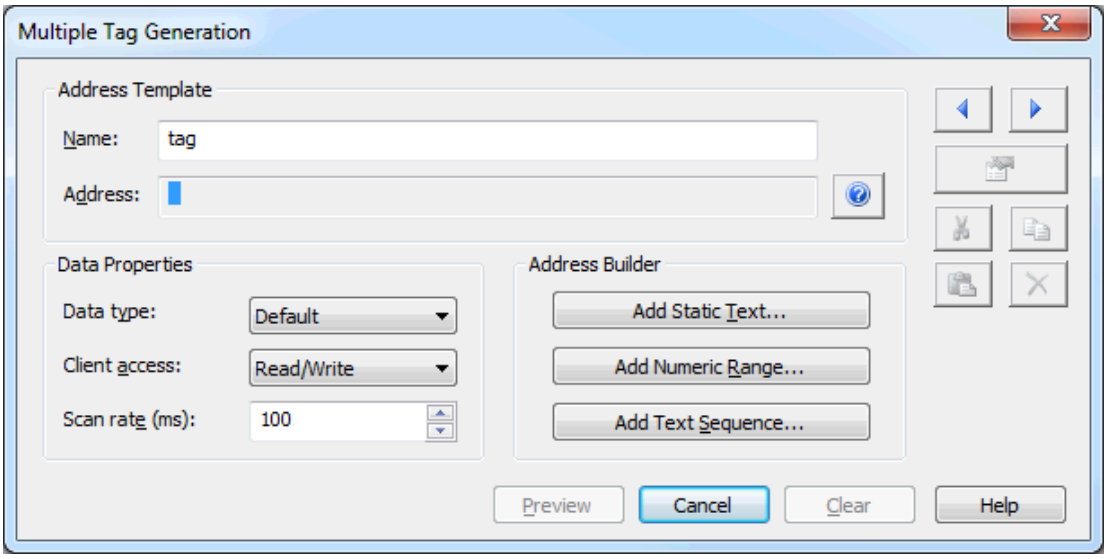
Generating Multiple Tags

The Multiple Tag Generation Tool dynamically creates tags using user-defined driver nomenclature. For information on using the tool, refer to the instructions below.

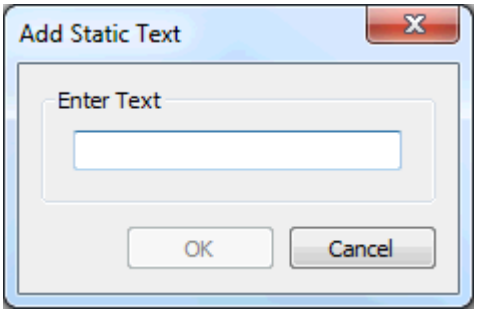
For more information on its properties, refer to Multiple Tag Generation.

1. To start, select a device and click **Edit | Connectivity | New Tag**. Alternatively, right-click on a device and select **New Tag**.
2. In **Tag Properties**, select the **Multiple Tag Generation** icon (located to the bottom-right of the Identification properties).

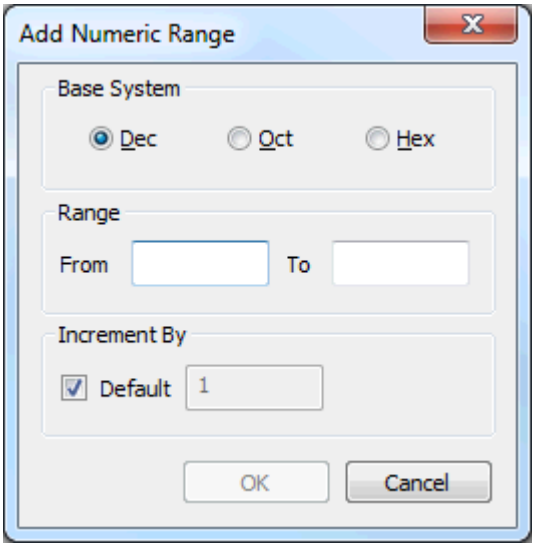
3. In **Multiple Tag Generation**, define the tag name, then configure the **Data Properties** properties as desired.



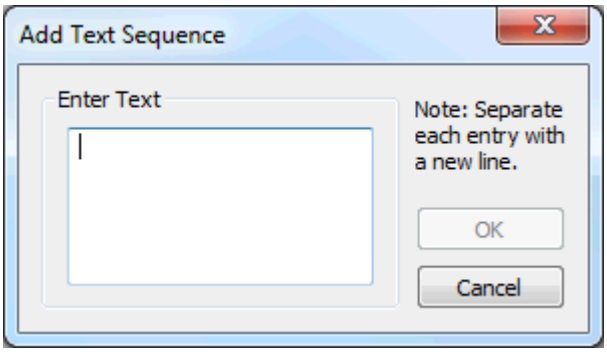
4. Click **Add Static Text**. In this group, enter the text as desired. Once finished, press **OK**.



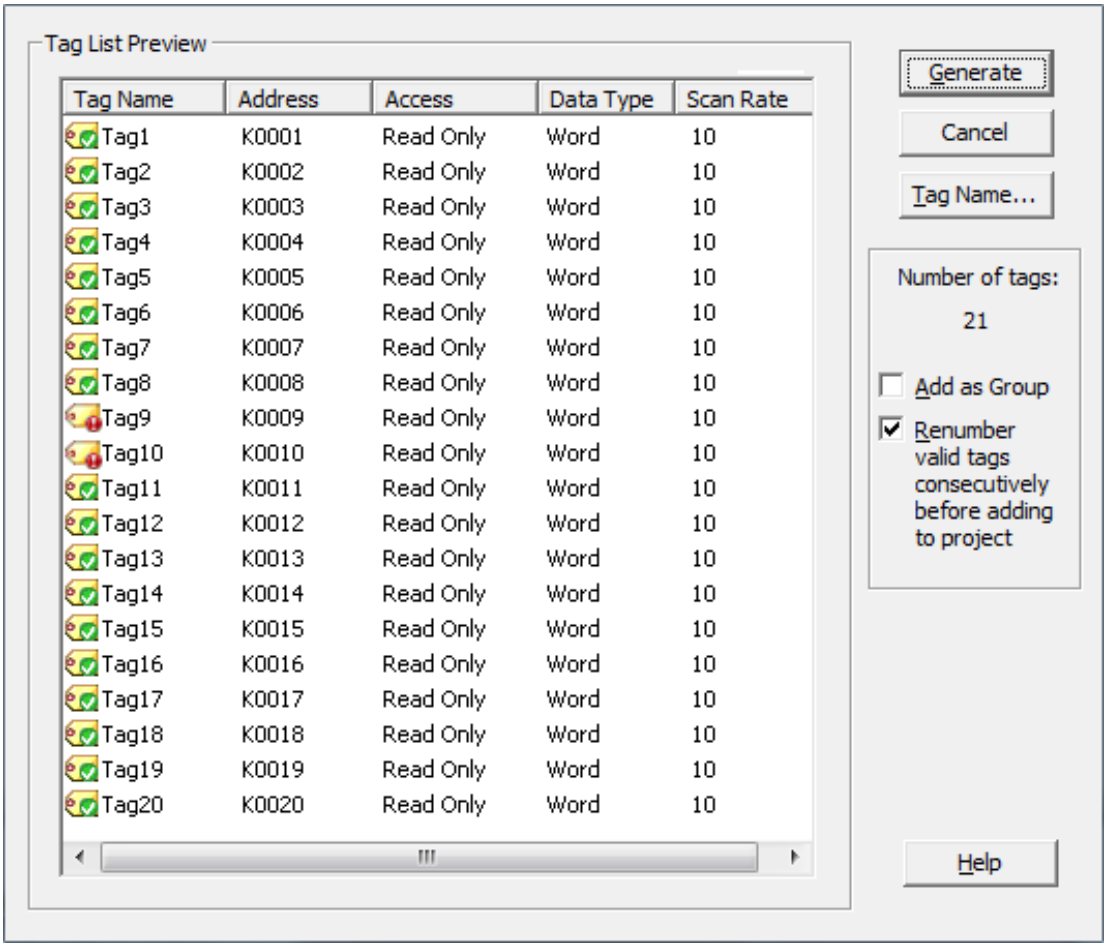
5. Click **Add Numeric Range**. In this group, enter the base system, range, and increment. Once finished, press **OK**.



6. Click **Add Text Sequence**. In this group, enter the text as desired. Separate each entry with a new line. Once finished, press **OK**.

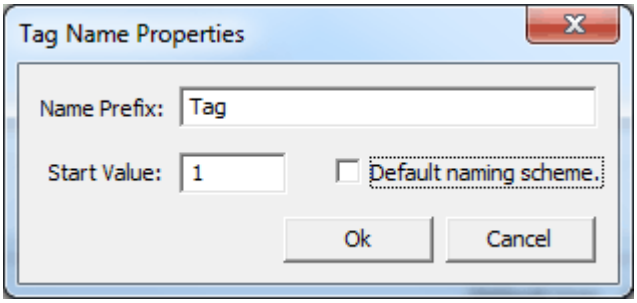


7. Click **Preview**.



Note: Valid tags are displayed with a green checkmark. Invalid tags are displayed with a red *x*.

8. To add the tags as a group, use **Add as Group**.
9. To change a tag's name or starting value, select **Tag Name**. Once finished, click **OK**.



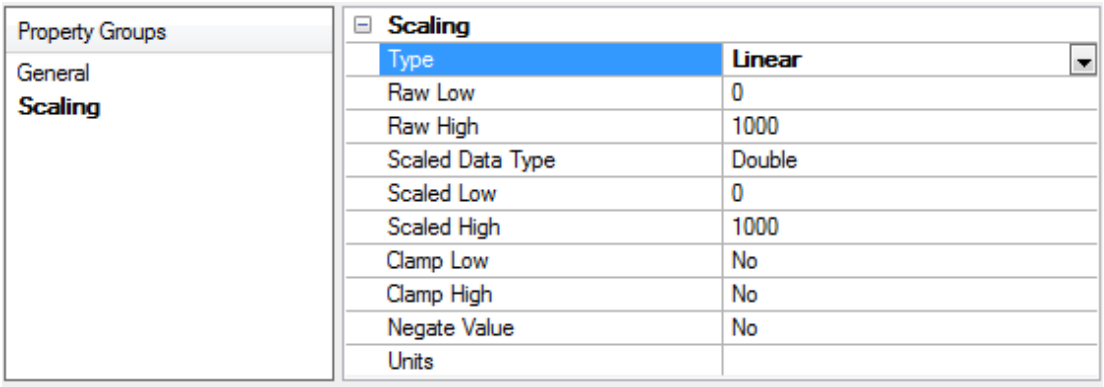
10. To generate the tags, click **Generate**. If the generation is successful, users return to the Multiple Tag Generation dialog.
11. Click **Close**. Then, click **OK**. The generated tags should be visible in the tag display window.

See Also: *Multiple Tag Generation*

Adding Tag Scaling

Users have the option of applying tag scaling when creating a new tag in the server. This allows raw data from the device to be scaled to an appropriate range for the application. There are two types of scaling: Linear and Square Root. For more information, refer to **Tag Properties — Scaling**.

1. To start, open the tag's **Tag Properties**.
2. Open the **Scaling** group.
3. For Type, select **Linear** or **Square Root**.
4. Specify the expected data range from the device with the high and low values and clamps. The scaled data type also allows users to specify how the resulting scaled value is presented to the OPC client application.



5. In **Units**, specify a string to the OPC client that describes the format or unit for the resulting engineering value. To use the Units field, an OPC client that can access the Data Access 2.0 tag properties data is required. If the client does not support these features, there is no need to configure this field.
6. Once the data has been entered as shown above, click **OK**.

Saving the Project

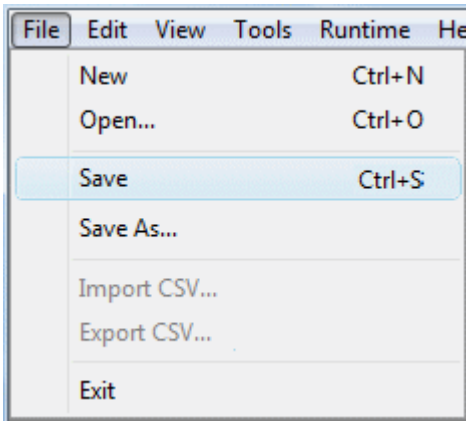
There should be a project configured with user-defined tags ready to be saved. How the project is saved depends on whether the project is a Runtime project or an offline project.

When editing a Runtime project, the server's online full-time operation allows immediate access to tags from a client once it has been saved to disk. Because the changes are made to the actual project, users can save by clicking **File | Save**.

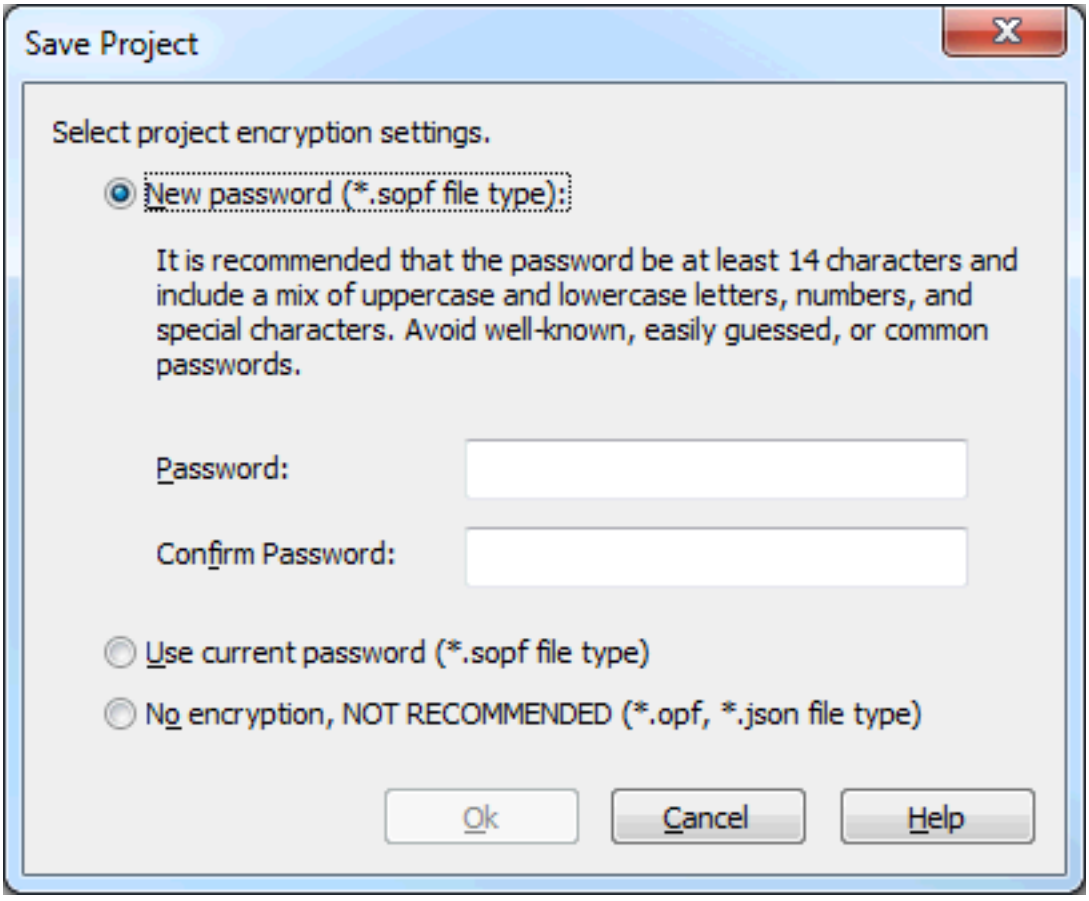
There are several valid file formats for project files: .OPF, .SOPF, and .JSON. The .OPF format is a binary project file format that is not encrypted. The .JSON (JavaScript Object Notation) format, while convenient, is human readable and text based, making it a less secure option to be used only where other security measures are in place. It is recommended that users save projects as .SOPF files as this file type is encrypted and the most secure way to save project files.

Users can overwrite the existing project or save edits as a new project, and are also given the option of loading the new project as the default Runtime project.

Open a saved project by choosing **File | Open** to locate and select the project file.



When editing an offline project, users have the option to save to the same project or to save as a new project. Once completed, click **Runtime | Connect** and load the new project as the default Runtime project.



When saving a new project with project file encryption enabled (on by default), a password must be set. Enter a password or select **No encryption** (not recommended) and click **Save**. The password can be modified and project encryption can be turned on or off under **Project Properties | General | Project File Encryption**. Click **Cancel** to abort without saving the project.

Tip: It is recommended that the password be at least 14 characters and include a mix of uppercase and lowercase letters, numbers, and special characters. Avoid well-known, easily guessed, or common passwords.

Projects that are saved as encrypted files with a password are saved as .SOPF files. The .JSON and .OPF files are not supported options for encrypted projects.

Note: An OPC client application can automatically invoke an OPC server when the client needs data. The OPC server, however, needs to know what project to run when it is called on in this fashion. The server loads the most recent project that has been loaded or configured. To determine what project the server will load, look to the **Most Recently Used** file list found in **File**. The loaded project is the first project file listed.

Project files are saved into the following directories by default.

For 64-bit OS versions, project files are saved (by default) in the directory:
C:\Users\<username>\Documents\Kepware\KEPServerEX\V6

For 32-bit OS versions, project files are saved (by default) in the directory:
C:\Users\<username>\Documents\Kepware\KEPServerEX\V6

The server automatically saves copies of the project in the following directory:

For 64-bit OS versions, project files are saved (by default) in the directory:
C:\ProgramData\Kepware\KEPServerEX\V6

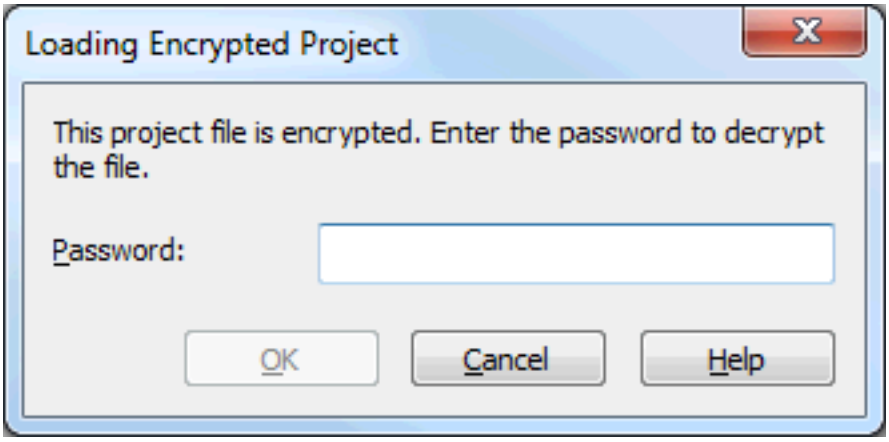
For 32-bit OS versions, project files are saved (by default) in the directory:
C:\ProgramData(x86)\Kepware\KEPServerEX\V6

Tip: If the file has been saved to an alternate location; search for *.opf, *.sopf, or *.json to locate available project files.

***See Also**: If security is of interest in the environment, consult the Secure KEPServerEX® Deployment guide.*

Opening an Encrypted Project

When opening a project file that has been saved with project file encryption enabled, the user is prompted to enter the password.

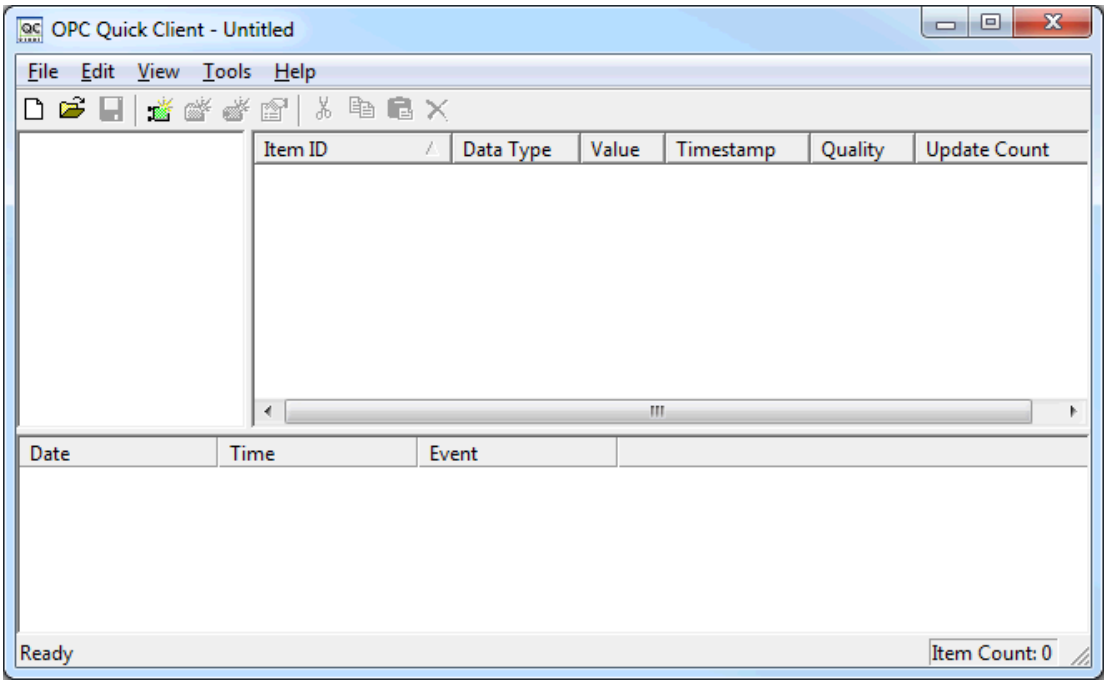


Enter the password used to encrypt the project file and click **OK** (or click **Cancel** to abort the file open operation).

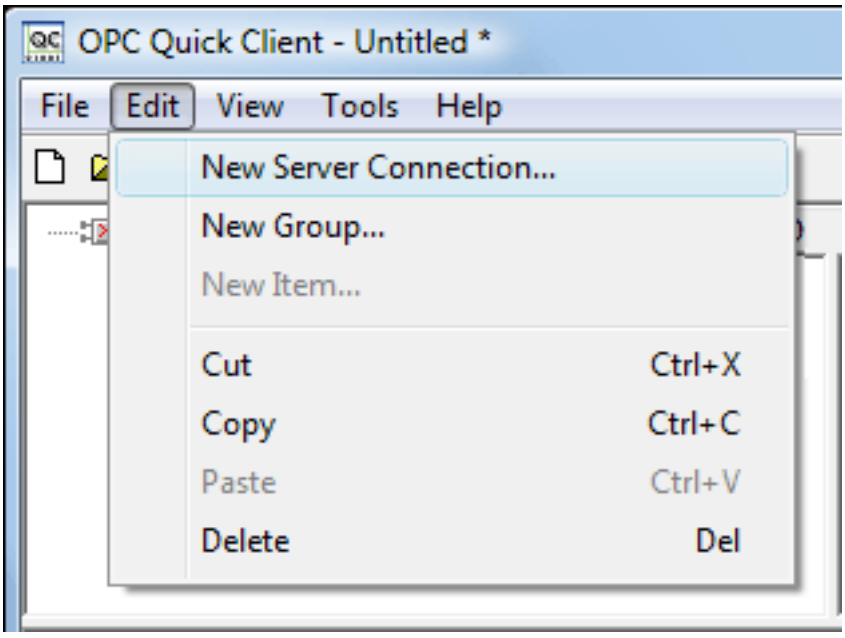
Testing the Project

The server includes a full-featured OPC Quick Client that supports all of the operations available in any OPC client application. The Quick Client can access all of the data available in the server application, and is used to read and write data, perform structured test suites, and test server performance. It also provides detailed feedback regarding any OPC errors returned by the server.

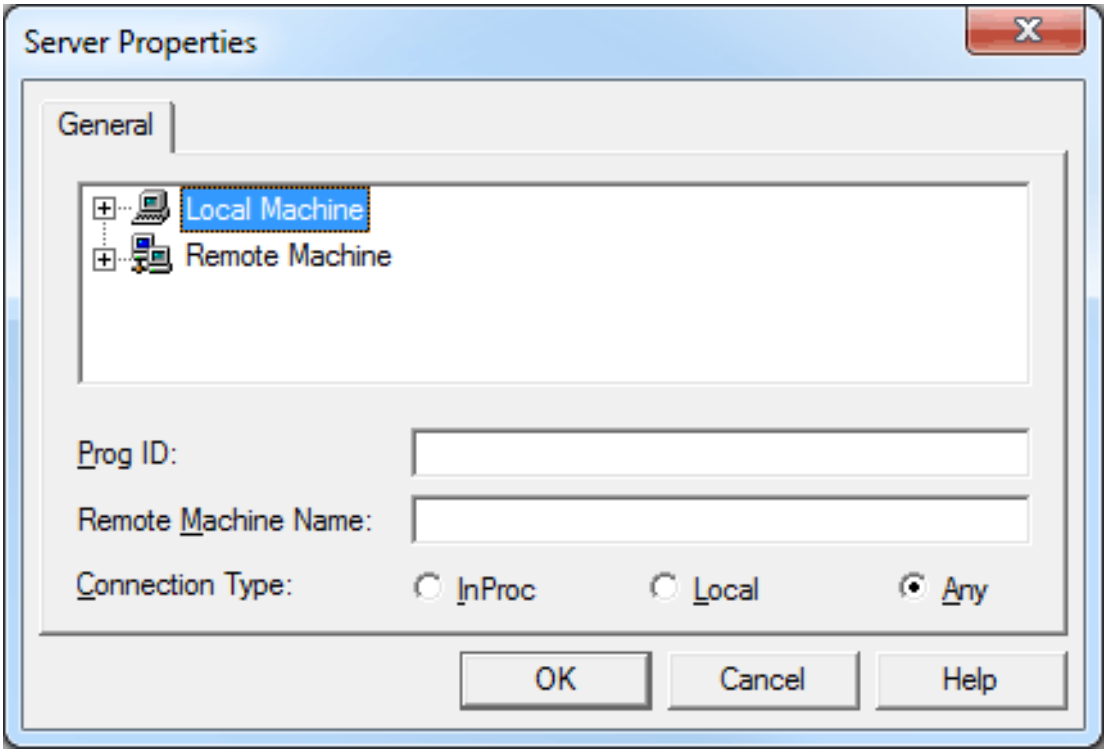
1. To start, locate the OPC Quick Client program in the same program group as the server. Then, run the OPC Quick Client.



2. Establish a connection by clicking **Edit | New Server Connection**.

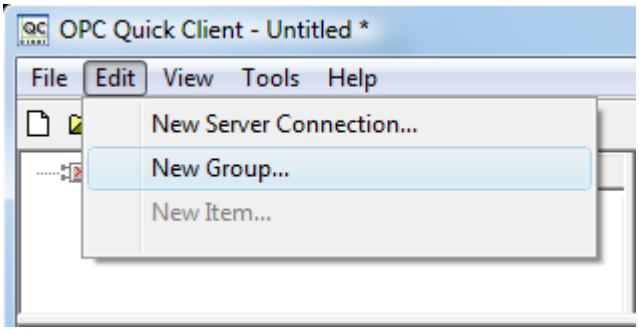


3. In **Server Properties**, make connections with an OPC server either locally or remotely via DCOM. By default, this dialog is pre-configured with the server's Prog ID (which is used by OPC clients to reference a specific OPC server).



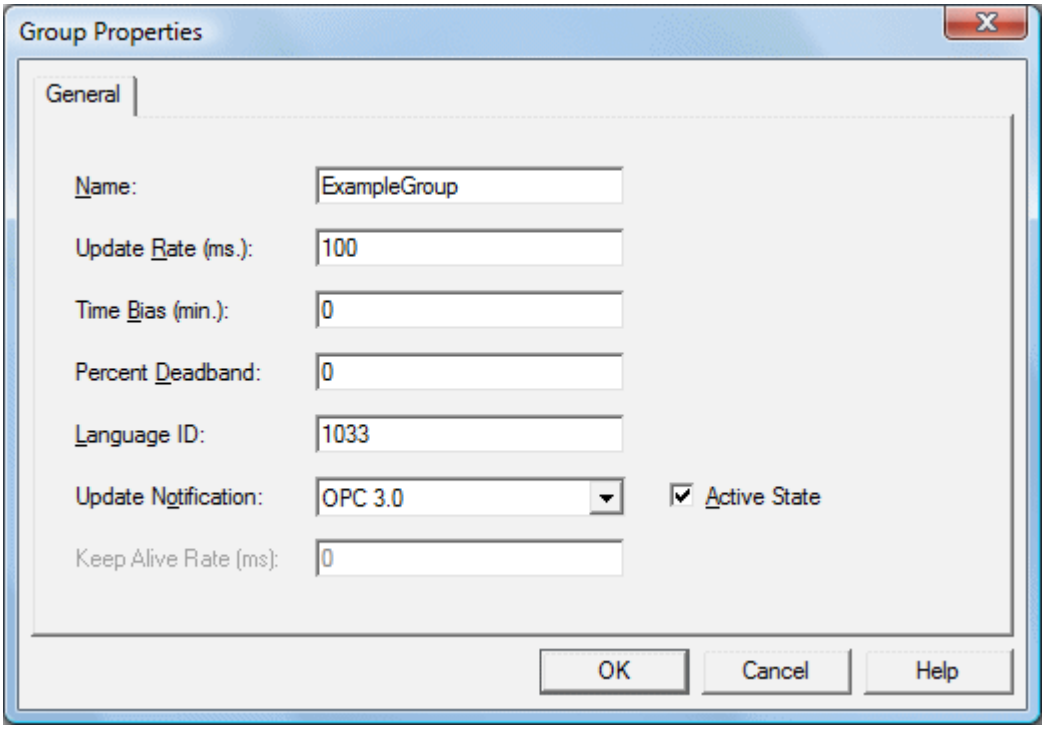
Note: Once a connection is made, two things may happen. If the server is running, the OPC Quick Client makes a connection to the server. If the server is not running, it starts automatically.

4. Add a group to the connection. To do so, select the server connection and click **Edit | New Group**.



Note: Groups act as a container for any tags accessed from the server and provide control over how tags are updated. All OPC clients use groups to access OPC server data. A number of properties are attached to a group that allow the OPC client to determine how often the data should be read from the tags, whether the tags are active or inactive, whether a dead band applies, and so forth. These properties let the OPC client control how the OPC server operates. For more information on group properties, refer to the OPC Quick Client help documentation.

5. For the purpose of this example, edit the group properties to match the following image.



Note: The Update Rate, Percent Dead Band, and Active State properties control when and if data is returned for the group's tags. Descriptions of the properties are as follows:

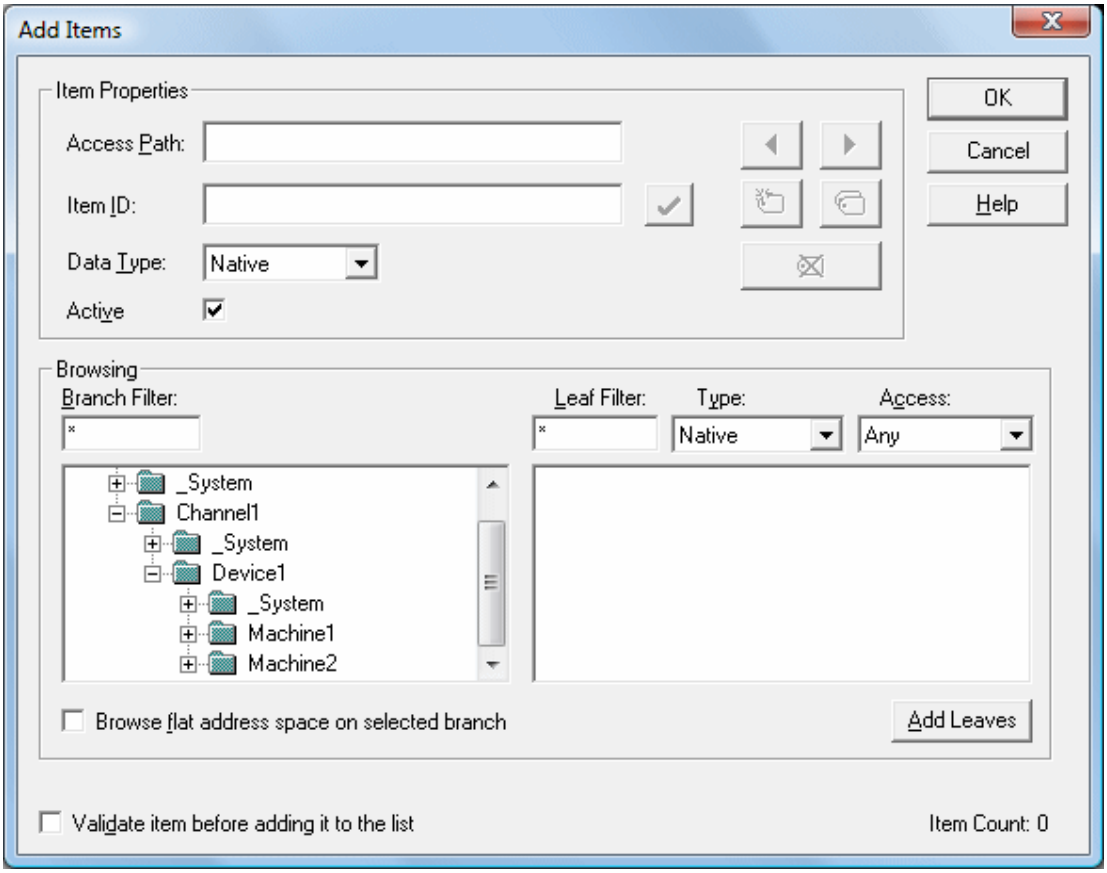
- **Name:** This property is used for reference from the client and can actually be left blank.
- **Update Rate**: icon to open how often data is scanned from the actual device and how often data is returned to the OPC client as a result of that scan.
- **Percent Dead Band**: This property eliminates or reduces noise content in the data by only detecting changes when they exceed the percentage change that has been requested. The percent change is a factor of the data type of a given tag.
- **Active State**: This property turns all of the tags in this group either on or off.

6. Once complete, click **OK**.

Accessing Tags

OPC server tags must be added to the group before they can be accessed. OPC data access specifications define a tag browsing interface as one 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, click on the desired tags to automatically add them to a group.

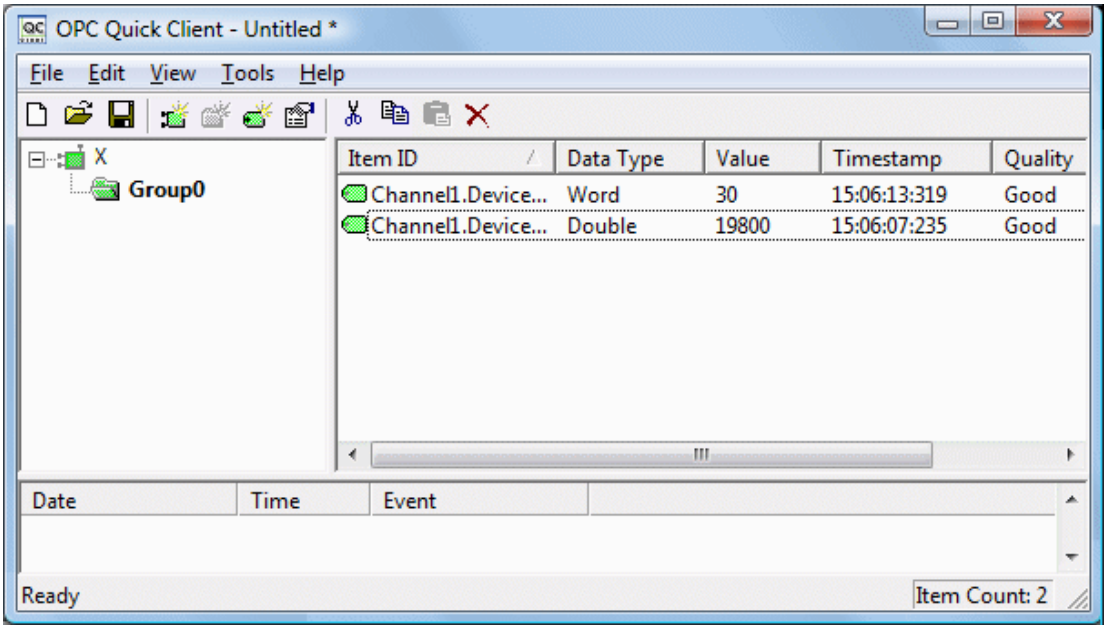
1. To start, select the group in which tags will be placed. Click **Edit | New Item**.



Note: The Add Items dialog also provides a tree view of the Browsing section and can be used to browse into an OPC server to find tags configured at the server. When using the "Example1" project, users can access the tags previously defined by expanding the branches of the view.

2. Once the tree hierarchy is at the point shown in the image above, users can begin adding tags to the OPC group by double-clicking on the tag name. As tags are added to the group, the **Item Count** shown at the bottom of the Add Items dialog increases to indicate the number of items being added. If both "MyFirstTag" and "MySecondTag" were added, the item count should be 2.
3. Once complete, click **OK**.

Note: Users should now be able to access data from the server using the two tags that were defined.

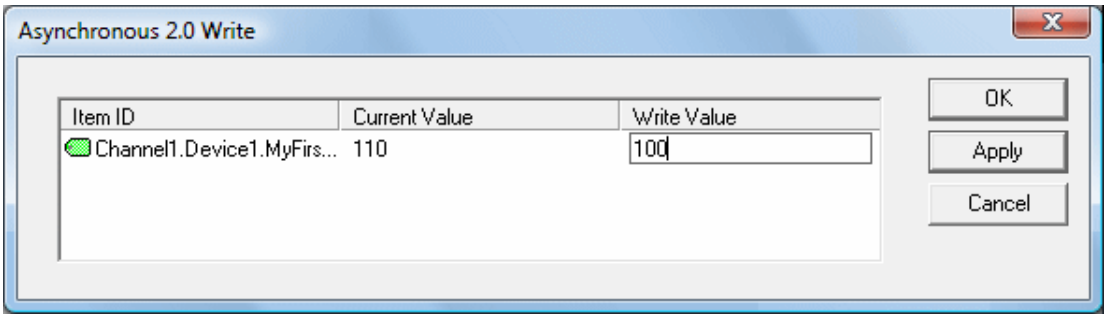


Note: The first tag, "MyFirstTag," should contain a changing value. The second tag should be zero at this point. If users only needed to test the reading of an OPC item, they are now finished. If, however, users desired to change an OPC item, they can use one of the write methods to send new data to the OPC item.

Writing Data to the OPC Server

The OPC Quick Client supports two methods for writing data to an OPC server: Synchronous Writes and Asynchronous Writes. Synchronous writes perform a write operation on the OPC server and wait for it to complete. Asynchronous writes perform a write on the OPC server but do not wait for the write to complete. Either method can be chosen when writing data to an OPC item: the different write methods are more of a factor in OPC client application design.

1. To start, first select the item. Then, right-click and select **Synchronous** or **Asynchronous Writes**. For the purpose of this example, right-click on "MyFirstTag" and select **Asynchronous Write**.



Note: Although the **Asynchronous 2.0 Write** dialog is displayed, the value continues to update.

2. To enter a new value for this item, click **Write Value** and enter a different value.
3. Click **Apply** to write the data. This allows users to continue writing new values, whereas clicking **OK** writes the new value and closes the dialog.
4. Click **OK**.

Note: If no new data has been entered, clicking **OK** does 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. Users are encouraged to continue testing various features of the server and the OPC Quick Client for greater understanding and comprehension. For more information on the OPC Quick Client, refer to its help documentation.

Users can now begin developing the OPC application. If using Visual Basic, refer to 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.

How Do I...

For more information, select a link from the list below.

Allow Desktop Interactions
Create and Use an Alias
Optimize the Server Project
Process Array Data
Properly Name a Channel, Device, Tag, and Tag Group
Resolve Comm Issues When the DNS/DHCP Device Connected to the Server is Power Cycled
Select the Correct Network Cable
Use an Alias to Optimize a Project
Use DDE with the Server
Use Dynamic Tag Addressing
Use Ethernet Encapsulation
Work with Non-Normalized Floating Point Values

Allow Desktop Interactions

Some communication interfaces require the server to interact with the desktop. For example, Windows Messaging Layer is used by DDE and FastDDE. It is important that the operating system be taken into consideration when choosing how to communicate with the desktop.

Windows Vista, Windows Server 2008, and Later Operating Systems

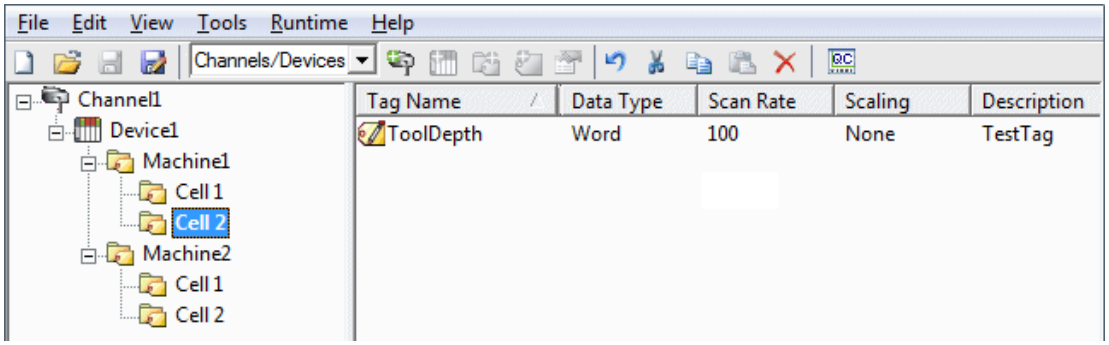
In Windows Vista, Windows Server 2008, and later operating systems, services run in an isolated session that is inaccessible to users logged on to the console. These operating systems require that the process mode be set to Interactive. This allows the Runtime to run in the same user account as the current user. *For information on changing the process mode, refer to* ***Settings - Runtime Process***.

See Also:
Accessing the Administration Menu

Create and Use an Alias

Complex Tag Reference Example

The image below displays a Complex tag reference in the server.



For example, to create a DDE link to an application for the "ToolDepth" tag, the DDE link must be entered as "<DDE service name>|_ddedata!Channel1.Device1.Machine1.Cell2.ToolDepth".

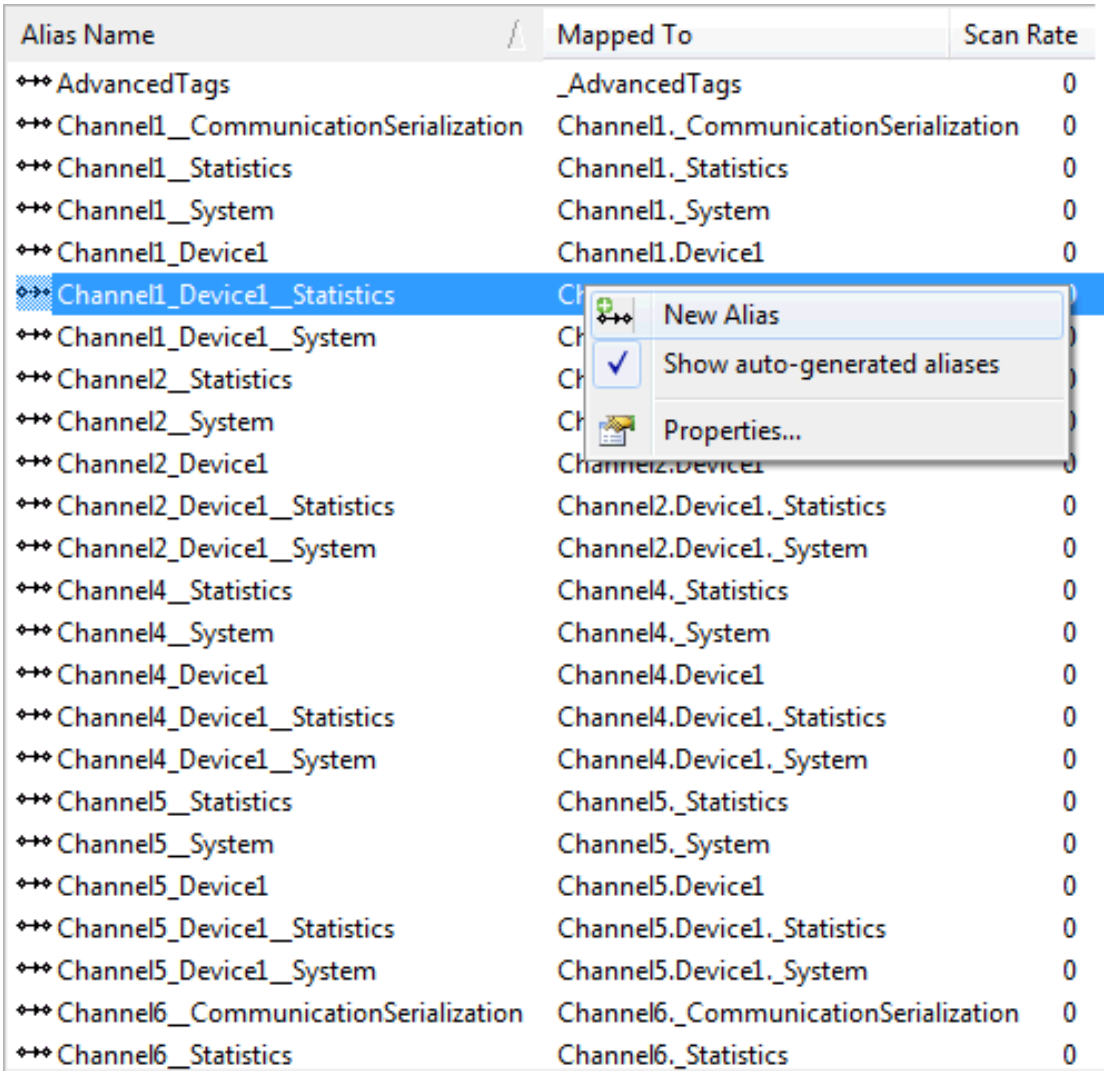
Although the DDE link's *<application>|<topic>!<item>* format still exists, the content becomes more complex when optional tag groups and the channel name are required as part of the topic. The alias map allows a shorter version of the reference to be used in DDE client applications.

For more information, refer to ***What is the Alias Map****.*

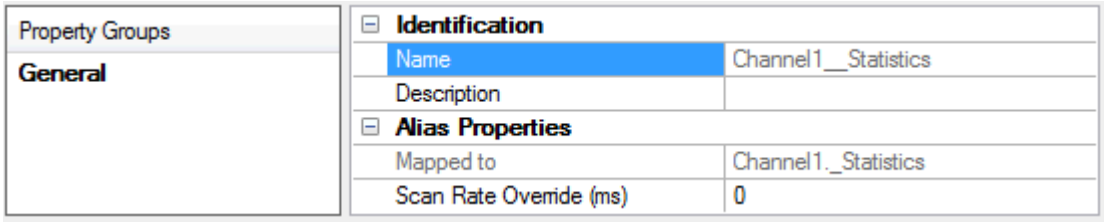
Creating Aliases for Complex Address Paths

For information on creating aliases to simplify complex tag address paths, follow the instructions below.

1. In the tree view, select the alias to edit and double-click to open the alias node.
2. In the detail view, right-click and select **New Alias** (OR choose **Edit | Aliases | New Alias**).



3. Browse to the group or device that contains the item to be referenced.



4. Enter an alias name to represent the complex tag reference. This alias name can now be used in the client application to address the tag found in the server. *For information on reserved characters, refer to* ***How To... Properly Name a Channel, Device, Tag, and Tag Group***.

5. The complex topic and item name "_ddedata! Channel1.Device1.Machine1.Cell2" can be replaced by using the alias "Mac1Cell2". When applied to the example above, the DDE link in the application can be entered as "<DDE service name>|Mac1Cell2!ToolDepth".

Note: Although possible, it is not recommended that users create an alias that shares a name with a channel. The client's item fails if it references a dynamic address using the shared name. For example, if an alias is named "Channel1" and is mapped to "Channel1.Device1," an item in the client that references "Chan-

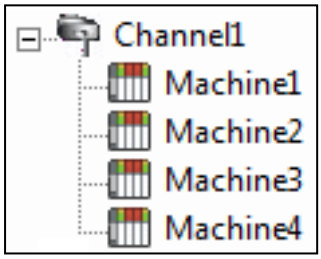
nel1.Device1.<address>" is invalid. The alias must be removed or renamed so that the client's reference can succeed.

See Also: *Alias Properties*

Optimize a Server Project

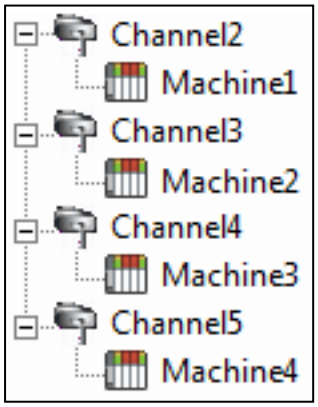
Nearly every driver of this server supports at least 100 channels; meaning, 100 COM/serial ports or 100 source sockets for Ethernet communications. To determine the number of supported channels available for each device, refer to the Driver Information under **Server Summary Information**.

This server refers to communications protocols as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must be defined under that channel. Each of these devices represents a single device from which data is collected. While this approach to defining the application provides a high level of performance, it won't take full advantage of the driver or the network. An example of how the application may appear when configured using a single channel is shown below.



Each device appears under a single channel. In this configuration, the driver must move from one device to the next as quickly as possible to gather information at an effective rate. As more devices are added or more information is requested from a single device, the overall update rate begins to suffer.

If the driver could only define one single channel, the example shown above would be the only option available. Using multiple channels distributes, however, the data collection workload by simultaneously issuing multiple requests to the network. An example of how the same application may appear when configured using multiple channels to improve performance is shown below.



Each device has now been defined under its own channel. In this new configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has fewer devices, it can be optimized exactly how it is shown here.

The performance improves even if the application has more devices than channels. While 1 device per channel is ideal, the application benefits from additional channels. Although by spreading the device load across all channels causes the server to move from device to device again, it does so with far fewer devices to process on a single channel.

This same process can be used to make multiple connections to one Ethernet device. Although the OPC server may allow 100 channels for most drivers, the device ultimately determines the number of allowed connections. This constraint comes from the fact that most devices limit the number of supported connections. The more connections that are made to a device, the less time it has to process request on each connect. This means that there can be an inverse tradeoff in performance as connections are added.

Process Array Data

Many of the drivers available for this server allow clients to access data in an array format. Arrays allow the client application to request a specific set of contiguous data in one request. Arrays are one specific data type; users would not have an array with a combination of Word and DWord data types. Furthermore, arrays are written to in one transaction. To use arrays in the server, the client application must support the ability to at least read array data.

Processing Array Data In a DDE Client

Array data is only available to the client when using CF_TEXT or Advanced DDE clipboard formats.

For client applications using Advanced DDE, the number of elements in the array is specified in the SPACKDDE_DATAHDR_TAG structure. 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 attempting to poke data in an array format to a tag that was not declared as an array, only the first value in the array is written. If attempting to poke more data than the tag's array size, only as much data as the tag's array size is written. If attempting to poke data while leaving some data values blank, the server uses the last known value for that array element when writing back to the device. If the value in that register has been changed but has not been updated in the server, it is overwritten with the old value. For this reason, it is best to be cautious when writing data to arrays.

Processing Array Data In an OPC Client

In OPC clients that support arrays, the OPC item data value is actually a variant array data type. The OPC client parses the array element data: some clients create sub tags for display purposes. For example, if the OPC client created a tag in its database named 'Process,' and the associated OPC item was a single dimensional array of 5 elements, it may create 5 tags named 'Process_1', 'Process2,' and so forth. Other clients (such as the OPC Quick Client) may display the data as Comma Separated Values (CSV).

Properly Name a Channel, Device, Tag, and Tag Group

When naming a channel, device, tag, or tag group, the following characters are reserved or restricted:

- Periods
- Double quotation marks
- Leading underscores
- Leading or trailing spaces

Note: Some of the restricted characters can be used in specific situations. For more information, refer to the list below.

1. Periods are used in aliases to separate the original channel name and the device name. For example, a valid name is "Channel1.Device1".
2. Underscores can be used after the first character. For example, a valid name is "Tag_1".
3. Spaces may be used within the name. For example, a valid name is "Tag 1".

Resolve Comm Issues When the DNS/DHCP Device Connected to the Server is Power Cycled

Certain drivers support DNS/DHCP resolution for connectivity, which allows users to assign unique domain / network names for identification purposes. When starting and connecting to the network, the devices request an IP address from the network DNS server. This process of resolving a domain name to an IP address for connectivity takes time. For greater speed, the operating system caches all of the resolved IP / domain names and re-uses them. The resolved names are held in cache for two hours by default.

- The server fails to reconnect to a device when the name of the IP address associated with the device's domain / network changes. If this change is a result of the device being power cycled, it acquires a new IP. This change may also be a result of the IP being manually changed on the device. In both cases, the IP address that was being used no longer exists.

Because the server automatically flushes the cache every 30 seconds, the IP is forced to resolve. If this does not correct the issue, users can manually flush the cache by typing the command string "ipconfig / flushdns" in the PC's command prompt.

- *For more information, refer to the following Microsoft Support article **Disabling and Modifying Client Side DNS Caching**.*

Select the Correct Network Cable

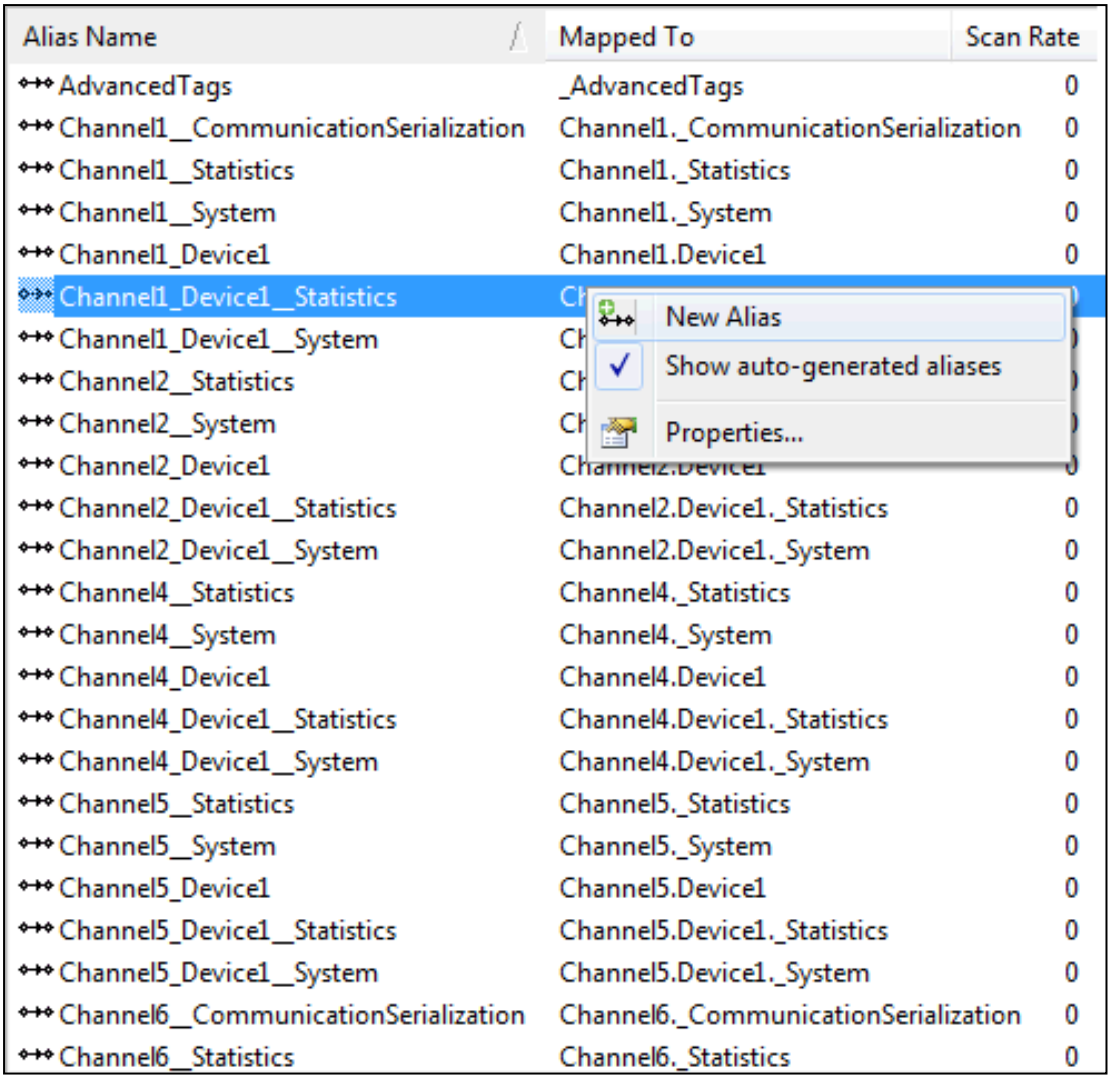
Without prior experience of Ethernet enabled devices or serial to Ethernet converters, users may find selecting the correct network cable a confusing task. There are generally two ways to determine the proper cable setup. If connecting to the device or converter through a network hub or switch, users need **Patch Cable**. A Patch Cable gets its name from the days when a telephone operator-style board was used to patch or connect devices to each other. If connecting directly to the device from the PC, however, users need a **Crossover Cable**. Both of these cables can be purchased from an electronic or PC supply store.

Use an Alias to Optimize a Project

To get the best performance out of a project, it is recommended that each device be placed on its own channel. If a project needs to be optimized for communication after it has been created, it can be difficult to change the client application to reference the new item names. By using an alias map, however, users can allow the client to make the legacy request to the new Configuration. To start, follow the instructions below.

1. To start, create a new channel for each device. Place the device under the new channel and delete the original channel.
2. Under Alias in the tree view, create a **New Alias** for each device in the **Alias Map**. The alias name is the original channel and device name separated by a period. For example, "Channel1.Device1".

- *For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*



Note: The server validates any request for items against the alias map before responding back to the client application with an error that the item does not exist.

Use DDE with the Server

Using DDE in an Application

Dynamic Data Exchange (DDE) 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 the application (service) name, topic name, and item name.

For DDE clients to connect to the server interface, the runtime must be allowed to interact with the desktop.

For more information, refer to ***How to Allow Desktop Interactions***.

Example 1: Accessing a Register Locally (Using the Default Topic)

The syntax is <*application*>|<*topic*>*!*<*item*> where:

- **application** DDE service name
- **topic** _ddedata*

- **item** Modbus.PLC1.40001

*This is the default topic for all DDE data that does not use an alias map entry.

Note: An example of the syntax is "MyDDE|_ddedata!Modbus.PLC1.40001".

Example 2: Accessing a Register Locally (Using an Alias Name as a Topic)

The syntax is <*application*>|<*topic*>!<*item*> where:

- **application** DDE service name
- **topic** ModPLC1*
- **item** 40001

*This is the topic using the alias map entry.

Note: An example of the syntax is "MyDDE|ModPLC1!40001" . For additional possible syntax, refer to the DDE client's specific help documentation.

See Also:

Project Properties — DDE
Project Properties — FastDDE & SuiteLink
What is the Alias Map?

Use Dynamic Tag Addressing

This server can also be used to dynamically reference a physical device data address from the server. The server dynamically creates a tag for the requested item. Users cannot browse for tags from one client that were dynamically added by another. Before adding tags dynamically, users should note the following:

- The correct syntax must be used for the data address. For more information on the specific driver's syntax, refer to its help documentation.
- If users do not specify the requested item's data type, it is set to the default setting by the application. For more information on the specific driver's supported data types, refer to its help documentation.

Note: In the examples below, the Simulator Driver is used with a channel name of 'Channel1' and a device name of 'Device1'.

Example 1: Using Dynamic Tag Addressing In a Non-OPC Client

To get data from register 'K0001' in the simulated device, use an item ID of "Channel1.Device1.K001." The default data type for this register is Short. Since non-OPC clients do not provide an update rate to the server, the Dynamic tag's default update rate is 100 ms. Both data type and update rate can be overridden after the dynamic request is sent.

To override the tag defaults, use the commercial AT sign ('@') at the end of the item. If intending to add the register as a DWord (unsigned 32-bit) data type, use an item ID of "Channel1.Device1.K0001@DWord." To change the default update rate to 1000 ms, use "Channel1.Device1.K0001@1000." To change both defaults, use "Channel1.Device1.K0001@DWord,1000."

Note: The client application must be able to accept special characters like the '@' in its address space.

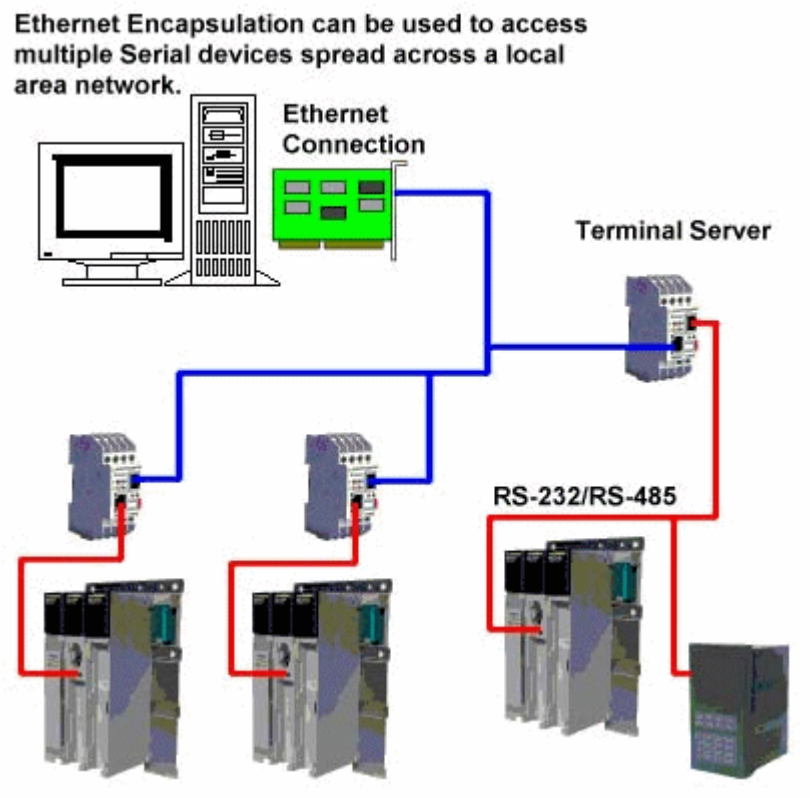
Example 2: Using Dynamic Tag Addressing In an OPC Client

In an OPC client, the same syntax can be used to override the data type if the client application does not provide a way to specify a data type when the OPC item is added. Since the item's update rate is not used in OPC, there is no need to override it.

Note: The client application must be able to accept special characters like the '@' in its address space.

Use Ethernet Encapsulation

Ethernet Encapsulation mode is designed to provide communications with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port that converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted to serial form, users can connect standard devices that support serial communications to the terminal server. The diagram below shows how to employ Ethernet Encapsulation mode.



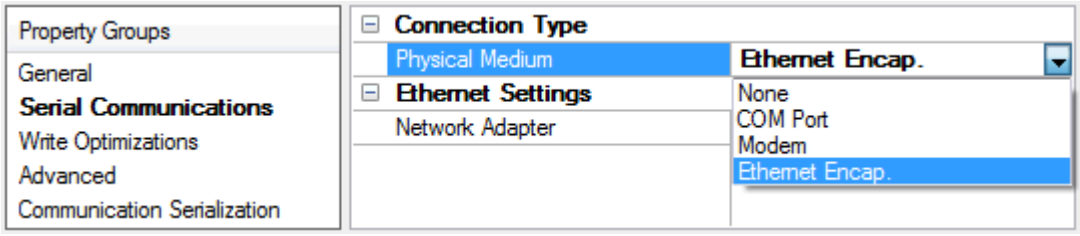
Note: For unsolicited drivers that support Ethernet Encapsulation, users must configure the port and the protocol settings at the channel level. This allows the driver to bind to the specified port and process incoming requests from multiple devices. An IP address is not entered at the channel because the channel accepts incoming requests from all devices.

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

Configuring Ethernet Encapsulation Mode

To enable Ethernet Encapsulation mode, open **Channel Properties** and select the **Serial Communications** group. In the **Connection Type** drop-down menu, select **Ethernet Encap**.

Note: Only the drivers that support Ethernet Encapsulation allows the option to be selected.



Note: The server's multiple channel support allows up to 16 channels on each driver protocol. This allows users to specify one channel to use the local PC serial port and another channel to use Ethernet Encapsulation mode.

When Ethernet Encapsulation mode is selected, the serial port settings (such as baud rate, data bits, and parity) are unavailable. After the channel has been configured for Ethernet Encapsulation mode, users must configure the device for Ethernet operation. When a new device is added to the channel, the Ethernet Encapsulation settings can be used to select an Ethernet IP address, an Ethernet Port number, and the Ethernet protocol.

Note: The terminal server being used must have its serial port configured to match the requirements of the serial device to be attached to the terminal server.

Work with Non-Normalized Floating Point Values

A non-normalized floating point value is defined as Infinity, Not-a-Number (NaN), or as a Denormalized Number. For more information, refer to the table below.

Term	Definition
Non-Normalized Floating Point Value	An IEEE-754 floating point number that is one of the following: • Negative Infinity to Quiet Negative NaN. • Positive Infinity to Quiet Positive NaN. • Negative Denormalized Values. • Positive Denormalized Values.
NaN	A number that exists outside of the range that may be represented as floating points. There are two types of NaN representations: Quiet and Signaling.*
Denormalized Number	A non-zero floating point number whose magnitude is less than the magnitude of the smallest IEEE 754-2008 value that may be represented for a Float or a Double. • For Floats, these include numbers between -1.175494E-38 and -1.401298E-45 (Negative Denormalized) and 1.401298E-45 and 1.175494E-38 (Positive Denormalized). • For Doubles, these include numbers between -2.225074E-308 and -4.940657E-324 (Negative Denormalized) and 4.940657E-324 and 2.225074E-308 (Positive Denormalized).

*A floating point value that falls within the Signaling NaN range is converted to a Quiet NaN before being transferred to a client for Float and Double data types. To avoid this conversion, use a single element floating-point array.

Handling Non-Normalized IEEE-754 Floating Point Values

Users can specify how a driver handles non-normalized IEEE-754 floating point values through the "Non-Normalized Value Should Be" property located in **Channel Properties — Advanced**. When Unmodified is selected, all values are transferred to clients without any modifications. For example, a driver that reads a 32-bit float value of 0xFF800000(-Infinity) transfers that value "as is" to the client. When Replaced with Zero is selected, certain values are replaced with zero before being transferred to clients. For example, a driver that reads a 32-bit float value of 0xFF800000(-Infinity) are replaced with zero before being transferred to a client.

Note: For information on which values are replaced with zero before being transferred to clients, refer to the tables below.

IEEE-754 Range for 32-Bit Floating Point Values

Name	Hexadecimal Range	Decimal Range
Quiet -NaN	0xFFFFFFFF to 0xFFC00001	N/A
Quiet +NaN	0x7FC00000 to 7FFFFFFF	N/A
Indeterminate	0xFFC00000	N/A
Signaling -NaN	0xFFBFFFFF to 0xFF800001	N/A
Signaling +NaN	0x7F800001 to 7FBFFFFF	N/A
-Infinity (Negative Overflow)	0xFF800000	≤ -3.4028235677973365E+38
+Infinity (Positive Overflow)	0x7F800000	≥ 3.4028235677973365E+38
Negative Normalized -1.m × 2(e-127)	0xFF7FFFFF to 0x80800000	-3.4028234663852886E+38 to -1.1754943508222875E-38
Negative Denormalized -0.m × 2(-126)	0x807FFFFF to 0x80000001	-1.1754942106924411E-38 to -1.4012984643248170E-45(-7.0064923216240862E-46)
Positive Denormalized 0.m × 2(-126)	0x00000001 to 0x007FFFFF	(7.0064923216240862E-46) * 1.4012984643248170E-45 to 1.1754942106924411E-38
Positive Normalized 1.m × 2(e-127)	0x00800000 to 0x7F7FFFFF	1.1754943508222875E-38 to 3.4028234663852886E+38

IEEE-754 Range for 64-Bit Floating Point Values

Name	Hexadecimal Range	Decimal Range
Quiet -NaN	0xFFFFFFFFFFFFFFFF to 0xFFF8000000000001	N/A
Quiet +NaN	0x7FF8000000000000 to 0x7FFFFFFFFFFFFFFF	N/A
Indeterminate	0xFFF8000000000000	N/A
Signaling -NaN	0xFFF7FFFFFFFFFFFF to 0xFFF0000000000001	N/A
Signaling +NaN	0x7FF0000000000001 to 0x7FF7FFFFFFFFFFFF	N/A
-Infinity (Negative Overflow)	0xFFF0000000000000	≤ -1.7976931348623158E+308
+Infinity (Positive Over-flow)	0x7FF0000000000000	≥ 1.7976931348623158E+308
Negative Normalized -1.m × 2(e-1023)	0xFFEFFFFFFFFFFFFF to 0x8010000000000000	-1.7976931348623157E+308 to -2.2250738585072014E-308
Negative Denormalized -0.m × 2(-1022)	0x800FFFFFFFFFFFFF to 0x8000000000000001	-2.2250738585072010E-308 to -4.9406564584124654E-324 (-2.4703282292062328E-324)
Positive Denor-malized 0.m × 2(-1022)	0x0000000000000001 to 0x000FFFFFFFFFFFFF	(2.4703282292062328E-324) * 4.9406564584124654E-324 to 2.2250738585072010E-308
Positive Normalized 1.m × 2(e-1023)	0x0010000000000000 to 0x7FEFFFFFFFFFFFFF	2.2250738585072014E-308 to 1.7976931348623157E+308

Device Demand Poll

Device Demand Poll is useful for customers that require full control of polling devices from their client applications. It is particularly helpful in SCADA industries like Oil & Gas, Water/Waste Water, Electric, and others that may experience significant communication delays.

Many client-side SCADA systems either do not have configurable scan rates or have scan rates whose minimum value is too long for the data updates that are required by SCADA operators. To bypass this limitation, the SCADA system can perform writes to the Device Demand Poll tags available in the server. In this scenario, each device in the server exposes a _DemandPoll tag that polls all referenced tags on the device when written to by a client. During the poll, the _DemandPoll tag becomes True (1). It returns to False (0) when the final active tag signals that the read requests have completed. Subsequent writes to the _DemandPoll tag fails until the tag value returns to False. The demand poll respects the read / write duty cycle for the channel. Client-side SCADA scripts (such as a Refresh button script) can be developed to write to the _DemandPoll tag and cause a poll to occur. The poll results are passed on to the client application. For more information, refer to **System Tags**.

Note: The procedure described above is not OPC-compliant behavior. If this is a problem, it is recommended that communications be separated onto two devices. One device can use the classic OPC update interval, and the other device can set the Scan Mode to "Do not scan, demand poll only" and only poll when the _DemandPoll tag is written to.

Regardless of whether Device Demand Poll is being utilized, clients that are limited by tag scan rates may also encounter operator wait time due to the server complying with the OPC client's group update rate. To circumvent this OPC-compliant behavior, users can configure the "Ignore group update rate, return data as soon as it is available" setting. This returns the poll results immediately and disregards the update interval. For more information, refer to **Project Properties — OPC DA Compliance**.

See Also: *Device Properties — Scan Mode*

Configuring from iFIX Applications

For information on configuring process database blocks to reference IGS I/O addresses, select a link from the list below.

Overview: Creating Datablocks Inside iFIX Applications
Setting Options for IGS
Entering Driver Information in iFIX Database Manager
Specifying I/O Drivers in the Device Field
Specifying I/O Addresses in iFIX Database Manager
Specifying Signal Conditioning in iFIX Database Manager
I/O Signal Conditioning Options
Using Offset fields with Analog and Digital Registers (AR/DR)
Project Startup for iFIX Applications

Overview: Creating Datablocks Inside iFIX Applications

The IGS Driver Configuration program does not need to be used to create all of the IGS driver tags. With the correct information, users can add IGS driver tags while configuring the database in the iFIX Database Manager. To do so, the following information is required:

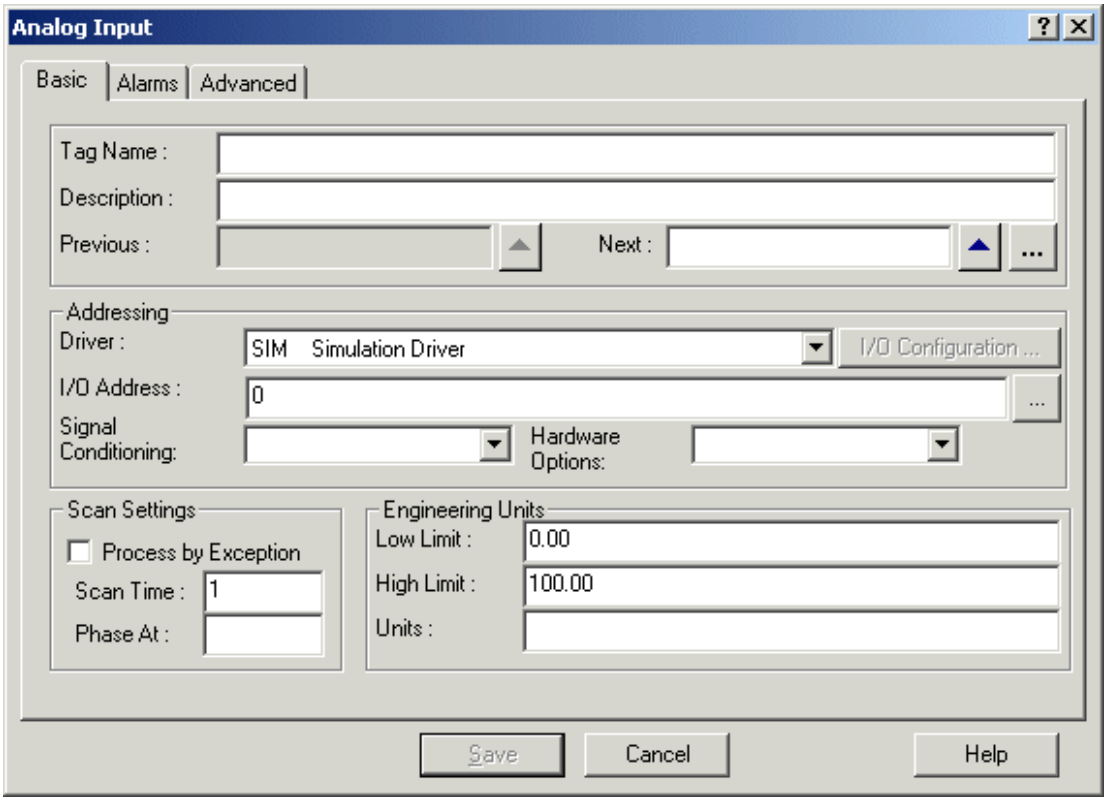
- The driver's three-letter acronym. For the IGS driver, the acronym is "IGS".
- The name of the channel, device, and tag from which data will be collected (as defined in the IGS Driver Configuration program).
- Any other information about the tag, such as the array element of the bit offset.

For more information on entering data in the Database Manager for automatic datablock creation, refer to *Entering Driver Information in iFIX Database Manager*.

Entering Driver Information in iFIX Database Manager

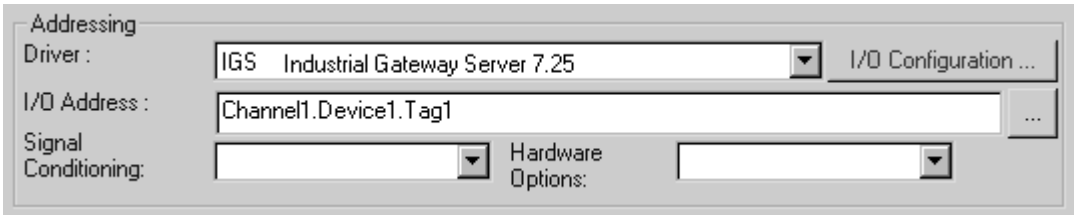
For information on entering driver specifications for a database block in the iFIX Database Manager, refer to the instructions below.

1. In the **iFIX Database Manager**, click **Blocks | Add**.
2. Select the type of block and click **OK**.



3. In **Tag Name**, specify a name for the database block. Then, enter driver-appropriate information in the remaining properties.

Note: This driver does not use the **Hardware Options** and **Signal Conditioning** fields.



See Also: For information on the valid entries required for each field, select a link from the list below.

Specifying the I/O Driver in iFIX Database Manager

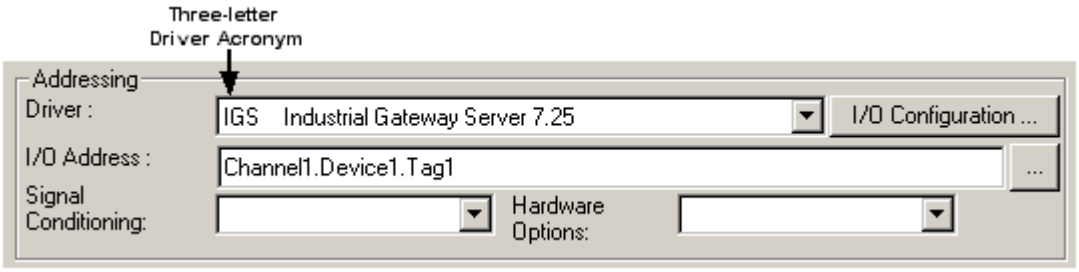
Specifying I/O Addresses in iFIX Database Manager

Specifying Signal Conditioning in iFIX Database Manager

Specifying the I/O Driver in iFIX Database Manager

To identify the I/O driver that the database block will access, locate the **Driver** property in the Database Manager. Then, specify the driver's three-letter acronym. To use the IGS driver, enter "IGS".

To find the default driver, open the **System Configuration Utility (SCU)** and click **SCADA Configuration**. The default driver is the first driver listed in the Configured I/O Driver list box.

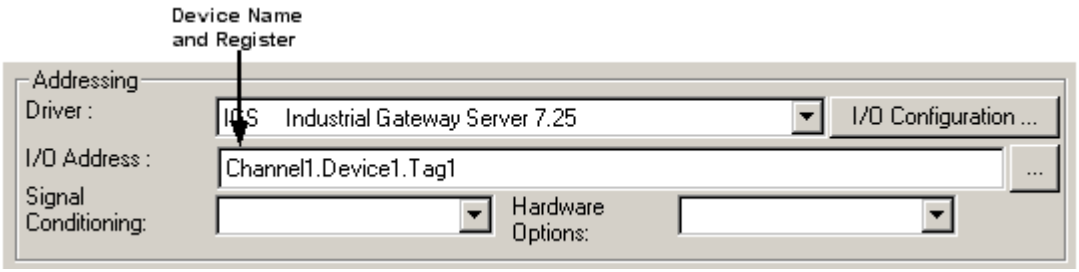


Note: For Database Manager to recognize the acronym entered, it must appear in the SCU's Configured I/O Driver list box.

Specifying I/O Addresses in iFIX Database Manager

To specify the datablock address to be accessed, locate the **I/O Address** property in the Database Manager. Then, enter the I/O address. This field is not case sensitive. For an IGS driver, I/O addresses typically consist of the name of the channel, device, and tag and are specific to the driver.

Note: Multiple blocks may use the same I/O address with the IGS server.



The I/O address for the driver has the following format: *Channel_Name.Device_Name.Tag_Name*

where:

- **Channel_Name** This is the name of the protocol or driver being used in the IGS server project. It must match the channel name in the IGS configuration.
- **Device_Name** This is the name of the PLC or other hardware with which the server communicates. It must match the device name for the specified channel in the IGS configuration.
- **Tag_Name** This is the name of the address within the PLC or other hardware device with which the server communicates. It must match the tag name for the specified channel and device in the IGS configuration.

Note: If tags were imported from a Controllogix L5K file, the full path to the tag name must be included.

Bit Addressing

Bit addressing can be accomplished by using one of the following two methods:

1. If a Digital Register (DR) block is being used, bits within integer data (or bits within Boolean array data) can be specified with the numbered fields F_0, F_1, F_2, and so forth. For more information, refer to Using Offset fields with Analog and Digital Registers (AR/DR).
2. If a DR block is not being used, a tag should be configured in the IGS server project with the desired bit specified in the tag address. Alternatively, specify the appropriate bit address in the block's I/O

address so that the tag may be dynamically created. For more information, refer to the IGS device driver help documentation.

Notes:

1. Users may also specify an integer tag in the I/O address of DA and DI blocks; however, only the least significant bit of that integer can be read or written to with these block types.
2. Because bit addressing is not supported when tags are imported from the L5K file, users must manually add bit addresses and their associated tag names in the IGS server configuration program. For example, assume that the global controller tag "ValveArea3" is configured as a short data type in the L5K import file. To address bit 1 of this tag in the iFIX PDB, users must first manually add the bit 1 address and its corresponding tag name in the IGS server configuration program. In this example, "ValveArea3_1" is the designated tag name for the bit 1 address. The I/O addressing for the bit address in the iFIX PDB is "Channel1.Device1.Global.ValveArea3_1".

Array Addressing

Many of the IGS server's device drivers support arrays. Users may access individual elements of an array tag using Analog Register (AR) blocks and the numbered fields F_0, F_1, F_2, and so forth. Digital Registers (DR) may be used to access any bit within any element of a Boolean or integer array. For more information, refer to Using Offset fields with Analog and Digital Registers (AR/DR).

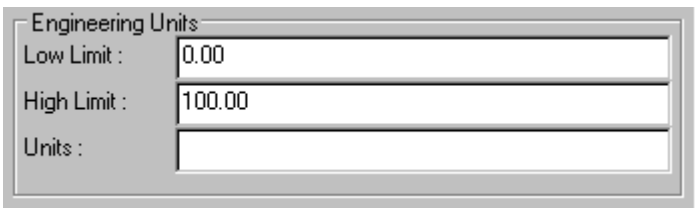
The entire array can be accessed in text form using a TX block. Access to individual elements or bits within an array using other means is not currently supported. If other types of blocks are used, the data must be addressed with individual tags. For more information on array addressing support and syntax, refer to the IGS device driver help documentation.

Specifying Signal Conditioning in iFIX Database Manager

The IGS driver can apply signal conditioning to the data. Users can configure signal conditioning options for each block defined in the iFIX Database Manager. For more information, refer to the instructions below.

1. In **Signal Conditioning**, specify the desired algorithm. For no signal conditioning, select **None**.

2. Specify the **Engineering Units (EGU)** range for the conditioned data.



Note: For more information on supported signal conditioning algorithms, refer to **iFIX Signal Conditioning Options**.

iFIX Signal Conditioning Options

The following signal conditioning options are available through the iFIX Database Manager:

Note: Linear and logarithmic scaling is available through the server for Static tags only. For more information, refer to Tag Properties — Scaling and Static Tags (User-Defined).

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=(((InputData-Lo_egu) / 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=((Raw_value/255) * 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 same algorithm as 8BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result=(((InputData-Lo_egu)/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:

Description	8-bit binary number
	Result =((Raw_value/255) * 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 an 8-bit register using the following algorithm: Result =(((InputData-Lo_egu)/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=((Raw_value/4095) * 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 same algorithm as 12BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result=(((InputData-Lo_egu)/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 =((Raw_value/4095) * 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:

Description	12-bit binary number
	Result =(((InputData-Lo_egu)/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=((Raw_value/8191) * 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 same algorithm as 13BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result=(((InputData-Lo_egu)/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: Result =((Raw_value/8191) * 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) * 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. Result=((Raw_value/16383) * 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 same algorithm as 14BN, and returns a status indicating whether the value is out of range and in an alarm state, or OK. Result=(((InputData-Lo_egu)/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: Result=((Raw_value/16383) * 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) * 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=((Raw_value/32767) * Span_egu) + Lo_egu.
Read Algorithm Vari-	Lo_egu - the database block's low engineering value.

Description	15-bit binary number
ables	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=(((InputData-Lo_egu)/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 =((Raw_value/32767) * 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) * 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	Lo_egu - the low engineering value.

Description	6400 – 32000 clamp
Variables	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.

Project Startup for iFIX Applications

The server's iFIX interface has been enhanced to provide iFIX users with better startup performance. This enhancement applies to iFIX applications that use Analog Output (AO), Digital Output (DO), and/or Alarm Values that were previously initialized improperly on startup. The server maintains a special iFIX configuration file for the default server project that contains all items that to be accessed by the iFIX client. This configuration file is used to automatically start scanning items before iFIX requests item data. Therefore, data updates that are only requested once (such as AO/DO) have an initial value when requested by iFIX. For information on using this feature for existing iFIX projects, refer to the instructions below.

1. To start, export the PDB database from the iFIX Database Manager.
2. Re-import the exported file so that each item in the database is re-validated with the server.
3. In the **Confirm Tag Replacement** message box, select **Yes to all**.

   Note: A new configuration file is created in the same folder as the default server project file, containing the name "default_FIX.ini".

4. Depending on how long it takes to read an initial value for all the items in the project, it may be necessary to delay the start of SAC processing. Doing so allows the server enough time to retrieve all initial updates before the iFIX client requests data from the server. For more information on the specific iFIX version, refer to the iFIX documentation.
5. Restart both the iFIX application and the server to put the changes into effect.

Note: For new projects (or when adding additional items to an existing iFIX database) users do not need to perform the steps described above. The item is validated by the server upon its addition to the database. If valid, the server adds the item to the configuration file.

Store and Forward Service

The Store and Forward Service allows different server components to store data on a local disk for a period of time. The service installs with components that require store and forward functionality. The Store and Forward service starts and stops automatically based on features that support store and forward.

See Also:
ThingWorx Project Properties
Store and Forward Configuration Settings
Store and Forward System Tags
ThingWorx Access Rights

Configuration API Service

The Configuration API allows an HTTP RESTful client to add, edit, read, and delete objects such as channels, devices, and tags in the server. The Configuration API offers the following features:

- Object definition in standard human-readable JSON data format
- Support for triggering and monitoring actions on some objects within the server
- Security via HTTP basic authentication and HTTP over SSL (HTTPS)
- Support for user-level access based on the User Manager and Security Policies Plug-In
- Transaction logging with configurable levels of verbosity and retention

Note: This document assumes familiarity with HTTP communication and REST concepts.

Initialization - The Configuration API is installed as a Windows service and starts automatically with the system.
Operation - The Configuration API supports connections and commands between the server and REST clients.
Shutdown - If the Configuration API must be stopped, use the Windows Service Control Manager to terminate the Configuration API service.

Security

REST clients to the Configuration API must use HTTP Basic Authentication. The user credentials are defined in the server User Manager.

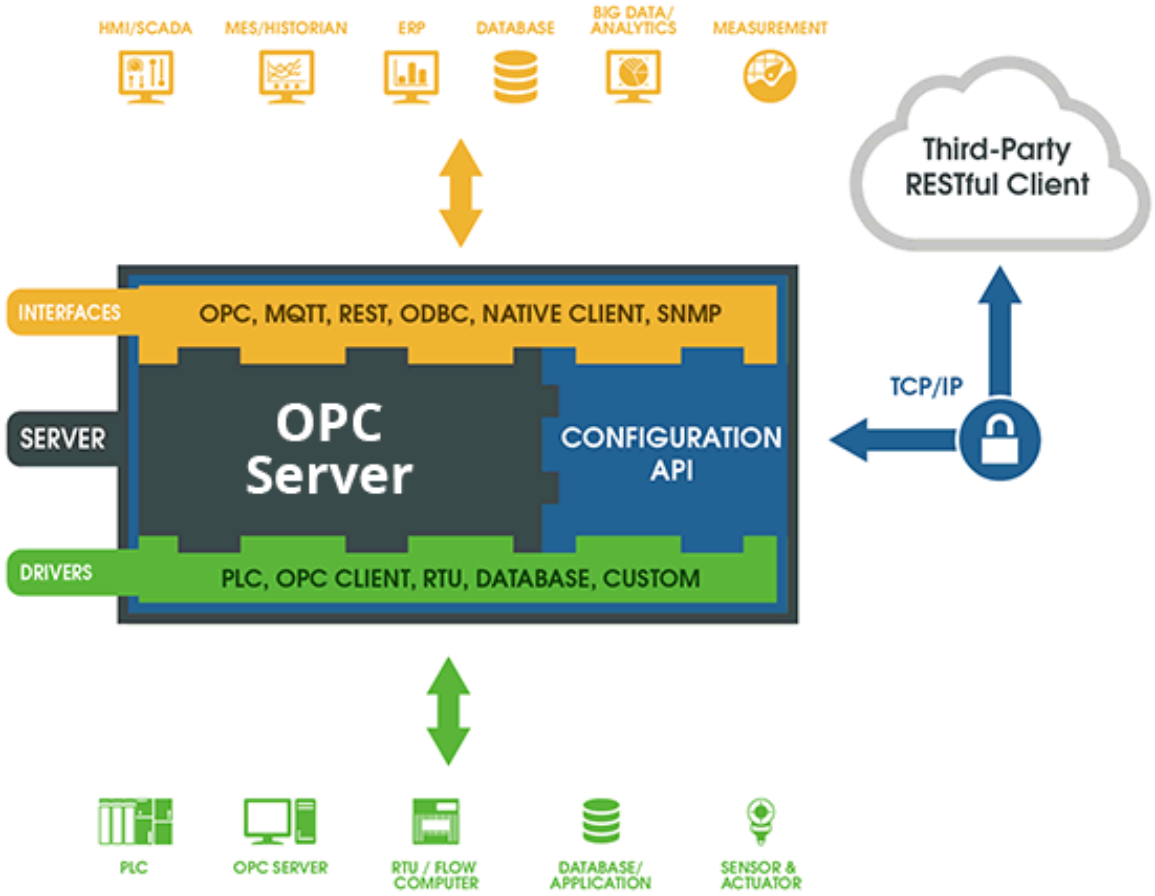
Documentation

Please consult additional information on properties, data ranges, endpoint mapping scheme, and acceptable actions for each endpoint is available at the Configuration API Landing Page at http://<localhost>:<port>/config/ (for default configurations).

Documentation served from the landing page is HTML-encoded by default. To obtain JSON-encoded documentation, include an "Accept" request header with "application/json".

Config API Service — Architecture

The diagram below shows the layout of the components. The Configuration API Service is installed on the same machine with the server.



Config API Service — Concurrent Clients

The Configuration API can serve multiple REST clients at the same time. To prevent a client from editing stale configurations, the Server Runtime maintains a numeric project ID. Each time an object is edited through the Configuration API or the local Configuration client, the Project ID changes. The current project ID is returned in each GET response. The current project ID must be specified by the client in all PUT requests.

The best practice is to issue a GET request, save the current project ID, and use that ID for the following PUT request. If only one client is used, the client may put the property "FORCE_UPDATE": true in the PUT request body to force the Configuration API server to ignore the project ID.

Config API Service — Logging

The Configuration API Transaction Log can be accessed from a REST client by sending a GET to http://<host-name>:<port>/config/v1/transaction_log. The response contains comma-separated entries.

Endpoint (GET):

```
http://<hostname/ip>:<port>/config/v1/event_log?limit=25
```

Command:

```
{
  "action": "GET",
  "endpoint": "/config/v1/drivers/channels/channel1",
  "response": 404,
```

```
  "source": "127.0.0.1",
  "timestamp": "2016-02-04T20:28:41.178",
  "user": "Administrator"
}
```

where:

- **Action** = The HTTP request method that was sent to the Configuration API
- **Endpoint** = The URL where the request was sent (excluding the IP address and port number)
- **Response** = The HTTP response code returned to the user
- **Source** = The IP address of the sender
- **Timestamp** = The time (UTC) that the response was sent from the Configuration API
- **User** = The user who sent the request

Example Return:

```
[
{
"timestamp": "2018-11-13T16:34:57.966",
"event": "Security",
"source": "KEPServerEX\\Runtime",
"message": "Configuration session started by admin as Default User (R/W)."
},
{
"timestamp": "2018-11-13T16:35:08.729",
"event": "Warning",
"source": "Licensing",
"message": "Feature Modbus TCP/IP Ethernet is time limited and will expire at
11/13/2019 12:00 AM."
}
…
]
```

Filtering: The Configuration API Transaction Log allows log items to be sorted or limited using filter parameters specified in the URI. The filters, which can be combined or used individually, allow the results of the log query to be restricted to a specific time period (e.g. events which occurred since a given date, events which occurred before a given date, or events that occurred between two dates). Example filtered log query:

```
http://<hostname>:<port>/config/v1/log?limit=10&start=2016-01-01T00:00:00.000&end-
d=2016-01-02T20:00:00.000
```

where:

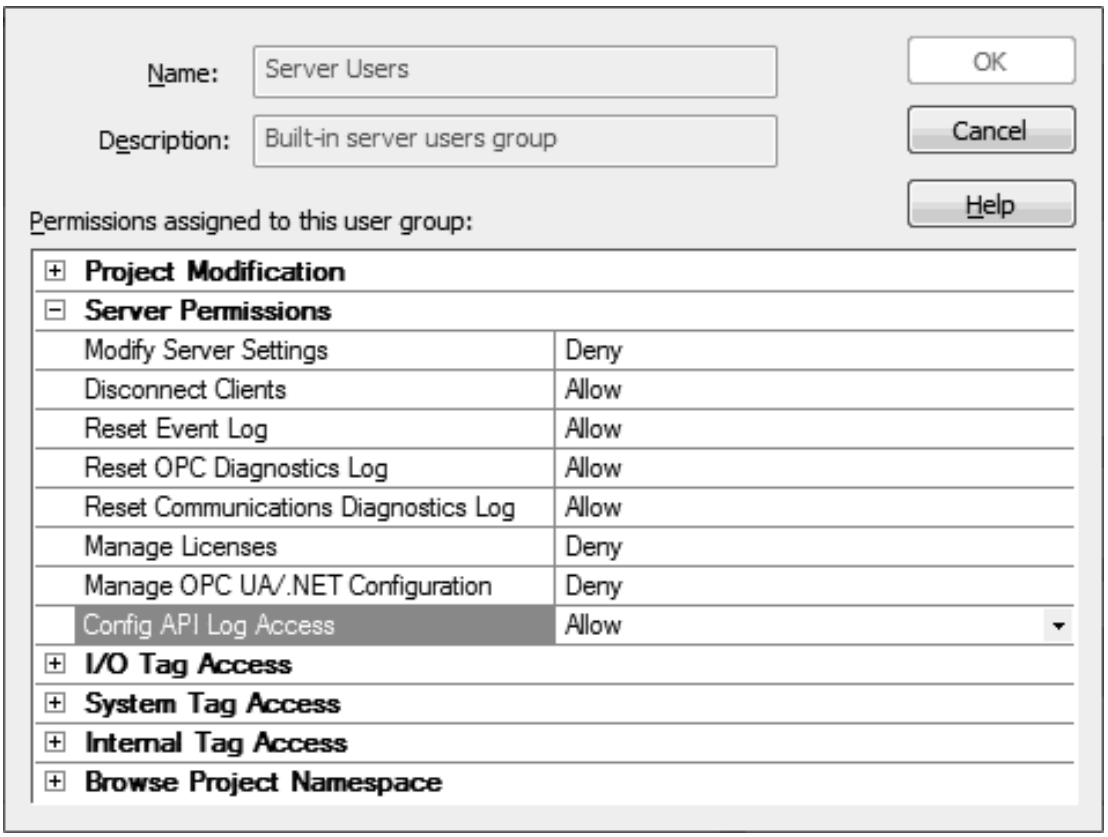
- **Limit** = Maximum number of log entries to return. The default setting is 100 entries.
- **Start** = Earliest time to be returned in YYYY-MM-DDTHH:mm:ss.sss (UTC) format.
- **End** = Latest time to be returned in YYYY-MM-DDTHH:mm:ss.sss (UTC) format.

Note: The Limit filter overrides the result of the specified time period. If there are more log entries in the time period than the Limit filter allows, the newest records that match the filter criteria are displayed.

Verbose Logging: records the request and response JSON bodies, which can be useful for troubleshooting. Turning on verbose logging can add two properties (requestbody and responsebody) to each log entry, depending on the request type. To turn on verbose logging, open **Settings | Configuration API Service | Transaction Logging** and change **Verbose** to **Yes**.

Warning: Verbose logging causes the transaction log to grow rapidly. Do not activate for normal use.
Note: Log queries are not logged in a verbose manner; the entries display the shorter format.

Logging Permissions: allows additional permission settings to prevent unauthorized users from accessing the log. The default is Deny for all non-administrator users.



See Also: *Refer to server help for more information on changing permissions in User Manager.*

Event Log: The Event Log service collects information, warning, error, and security events. The same filtering and query parameter definitions apply to the event log (http://<hostname>:<port>/config/v1/event_log) as they do to the transaction log.

Example:

```
{
  "event": "Warning",
  "message": "Delete object 'Channel1' failed: Active connections still exist.",

  "source": "KEPServerEX\\Runtime",
  "timestamp": "2017-01-18T21:51:38.495"
}
```

where:

- **Event** = The event type information, warning, error, security or debug.
- **Message** = The text portion of the event log message.
- **Source** = The component within the product where the message originated.
- **Timestamp** = The time (UTC) when the message was posted to the log.

- The Log endpoint provides access to the servers diagnostic and event log.

Config API Service — Content Retrieval

Content is retrieved from the server by issuing an HTTP GET request. The URI specified in the request can target one of the following areas:

1. Online documentation (ex. /config/v1/doc or /config/v1/doc/drivers)
2. Event log entries (ex. /config/v1/event_log)
3. Transaction log entries (ex. /config/v1/transaction_log)
4. Project configuration (ex. /config/v1/project or /config/v1/project/channels/Channel1)

When targeting project configuration, a REST client can specify the type(s) of content that should be returned. In this context the word "content" refers to a category or categories of data about a collection or object instance.

By default, when a GET request is issued using an endpoint that identifies a collection, the server will return a JSON array that contains one value for each instance in the collection where each value is a JSON object that contains the properties of the instance.

By default, when a GET request is made using an endpoint that identifies an object instance, the server will return a JSON object that contains the properties of that instance.

The default behavior of these requests can be altered by specifying one or more "content" query parameters appended to the URL as in http://<hostname>:<port>/config/v1/project?content=children. The following table shows the available content types and their applicability to each endpoint type:

Content Type	Collection Endpoint	Object Instance Endpoint
properties	yes	yes
property_definitions	no	yes
property_states	no	yes
type_definition	yes	yes
children	yes	yes

The following table shows the structure of the JSON response for a given content type:

GET Request URI	JSON Response Structure

/config/v1/project?content=properties	{
	<property name>: <value>,
	<property name>: <value>,
	...
	}

/config/v1/project?content=property_definitions	[{<property definition>}, {<property definition>}, ...]
/config/v1/project?content=property_states	{ "allow": {

GET Request URI	JSON Response Structure

	<property name>: true/false,
	<property name>: true/false,
	...
	},
	"enable":
	{
	<property name>: true/false,
	<property name>: true/false,
	...
	}
	}

/config/v1/project?content=type_definition	{
	"name": <type name>,
	"collection": <collection name>,
	"namespace": <namespace name>,
	"can_create": true/false,
	"can_delete": true/false,
	"can_modify": true/false,
	"auto_generated": true/false,
	"requires_driver": true/false,
	"access_controlled": true/false,
	"child_collections": [<collection names>]
	}

/config/v1/project?content=children	{
	<collection name>: [
	{
	"name": <object instance name>,
	"href": <object instance uri>
	},
	...
	],
	<collection name>: [
	{
	"name": <object instance name>,
	"href": <object instance uri>
	},
	...
	],
	...
	}

Multiple content types can be specified in the same request by separating with a comma. For example, http://<hostname>:<port>/config/v1/project?content=children,type_definition. When multiple types are specified, the JSON response will contain a single object with a member for each requested content type as in:

```
{
  "properties": <properties response structure>,
  "property_definitions": <property definitions response structure>,
  "property_states": <property states response structure>,
  "type_definition": <type definition response structure>,
  "children": <children response structure>
}
```

Type Definitions

The following table describes the members of the type definition JSON object.

Member	Type	Description
name	string	Object type name.
collection	string	Collection name. Identifies the collection in which objects of this type will exist. This name constitutes a valid endpoint that can be addressed using the REST interface.
namespace	string	Namespace that implements the object type. Objects that are implemented by the server exist in the “servermain” namespace. Other namespaces are defined by optional components such as drivers, plug-ins and client interfaces.
can_create	bool	Indicates whether or not instances of this type can be created by an end user. For example, this is false for the “Project” type because it’s not something that can be created.
can_delete	bool	Indicates whether or not instances of this type can be deleted by an end user. Again, the “Project” type is not something that can be deleted.
can_modify	bool	Indicates whether or not instances of this type can be modified by an end user. For example, the server has some auto-generated objects that exist to create a child collection only and do not themselves have any modifiable properties.
auto_generated	bool	If true, instances of this type are auto-generated by the server. Typically objects of this type will have the previous three members defined as “false”.
requires_driver	bool	True if instances of this type cannot be created without supplying the name of an installed driver.
access_controlled	bool	True if the server provides group-level access control over the CRUD operations that can be executed against an instance of this type *(see **User Manager**)*.
child_collections	array	An array of collection names that are supported as children under an object of this type. For example, if a type includes “devices” in “child_collections”, then object instances of that type will support one or more “Device” instance as a child.

Property Definitions

A property definition identifies the characteristics of a given property, including the type of data it supports, applicable ranges, default value, etc. The JSON structure of a property definition object is defined as follows:

Member	Type	Description
symbolic_name	string	Identifies the property by canonical name in the form <namespace>.<property name>.
display_name	localized string	The name the property would have if shown in the Server Configuration property editor. Value will be returned in the language the server is currently configured to use.
display_description	localized string	The description the property would have if shown in the Server Configuration property editor. Value will be returned in the language the server is currently configured to use.
read_only	Boolean	True if the property is informational, not expected to change once initially defined.
type	string	Determines the data type of the property value (see “Property Types” below).
minimum_value	number or null (applies to numeric types)	Minimum value the property can have to be considered valid. If null, there is no minimum.

Member	Type	Description
maximum_value	number or null (applies to numeric types)	Maximum value the property can have to be considered valid. If null, there is no maximum.
minimum_length	number (applies to strings only)	Minimum length a string value may have. 0 means no minimum.
maximum_length	number (applies to strings only)	Maximum length a string value may have. -1 means no maximum.
hints	arrays of strings (applies to strings only)	An array of possible choices that may be assigned to the property value. This member not included if no hints exist.

enumeration	object (applies to enumerations only)	For enumeration properties, this object identifies the valid name / value pairs the enumeration can have. Structure is as follows:
		{
		<name>: number,
		<name>: number,
		...
		}

allow	array of objects	Defines a conditional dependency on one or more other properties that determines whether this property is relevant. Properties that are not allowed are not shown in the Server Configuration property editor (see “Allow and Enable Conditions” below).
enable	array of objects	Defines a conditional dependency on one or more other properties that determines whether this property should be enabled for the client to change. Properties that are not enabled are grayed out in the Server Config property editor (see “Allow and Enable Conditions” below).

To get specific information about the property definitions of a specific endpoint, add "?content=property_definitions" to the end of the URL of a GET request.

For example, to get the property definitions for a channel named Channel1 with the server running on the local host, the GET request would be sent to:

```
/config/v1/channels/Channel1?content=property_definitions
```

The returned JSON block would look something like the following:

```
[
  {
    "symbolic_name": "common.ALLTYPES_NAME",
    "display_name": "Name",
    "display_description": "Specify the identity of this object.",
    "read_only": false,
    "type": "String",
    "default_value": null,
    "minimum_length": 1,
    "maximum_length": 256
```

```
  },
  {
    "symbolic_name": "common.ALLTYPES_DESCRIPTION",
    "display_name": "Description",
    "display_description": "Provide a brief summary of this object or its use.",
    "read_only": false,
    "type": "String",
    "default_value": null,
    "minimum_length": 0,
    "maximum_length": 255
  },
…
```

Property Types

The following table describes the different values that a property definition may contain for the "type" member. The "Value Type" identifies what JSON type the property value should have.

Type Name	Value Type	Description
AllowDeny	bool	Describes a property that would show a drop-down list that contains the choices "Allow"=true and "Deny"= false.
EnableDisable	bool	Describes a property that would show a drop-down list that contains the choices "Enable"=true and "Disable"= false.
YesNo	bool	Describes a property that would show a drop-down list that contains the choices "Yes"=true and "No"= false.
String	string	Generic string. Properties of this type will include minimum_length and maximum_length specifiers.
StringArray	array	Array of strings. Properties of this type will include minimum_length and maximum_length specifiers that apply to the strings themselves, not the length of the array.
Password	string	Obfuscated string that contains a password. When changing the value of a property of this type, a plain-text password is expected. Password values should only be changed over a secure connection.
LocalFileSpec	string	A fully qualified file specification in the local Windows file system.
UncFileSpec	string	A fully qualified file specification in a network location.
LocalPathSpec	string	A fully qualified path specification in the local Windows file system.
UncPathSpec	string	A fully qualified path specification to a network location.
StringWithBrowser	string	Describes a property that has a string value that would normally be chosen from a collection of dynamically generated strings.
Integer	number	Unsigned 32-bit integer value.
Hex	number	Unsigned 32-bit integer value intended to be displayed / edited in hexadecimal notation.
Octal	number	Unsigned 32-bit integer value intended to be displayed / edited in octal notation.
SignedInteger	number	Signed 32-bit integer value.
Real4	number	Single precision floating point value.
Real8	number	Double precision floating point value.

Type Name	Value Type	Description
Enumeration	number	One of the possible numeric values from the “enumeration” member of the property definition.
PropArray	object	Describes a structure containing members that each have a fixed-length array of values.
TimeOfDay	number	Integer value containing the number seconds since midnight that would define a specific time of day.
Date	number	Unix time value that specifies midnight on a given date.
DateAndTime	number	Unix time value that specifies a specific time on a given date.
Blob	array	Array of byte values that represents an opaque collection of data. Data of this type originates in the server and is hashed to prevent modification.

Allow and Enable Conditions

For definitions that contain allow and/or enable conditions, this is the structure they would have in the JSON:

```
<condition>: [
  {
    “depends_on”: <property name>
    “operation”: “==” or “!=”
    “value”: <value>
  },
...
]
```

Each condition identifies another property that is a dependent and how it depends as equal or not equal to the value of that property. More than one dependency can exist, either on the same property or different ones. If multiple exist, the “operation” will always be the same. Evaluation of the expression to determine the state of the condition when multiple dependencies exist is a logical “or” for “==” and a logical “and” for “!=”.

When using “content=property_states”, the returned JSON describes the outcome of the evaluation of these conditions (if they exist) for each property.

Language Specifications

The server supports multiple languages. It will return localized text to the client in the language it is configured to use. The client can override the configured language in a GET request by specifying an “Accept-Language” field in the request header.

See https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.

As an example, if the server is configured for English and the client wants German, it can specify the following in the request header: “Accept-Language: de”

Note: If the client specifies a language that is not supported by the server, the currently configured language will be used.

Config API Service — Data

The Configuration API Service receives requests in standard JSON format from the REST client. These requests are consumed by the server and broken down into create, read, update, or delete commands.

Please consult additional information on properties, data ranges, endpoint mapping scheme, and acceptable actions for each endpoint is available at the Configuration API Landing Page at http://<localhost>:<port>/config/ (for default configurations).

Documentation served from the landing page is HTML-encoded by default. To obtain JSON-encoded documentation, include an "Accept" request header with "application/json".

Object names containing spaces, or other characters disallowed in URL formatting, must be percent-encoded to be correctly interpreted by the Configuration API. Percent encoding involves replacing disallowed characters with their hexadecimal representation. For example, an object named 'default object' is percent-encoded as default%20object. The following characters are not permitted in a URL and must be encoded:

space	!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%20	%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

All leading and trailing spaces are removed from object names before the server validates them. This can create a discrepancy between the object name in the server and the object name a user provides via the Configuration API. Users can send a GET on the parent object after sending a PUT/POST to verify the new or modified object name in the server matches what was sent via the API.

The priority order for dialing entries in the phonebook can be changed via the Configuration API by writing to the servermain.PHONEBOOK_PRIORITY property of the PhonePriority object. This string property is a comma-delimited list of quoted phonebook entry names.

An example GET request to endpoint http://{localhost:number}/config/v1/project/channels/{channel name}/-phonebooks/phonebook/phonePriorities/ returns:

```
{
  "PROJECT_ID": 1270990535,
  "common.ALLTYPES_DESCRIPTION": "",
  "common.ALLTYPES_NAME": "PhonePriority",
  "servermain.PHONEBOOK_PRIORITY": "\"Phone3\",\"Phone1\",\"Phone2\""
}
```

An example PUT request to endpoint http://{localhost:number}/config/v1/project/channels/{channel name}/-phonebooks/phonebook/phonePriorities/PhonePriority to change the servermain.PHONEBOOK_PRIORITY property:

```
{
  "PROJECT_ID": 1270990535,
  "servermain.PHONEBOOK_PRIORITY": "\"Phone1\",\"Phone2\",\"Phone3\""
}
```

Percent encoding does not guarantee that a name is valid. To determine the valid name values, refer to the documentation for the specific object being created.

The quotes around the phonebook entry names in a string property must be escaped with a backslash character (\"). The server configuration application is recommended for selecting and configuring a modem; not the API.

An attempt to perform a POST/PUT/DELETE with the API as a non-admin user fails if a user has the server configuration open at the same time. The error is a 401 status code (unauthorized). Only one user can write

to the runtime at a time; the API cannot take permissions from the server configuration if it has insufficient credentials.

Create an Object

An object can be created by sending an HTTP POST request to the Configuration API. When creating a new object, the JSON must include required properties for the object (ex. each object must have a name), but doesn't require all properties. All properties not included in the JSON are set to the default value on creation. Example POST JSON body:

```
{
  "<Property1_Name>": <Value>,
  "<Property2_Name>": <Value>,
  "<Property3_Name>": <Value>
}
```

Create Multiple Objects

Multiple objects may be added to a given collection by including the JSON property objects in an array. Example POST JSON body:

```
[
  {
    "<Property1_Name>": <Value>,
    "<Property2_Name>": <Value>,
    "<Property3_Name>": <Value>
  },
  {
    "<Property1_Name>": <Value>,
    "<Property2_Name>": <Value>,
    "<Property3_Name>": <Value>
  }
               ]
```

When a POST includes multiple objects, if one or more cannot be processed due to a property validation failure or some other error, the HTTP status code 207 (Multi-Status) will be returned along with a JSON object array containing the status for each object in the request. For example, if two objects are included in the request and the second one specifies the same name as the first:

```
[
  {
    "code": 201,
    "message": "Created"
  },
  {
    "code": 400,
    "message": "Validation failed on property common.ALLTYPES_NAME in object definition at line 7: The name 'Channel1' is already used."
  }
]
```

Create an Object with Child Hierarchy

An object may be created with a full child object hierarchy beneath it. To do this, include that hierarchy in the POST request just as it would appear when saved in a JSON project file. For example, to create a channel with a device underneath it, the following JSON could be used:

```
[
  {
       "common.ALLTYPES_NAME": "Channel1",
       "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
       "devices":
       [
         {
         "common.ALLTYPES_NAME": "Device1",
         "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
         "servermain.DEVICE_MODEL": 0
         }
       ]
  }
]
```

Read an Object

An object can be read by sending an HTTP GET request to the Configuration API. All object properties are returned on every GET request and each object includes a Project_ID. The Project_ID property is used to track changes in the configuration and is updated on any change from the Configuration API or a server configuration client. This property should be saved and used in all PUT requests to prevent stale data manipulations. Example response body:

```
{
   "<Property1_Name>": <Value>,
   "<Property2_Name>": <Value>,
   "PROJECT_ID": 12345678
}
```

See Also: Content Retrieval

Edit an Object

An object can be edited by sending an HTTP PUT request to the Configuration API. PUT requests require the Project_ID or Force_Update property in the JSON body. Setting Force_Update to True ignores Project_ID validation. Example PUT body:

```
{
  "<Property1_Name>": <Value>,
  "<Property2_Name>": <Value>,
  "PROJECT_ID": 12345678,
  "FORCE_UPDATE": true
}
```

Normally when a PUT request succeeds and all properties are assigned successfully, there is no response body returned to the client, there is simply a 200 status code to indicate success. There can be cases where a property is included in a PUT request that is not assigned to the object instance by the Server Runtime. In these cases, a response body will be generated as follows:

```
{,
  "not_applied":,
    {,
    "servermain.CHANNEL_UNIQUE_ID": 2466304381
    },
  "code": 200,
```

```
  "message": "Not all properties were applied. This could be due to active client
reference or property is disallowed/disabled/read-only."
}
```

The response indicates which property or properties were not applied to the object instance where each contains the value that is actually in use. There are several possible reasons why the property value could not be applied, such as:

- The property is read-only and cannot be changed.
- There is a client reference on the object that restricts what properties can be updated.
- The property is not allowed based on the values of other properties on which this condition depends.
- The property is not enabled based on the values of other properties on which this condition depends.
- The value was transformed in some way (ex. rounded or truncated).

Delete an Object

An object can be deleted by sending an HTTP DELETE request to the Configuration API. The Configuration API does not allow deleting multiple items on the same level with a single request (such as deleting all of the devices in a channel), but can delete an entire tree (such as deleting a device deletes all its child tags).

Errors

All Configuration API Service requests return errors in JSON format. Example:

```
{
  "code": 400,
  "message": "Invalid property: 'NAME'."
}
```

See Also: Troubleshooting

Channel Configuration API Commands

The following commands define a channel using the Configuration API service.

General Properties

`common.ALLTYPES_NAME` * Required parameter.

Note: Changing this property causes the API endpoint URL to change.

`common.ALLTYPES_DESCRIPTION`

`servermain.MULTIPLE_TYPES_DEVICE_DRIVER` * Required parameter

`servermain.CHANNEL_DIAGNOSTICS_CAPTURE`

Ethernet Communication Properties

`servermain.CHANNEL_ETHERNET_COMMUNICATIONS_NETWORK_ADAPTER_STRING`

Advanced Properties

`servermain.CHANNEL_NON_NORMALIZED_FLOATING_POINT_HANDLING` * Required parameter

Write Optimizations

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD`

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_DUTY_CYCLE`

***See Also**: The server help system Configuration API Service section.*

Config API Service — Creating a Channel

To create a channel via the Config API service, only a minimum set of properties are required; all others are set to the default value. Once a channel is defined, its properties and settings are used by all devices assigned to that channel. The specific properties are dependent on the protocol or driver selected.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the channel end-point.

The example below creates a channel named Channel1 that uses the Simulator driver on a server running on the local host.

```
POST    http://<localhost>:<port>/config/v1/project/channels

{
  "common.ALLTYPES_NAME": "Channel1",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator"
}
```

Refer to the driver specific help documentation to find out what properties are required to create a channel for that driver.

Config API Service — Updating a Channel

To update a property or collection of properties on a channel, a GET request must first be sent to the end-point to be updated to get the Project ID.

For more information about the Project ID see the Concurrent Clients section.

In the example below, the channel being updated is Channel1.

```
GET     http://<localhost>:<port>/config/v1/project/channels/Channel1
```

The GET request will return a JSON blob similar to the following.

```
{
  "PROJECT_ID": 1988502408,
  "common.ALLTYPES_NAME": "Channel1",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
```

```
  "servermain.CHANNEL_DIAGNOSTICS_CAPTURE": false,
  "servermain.CHANNEL_UNIQUE_ID": 2154899492,
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD": 2,
…
```

To update or change a channel property, a PUT request is sent to the channel with the Project ID and the new property value defined. In the following example, the channel name will change from Channel1 (from above) to Simulator.

```
PUT     http://<localhost>:<port>/config/v1/project/channels/Channel1
```

```
{
  "PROJECT_ID": 1988502408,
  "common.ALLTYPES_NAME": "Simulator"
}
```

Following the PUT, a GET can be sent to the channel's endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

Note: Some properties are client restricted and cannot be changed when a client is connected.

```
GET     http://<localhost>:<port>/config/v1/project/channels/Simulator
```

The response from the GET request should show the property value has changed. The response to the GET above should look similar to the following:

```
{
  "PROJECT_ID": 2899413628,
  "common.ALLTYPES_NAME": "Simulator",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.CHANNEL_DIAGNOSTICS_CAPTURE": false,
  "servermain.CHANNEL_UNIQUE_ID": 2154899492,
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD": 2,
…
```

Config API Service — Removing Channel

To remove a channel, send a DELETE command to the channel endpoint to be removed. This causes the channel and all of its children to be removed.

In the example below, the channel Simulator will be removed.

```
DELETE  http://<localhost>:<port>/config/v1/project/channels/Simulator
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a GET to the "channels" endpoint; the removed channel will not be in the list of channels returned from the GET request.

Device Configuration API Commands

The following commands define a channel using the Configuration API service.

General Properties

`common.ALLTYPES_NAME`

common.ALLTYPES_DESCRIPTION

servermain.DEVICE_CHANNEL_ASSIGNMENT

servermain.MULTIPLE_TYPES_DEVICE_DRIVER

servermain.DEVICE_MODEL

servermain.DEVICE_ID_STRING

servermain.DEVICE_DATA_COLLECTION

servermain.DEVICE_SIMULATED

Scan Mode

servermain.DEVICE_SCAN_MODE * Required parameter

servermain.DEVICE_SCAN_MODE_RATE_MS

servermain.DEVICE_SCAN_MODE_RATE_MS

servermain.DEVICE_SCAN_MODE_PROVIDE_INITIAL_UPDATES_FROM_CACHE

Auto Demotion

servermain.DEVICE_AUTO_DEMOTION_ENABLE_ON_COMMUNICATIONS_FAILURES

servermain.DEVICE_AUTO_DEMOTION_DEMOTE_AFTER_SUCCESSIVE_TIMEOUTS

servermain.DEVICE_AUTO_DEMOTION_PERIOD_MS

servermain.DEVICE_AUTO_DEMOTION_DISCARD_WRITES

Tag Generation

servermain.DEVICE_TAG_GENERATION_ON_STARTUP * Required parameter

servermain.DEVICE_TAG_GENERATION_DUPLICATE_HANDLING * Required parameter

servermain.DEVICE_TAG_GENERATION_GROUP

servermain.DEVICE_TAG_GENERATION_ALLOW_SUB_GROUPS

Tip: To Invoke Automatic Tag Generation, send a PUT with an empty body to the TagGeneration service endpoint on the device.
See Also: *For more information see Services help.*

Timing

servermain.DEVICE_CONNECTION_TIMEOUT_SECONDS

servermain.DEVICE_REQUEST_TIMEOUT_MILLISECONDS

servermain.DEVICE_RETRY_ATTEMPTS

servermain.DEVICE_INTER_REQUEST_DELAY_MILLISECONDS

***See Also**: The server help system Configuration API Service section.*

Config API Service — Creating a Device

To create a device via the Config API service, only a minimum set of properties are required; all others are set to the default value. The specific properties are dependent on the protocol or driver selected.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the devices end-point under a channel.

The example below will create a device named Device1 under Channel1 that uses the Simulator driver on a server running on the local host.

```
POST    http://<localhost>:<port>/config/v1/project/channels/Channel1/devices
```

```
{
  "common.ALLTYPES_NAME": "Device1",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator"
}
```

Refer to the driver specific help documentation to find out what properties are required to create a device for that driver.

Config API Service — Updating a Device

To update a property or collection of properties on a device, a GET request must first be sent to the endpoint to be updated to get the Project ID.

*For more information about the Project ID, see the **Concurrent Clients** section.*

In the example below, the device being updated is Device1 under Channel1.

```
GET     http://<localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1
```

The GET request will return a JSON blob similar to the following.

```
{
  "PROJECT_ID": 1342460467,
  "common.ALLTYPES_NAME": "Device1",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.DEVICE_MODEL": 0,
  "servermain.DEVICE_UNIQUE_ID": 3671734680,
  "servermain.DEVICE_CHANNEL_ASSIGNMENT": "Channel1",
…
```

To update or change a device property a PUT request is sent to the device with the Project ID and the new property value defined. In the following example the device name will change from Device1 (from above) to Simulator.

```
PUT     http://<localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1
```

```
{
  "PROJECT_ID": 1342460467,
  "common.ALLTYPES_NAME": "Simulator"
}
```

Following the PUT, a GET can be sent to the device endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

Note: Some properties are client restricted and cannot be changed when a client is connected.

```
GET     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Simulator
```

The response from the GET request will show the property value has changed. The response to the GET above should look similar to the following.

```
{
  "PROJECT_ID": 2135279368,
  "common.ALLTYPES_NAME": "Simulator",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.DEVICE_MODEL": 0,
  "servermain.DEVICE_UNIQUE_ID": 3671734680,
  "servermain.DEVICE_CHANNEL_ASSIGNMENT": "Channel1",
…
```

Config API Service — Removing a Device

To remove a device, send a DELETE to the device endpoint to be removed. This will cause the device and all of its children to be removed.

In the example below, the device Simulator will be removed.

```
DELETE  http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Simulator
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a get to the devices endpoint and the removed device will not be in the list of devices returned from the GET request.

Config API Service — Creating a Tag

To create a tag via the Config API service, only a minimum set of properties are required; all others are set to the default value. The specific properties are dependent on the protocol or driver selected.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the tags endpoint under a device.

The example below will create a tag named MyTag for address R5 under Channel1/Device1 that uses the Simulator driver on a server running on the local host.

```
POST    http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags
```

```
{
  “common.ALLTYPES_NAME”: “MyTag”,
  "servermain.TAG_ADDRESS": "R5"
}
```

Tags can also be created within a tag group. The process for adding a tag group is the same except the URL will change to include the tag_group endpoint and the group name.

In the following example, the tag group RampTags already exists and a tag named MyTag will be created under it with the address R5.

For more information on creating a tag group see Creating a Tag Group section.

```
POST    http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_
group/RampTags/tags
```

```
{
  “common.ALLTYPES_NAME”: “MyTag”,
  "servermain.TAG_ADDRESS": "R5"
}
```

Refer to the driver specific help documentation to find out what properties are required to create a tag for that driver.

Config API Service — Updating a Tag

To update a property or collection of properties on a tag, a GET request must first be sent to the endpoint to be updated to get the Project ID.

*For more information about the Project ID see the **Concurrent Clients** section.*

In the example below, the tag being updated is MyTag under Channel1/Device1.

```
GET     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags/MyTag
```

The GET request will return a JSON blob similar to the following.

```
{
  "PROJECT_ID": 1473736741,
  "common.ALLTYPES_NAME": "MyTag",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAG_ADDRESS": "R0005",
  "servermain.TAG_DATA_TYPE": 5,
  "servermain.TAG_READ_WRITE_ACCESS": 1,
  "servermain.TAG_SCAN_RATE_MILLISECONDS": 100,
…
```

To update or change a tag property, a PUT request is sent to the tag with the Project ID and the new property value defined.
In the following example, the tag name will change from MyTag (from above) to Tag1.

```
PUT     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags/MyTag
```

```
{
  "PROJECT_ID": 1473736741,
  "common.ALLTYPES_NAME": "Tag1"
}
```

Following the PUT a GET can be sent to the tag’s endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

```
GET     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags/Tag1
```

The response from the GET request will show the property value has changed. The response to the GET above should look similar to the following.

```
{
  "PROJECT_ID": 1239878456,
  "common.ALLTYPES_NAME": "Tag1",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAG_ADDRESS": "R0005",
  "servermain.TAG_DATA_TYPE": 5,
  "servermain.TAG_READ_WRITE_ACCESS": 1,
  "servermain.TAG_SCAN_RATE_MILLISECONDS": 100,
…
```

Config API Service — Removing a Tag

To remove a tag, send a DELETE to the tag's endpoint to be removed. This will cause the tag and all of its children to be removed.
In the example below, the tag Tag1 will be removed.

```
DELETE  http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags/Tag1
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a get to the tags endpoint and the removed tag will not be in the list of tags returned from the GET request.

Config API Service — Creating a Tag Group

To create a tag group via the Config API service, only a group name is required.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the tag_groups endpoint under a device.

The example below will create a tag group named RampTags under Channel1/Device1 that uses the Simulator driver on a server running on the local host.

```
POST    http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_groups
```

```
{
  "common.ALLTYPES_NAME": "RampTags"
}
```

Tag groups can have tags and more tag groups nested under them. *To add a Tag, see the* ***Creating a Tag*** *section.*

To nest a Tag Group within another group, another POST action is required to add the existing group name and the tag_groups endpoint to the end of the URL.

Continuing the example above, the new request would look like the following.

```
POST    http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_
groups/RampTags/tag_groups
```

```
{
  "common.ALLTYPES_NAME": "1-10"
}
```

Config API Service — Updating a Tag Group

To update a property or collection of properties on a tag, a GET request must first be sent to the endpoint to be updated to get the Project ID.

*For more information about the Project ID, see the **Concurrent Clients** section.*

In the example below, the tag group being updated is RampTags under Channel1/Device1.

```
GET     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_
groups/RampTags
```

The GET request will return a JSON blob similar to the following.

```
{
  "PROJECT_ID": 3115312875,
  "common.ALLTYPES_NAME": "RampTags",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAGGROUP_LOCAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_TOTAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_AUTOGENERATED": false
}
```

To update or change a tag group property, a PUT request is sent to the tag group with the Project ID and the new property value defined.
In the following example, the tag group name will change from RampTags (from above) to RampGroup.

```
PUT     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags/MyTag
```

```
{
  "PROJECT_ID": 3115312875,
  "common.ALLTYPES_NAME": "RampGroup"
}
```

Following the PUT, a GET can be sent to the tag group endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

```
GET     http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_
groups/RampGroup
```

The response from the GET request will show the property value has changed. The response to the GET above should look similar to the following.

```
{
  "PROJECT_ID": 4224221764,
  "common.ALLTYPES_NAME": "RampTags",
```

```
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAGGROUP_LOCAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_TOTAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_AUTOGENERATED": false
}
```

Config API Service — Removing a Tag Group

To remove a tag group, send a DELETE to the tag group endpoint to be removed. This will cause the tag group and all of its children to be removed. In the example below the tag group RampGroup will be removed.

```
DELETE  http://<-
localhost>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_
groups/RampGroup
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a get to the tag_groups endpoint and the removed tag group will not be in the list of tag groups returned from the GET request.

Config API Service — Creating a User

To create a user via the Config API service, only a minimum set of properties are required; all others are set to the default value.

- Only members of the Administrators group can create users.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the server_users endpoint.

The example below creates a user named User1 that is a member of the Administrators user group:

```
POST    http://<localhost>:<port>/config/v1/admin/server_users
```

```
{
  “common.ALLTYPES_NAME”: “User1”,
  "libadminsettings.USERMANAGER_USER_GROUPNAME": "Administrators"
}
```

Config API Service — Updating a User

To update a user via the Config API service, provide new values for the properties that require updating.

- Only members of the Administrators group can update users.
- There is no PROJECT_ID field for users.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the server_users/<username> endpoint.

The example below updates the user named User1 to add a description and move it to a different user group:

```
POST    http://<localhost>:<port>/config/v1/admin/server_users/User1
```

```
{
  "common.ALLTYPES_DESCRIPTION": "The user account of User1",
```

```
  "libadminsettings.USERMANAGER_USER_GROUPNAME": "Operators"
}
```

Config API Service — Creating a User Group

To create a group via the Config API service, only a minimum set of properties are required; all others are set to the default value. Once a user group is defined, its permissions are used by all users assigned to that user group.

- Only members of the Administrators group can create user groups.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the server_user-groups endpoint.

The example below creates a user group named Operators:

```
POST    http://<localhost>:<port>/config/v1/admin/server_usergroups
```

```
{
  "common.ALLTYPES_NAME": "Operators",
}
```

Config API Service — Updating a User Group

To edit a user group via the Config API service, provide new values for the properties that require updating.

- Only members of the Administrators group can update user groups.
- There is no PROJECT_ID field for user groups.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a PUT request to the server_user-groups/<groupname> endpoint.

The example below updates the user group named Operators to have permissions to modify server settings, cause clients to be disconnected, and loading new runtime projects; it also updates the description of the group:

```
POST    http://<localhost>:<port>/config/v1/admin/server_usergroups/Operators
```

```
{
  "common.ALLTYPES_DESCRIPTION": "User group for standard operators",
  "libadminsettings.USERMANAGER_SERVER_MODIFY_SERVER_SETTINGS": true,
  "libadminsettings.USERMANAGER_SERVER_DISCONNECT_CLIENTS": true,
  "libadminsettings.USERMANAGER_SERVER_REPLACE_RUNTIME_PROJECT": true
}
```

Config API Service — Configuring a User Group Project Permissions

All user groups contain a collection of project permissions. Each project permission corresponds to a specific permission applied when interacting with objects in the project. All permissions are always present under a user group (and therefore cannot be created nor deleted). An individual project permission can be granted or denied by updating that specific project permission under the desired User Group.

- Only members of the Administrators group can update a user group's project permissions.
- There is no PROJECT_ID field for project permissions.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a PUT request to the project_permissions/<permission_name> endpoint.

The example below updates the user-created user group named Operators to grant permission to users of that group to add, edit, and delete channels:

```
POST    http://<localhost>:<port>/config/v1/admin/server_user-
groups/Operators/project_permissions/Servermain Channel
```

```
{
  "libadminsettings.USERMANAGER_PROJECTMOD_ADD": true,
  "libadminsettings.USERMANAGER_PROJECTMOD_EDIT": true,
  "libadminsettings.USERMANAGER_PROJECTMOD_DELETE": true
}
```

Config API Service — Configuring a User Group Project Permissions

All user groups contain a collection of project permissions. Each project permission corresponds to a specific permission applied when interacting with objects in the project. All permissions are always present under a user group (and therefore cannot be created nor deleted). An individual project permission can be granted or denied by updating that specific project permission under the desired User Group.

- Only members of the Administrators group can update a user group's project permissions.
- There is no PROJECT_ID field for project permissions.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a PUT request to the project_permissions/<permission_name> endpoint.

The example below updates the user-created user group named Operators to grant permission to users of that group to add, edit, and delete channels:

```
POST    http://<localhost>:<port>/config/v1/admin/server_user-
groups/Operators/project_permissions/Servermain Channel
```

```
{
  "libadminsettings.USERMANAGER_PROJECTMOD_ADD": true,
  "libadminsettings.USERMANAGER_PROJECTMOD_EDIT": true,
  "libadminsettings.USERMANAGER_PROJECTMOD_DELETE": true
}
```

Config API Service — Invoking Services

Objects may provide services if there are actions that can be invoked on the object beyond the standard CRUD (Create, Retrieve, Update, Delete) operations. Services provide an asynchronous programmatic interface through which remote clients can trigger and monitor these actions. Services can be found in a collection called 'services' underneath the object on which they operate. For example, the project load service is located at the /config/v1/project/services/ProjectLoad endpoint as it operates on the project. Any object may provide services, so query if the service collection exists, then query the collection to see the available services.

Service Architecture

Services are designed to provide stateless interaction with the object on which they operate. Services are comprised of two components: a service and a job. The job executes the work asynchronously and provides a mechanism through which a client can monitor the job for completion or for any errors that occurred

during its operation. After a job completes, it is scheduled for deletion automatically by the server; no action is required by the client to clean up the job after it completes.

Service
The service is the interface through which an action is invoked. The service exposes all parameters that can be specified during its invocation as properties. To see the available parameters, perform a HTTP GET on the service endpoint. All properties, besides the name and description of the service, are the parameters that can be included when invoking a service. Depending on the service, some or all parameters may be required.

Invocation of a service is accomplished by performing a HTTP PUT request on the service endpoint with any parameters specified in the body of the request. Services may limit the total number of concurrent invocations. If the maximum number of concurrent invocations has been reached, the request is rejected with an "HTTP 429 Too Many Requests" response. If the limit has not been reached, the server responds with an "HTTP 202 Accepted" response and the body of the response including a link to the newly created job.

Successful PUT response example:

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/services/ProjectLoad/jobs/job1"
}
```

Busy PUT response example:

```
{
  "code": 429,
  "message": "The server is busy. Retry the operation at a later time."
}
```

Job
The job represents a specific request accepted by the server. To check the status of a job perform a HTTP GET request on the job endpoint. The **servermain.JOB_COMPLETE** property represents the current state of the job as a Boolean. The value of this property remains false until the job has finished executing. If the job fails to execute for any reason, it provides the client with an appropriate error message in the **servermain.JOB_STATUS_MSG** property.

Job Cleanup
Jobs are automatically deleted by the server after a configurable amount of time. By default, after a job has completed, the client has 30 seconds to interact with it before the job is deleted. If a longer amount of time is required by the client or the client is operating over a slow connection, the client can use the **servermain.JOB_TIME_TO_LIVE_SECOND** parameter when invoking the service to increase the time-to-live up to a maximum of five minutes. Each job has its own time-to-live and it may not be changed after a job has been created. Clients are not allowed to manually delete jobs from the server so it is best to choose the shortest time-to-live without compromising the client's ability to get the information from the job before it is deleted.

Service Automatic Tag Generation

The Automatic Tag Generation service operates under a device endpoint for a driver that supports Automatic Tag Generation. The properties that support Automatic Tag Generation for the device must be configured prior to initiating Automatic Tag Generation. *See the driver specific documentation for related properties.*

To initiate Automatic Tag Generation, a PUT is sent to the TagGeneration endpoint with a defined empty payload. In the following example, Automatic Tag Generation is initiated on Channel1/Device1.

```
PUT     /config/v1/project/channels/Channel1/devices/Device1/services/TagGeneration
```

The response should look something like the following.

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/con-
fig/v1/project/channels/Channel1/devices/Device1/services/TagGeneration/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job. This is done by sending a GET to the job's endpoint. The GET request should look like the following.

```
GET     /con-
fig/v1/project/channels/Channel1/devices/Device1/services/TagGeneration/jobs/job1
```

Jobs are automatically cleaned up after their wait time has expired. This wait time is configurable. *See the **Job Cleanup** section for more information.*

Note: Not all drivers support Automatic Tag Generation.

Service Project Load

Projects can be loaded by interacting with the ProjectLoad service on the ProjectLoad endpoint. First a GET request must be sent to get the Project ID to later be used in the PUT request.

Note: This service is not supported on the Linux platform.

The GET request should look like the following.

```
GET     /config/v1/project/ProjectLoad
```

The server should respond with something similar to the following.

```
{
  "PROJECT_ID": 3531905431,
  "common.ALLTYPES_NAME": "ProjectLoad",
  "servermain.JOB_TIME_TO_LIVE_SECONDS": 30,
  "servermain.PROJECT_FILENAME": "",
  "servermain.PROJECT_PASSWORD": ""
}
```

To initiate the project load, a PUT request is sent to the server with the absolute path to the project file, the project file password, and the Project ID. If there is no password on the project, that field is not required. Project loading supports SOPF, OPF, and JSON file types. The request should look similar to the following.

```
PUT     /config/v1/project/services/ProjectLoad
```

```
{
  "PROJECT_ID": 3531905431,
  "servermain.PROJECT_FILENAME": "/Absolute/Path/To/MyProject.json",
```

```
  "servermain.PROJECT_PASSWORD": ""
}
```

The server should respond with something similar to the following.

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/services/ProjectLoad/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job. This is done by sending a GET to the job's endpoint. The GET request should look like the following.

```
GET     /config/v1/project/ProjectLoad/jobs/job1
```

Jobs are automatically cleaned up after their wait time has expired. This wait time is configurable. *See the **Job Cleanup** section for more information.*

Service Project Save

Projects can be loaded by interacting with the ProjectSave service on the ProjectSave endpoint. A GET request must be sent to get the Project ID to later be used in the PUT request. The GET request should look similar to the following.

```
GET     /config/v1/project/ProjectSave
```

The server should respond with something similar to the following.

```
{
  "PROJECT_ID": 2401921849,
  "common.ALLTYPES_NAME": "ProjectSave",
  "servermain.JOB_TIME_TO_LIVE_SECONDS": 30,
  "servermain.PROJECT_FILENAME": ""
}
```

To initiate the project save a PUT request is sent with the project file path and name of the file with the extension (SOPF, OPF, or JSON), the password to encrypt it with, and the Project ID. The password property is required for SOPF file and ignored otherwise. The path is relative to the Application Data Folder. The PUT request should look similar to the following.

```
PUT     /config/v1/project/services/ProjectSave
```

```
{
  "PROJECT_ID": 2401921849,
  "servermain.PROJECT_FILENAME": "Projects/MyProject.sopf",
  "servermain.PROJECT_PASSWORD": "MyPassword"
}
```

The server should respond with something similar to the following.

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/services/ProjectSave/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job. This is done by sending a GET to the job's endpoint. The GET request should look like the following.

```
GET     /config/v1/project/ProjectLoad/jobs/job1
```

Jobs are automatically cleaned up after their wait time has expired. This wait time is configurable. *See the Job Cleanup section for more information.*

Config API Service — Response Codes

One of the following response codes may be returned from a REST request. Where possible, the body of the response contains specific error messages to help identify the cause of the error and possible solutions:

- HTTP/1.1 200 OK
- HTTP/1.1 201 Created
- HTTP/1.1 202 Accepted
- HTTP/1.1 207 Multi-Status
- HTTP/1.1 400 Bad Request
- HTTP/1.1 401 Unauthorized
- HTTP/1.1 403 Forbidden
- HTTP/1.1 404 Not Found
- HTTP/1.1 429 Too Many Requests
- HTTP/1.1 500 Internal Server Error
- HTTP/1.1 503 Server Runtime Unavailable
- HTTP/1.1 504 Gateway Timeout
- HTTP/1.1 520 Unknown Error

Consult the Configuration API Service Event Log Messages

Built-In Diagnostics

When communications problems occur, users can utilize both OPC and channel diagnostics to help determine the cause of the issue. These views provide diagnostics on both the server-level and driver-level. Since they may affect performance, users should only utilize diagnostics when debugging or trouble-shooting. For more information, select a link from the list below.

OPC Diagnostics Viewer
Channel Diagnostics

OPC Diagnostics Viewer

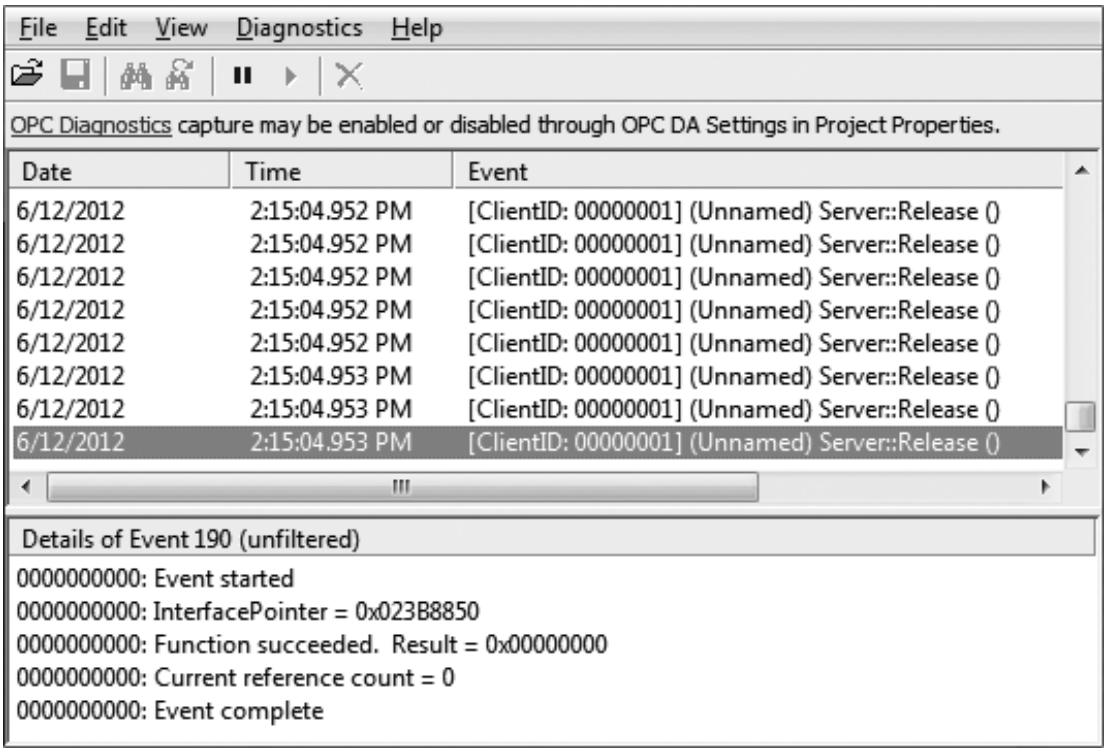
The OPC Diagnostics Viewer provides both a real-time and historical view of OPC events occurring between an OPC client and the server. An event is a method call that a client makes to the server, or a callback that the server makes to a client.

Accessing the OPC Diagnostics Viewer

The OPC Diagnostics Viewer is separate from the main server configuration window. To access the OPC Diagnostics Viewer, click **View | OPC Diagnostics**.

Note: Although the viewer can be accessed when capture is disabled, there are no diagnostics until it is enabled.

*For information on enabling OPC diagnostics, refer to **Project Properties — OPC DA**, **Project Properties — OPC UA Settings**, and **Project Properties — OPC HDA**.*



*For information on the log settings properties, refer to **Settings - Event Log**.*

Live Data Mode

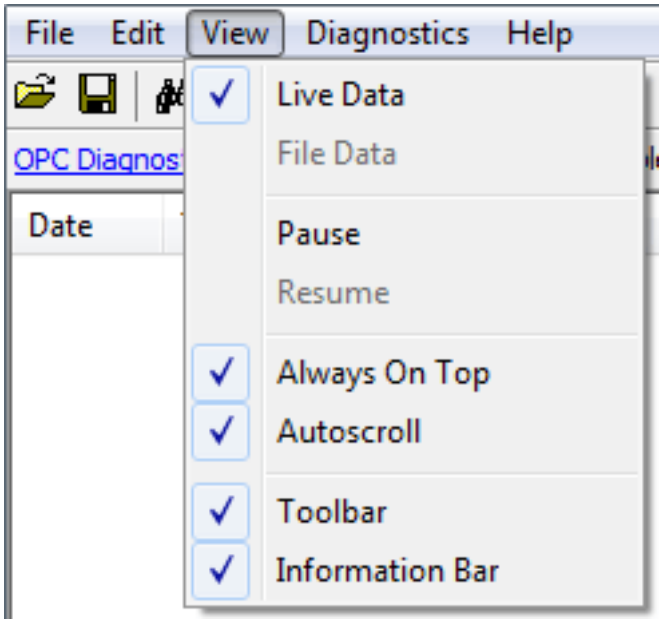
The OPC Diagnostics Viewer opens in Live Data Mode, which displays the persisted OPC Diagnostics data that is currently available from the Event Log. The viewer is updated in real time. To pause the display, click **View | Pause** or select the **Pause** icon. Although data continues to be captured, the display does not update.

To save an OPC Diagnostics file, click **File | Save As** and select **OPC Diagnostic Files (*.opcdiag)**.

File Data Mode

The OPC Diagnostics Viewer can open and display saved OPC Diagnostics files. When a saved file is opened, the viewer switches to File Data Mode and display the name and data from the loaded file. Users can switch between the modes through the View menu. Once a file is closed, the view switches to Live Data, and the File Data view is unavailable until another file is loaded.

View Menu

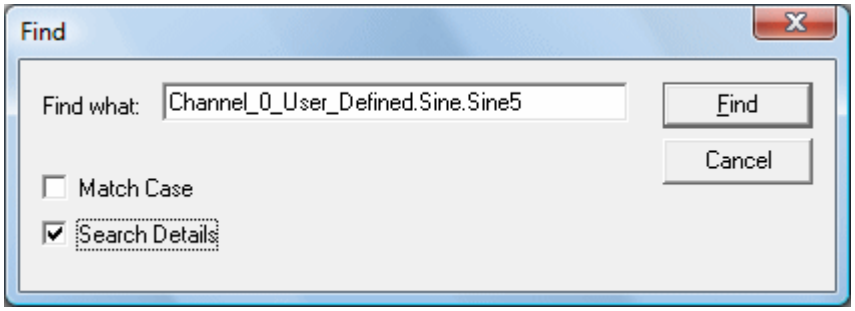


Descriptions of the options are as follows:

- **Live Data** When enabled, this option displays any persisted OPC Diagnostics data that is currently available from the Event Log. The default setting is enabled. For more information, refer to **Live Data Mode**.
- **File Data** When enabled, this option displays data from a saved OPC Diagnostics file. The default setting is disabled. For more information, refer to **File Data Mode**.
- **Always on Top** When enabled, this option forces the OPC Diagnostics window to remain on the top of all other application windows. The default setting is enabled.
- **Autoscroll** When enabled, this option scrolls the display as new events are received to ensure that the most recent event is visible. It turns off when users manually select an event (or when a selection is made by Find/Find Next).
- **Toolbar** When enabled, this option displays a toolbar of icons for quick access to the options available through the File, Edit, and View menus. The default setting is enabled.
- **Information Bar** When enabled, this option displays a bar of information above the OPC Diagnostics data. The default setting is enabled.

Find

This dialog searches the Diagnostics View for key information transferred between the client and server. For example, this search functionality can be used to find all actions on a particular item ID or group name.



Descriptions of the properties are as follows:

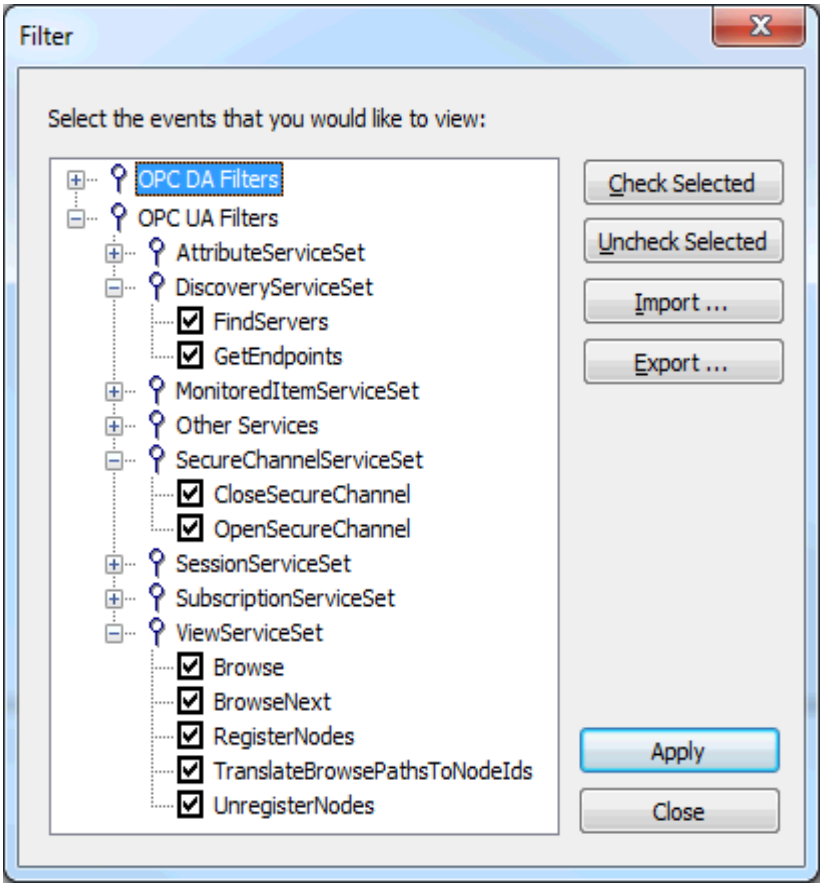
- **Find What** This field specifies the search criteria.
- **Match Case** When enabled, the search criteria is case sensitive.
- **Search Details** When enabled, the search criteria includes details.

Note: When an event or detail with the specified text is found, the line containing the text is highlighted. To perform a Find Next operation (and look for the next occurrence of the specified text), press "F3". When the last occurrence is found, a message box indicates this condition. Users can change the search criteria at any time by pressing "Ctrl+F".

Filter

This dialog specifies which events is visible in the OPC Diagnostics Viewer. For example, most clients make continuous GetStatus calls into the server to determine whether the server is still available. By filtering this event, users can just examine the diagnostics data. The filtering applied is to the view, not to the capture. All event types are captured regardless of the filter settings. Furthermore, because filters can be applied while the dialog is open, settings can be changed and applied independently. Changes may be made without closing and reopening the dialog.

Note: Each method (such as "IOPCCommon" or "GetErrorString") of every OPC Data Access 1.0, 2.0, and 3.0 interface that is supported by the server is available as a filter.



Descriptions of the options are as follows:

- **Check Selected**: When clicked, this button enables all events under the selected item for viewing. All methods for all interfaces are selected by default.
  For more information, refer to OPC DA Events and OPC UA Services.

- **Uncheck Selected** When clicked, this button enables all event types and methods under the selected item.
- **Import** When clicked, this button allows users to select an INI file for import to the Filter.
- **Export** When clicked, this button allows users to export the Filter as an INI file.

Notes:

1. Because the Filter settings are persisted when the OPC Diagnostics Viewer is closed, users can reopen and view the OPC diagnostic files at a later time. Files opened in File Data Mode may be filtered. When a file is saved from the OPC Diagnostics Viewer, only the events that are displayed as a result of the applied filter is saved. If an unfiltered data file is required, users must turn off filtering before saving the file.
2. The server's performance is affected when diagnostic information is captured because it is an additional layer of processing that occurs between the client/server communications. Furthermore, logging OPC Diagnostics in the Extended Datastore Persistence Mode can consume a lot of disk space. The Windows Event Viewer reports any related errors. For information on persistence modes, refer to **Settings - Event Log**.

OPC DA Events

For more information on a specific OPC Diagnostic Event, select a link from the list below.

IClassFactory
Server
IOPCCommon
IOPCServer
IConnectionPointContainer (Server)
IConnectionPoint (Server)
IOPCBrowse
IOPCBrowseServerAddressSpace
IOPCItemProperties
IOPCItemIO
Group
IOPCGroupStateMgt
IOPCGroupStateMgt2
IOPCItemMgt
IOPCItemDeadbandMgt
IOPCItemSamplingMgt
IOPCSyncIO
IOPCSyncIO2
IOPCAsyncIO
IDataObject
IAdviseSink
IAsyncIO2
IAsyncIO3
IConnectionPointContainer (Group)

IConnectionPoint (Group)
IOPCDataCallback
IEnumOPCItemAttributes

IClassFactory

The IClassFactory interface contains several methods intended to deal with an entire class of objects. It is implemented on the class object for a specific class of objects and is identified by a CLSID.

- **QueryInterface**: The client can ask the object whether it supports any outgoing interfaces by calling QueryInterface for IConnectionPointContainer. If the object answers "yes" by handing back a valid pointer, the client knows it can attempt to establish a connection.
- **AddRef**: Increments the reference count for an interface on an object. It should be called for every new copy of a pointer to an interface on a given object.
- **Release**: Decreases the reference count of the interface by 1.
- **CreateInstance**: Creates an uninitialized object.
- **LockServer**: Allows instances to be created quickly when called by the client of a class object to keep a server open in memory.

Server

The client calls CoCreateInstance to create the server object and the initial interface.

- **QueryInterface**: The client can ask the object whether it supports any outgoing interfaces by calling QueryInterface for IConnectionPointContainer. If the object answers "yes" by handing back a valid pointer, the client knows it can attempt to establish a connection.
- **AddRef**: Increments the reference count for an interface on an object. It should be called for every new copy of a pointer to an interface on a given object.
- **Release**: Decreases the reference count of the interface by 1.

IOPCCommon

This interface is used by all OPC server types (DataAccess, Alarm&Event, Historical Data, and so forth). It provides the ability to set and query a Locale ID which would be in effect for the particular client/server session. The actions of one client do not affect other clients.

- **GetErrorString**: Returns the error string for a server specific error code. The expected behavior is that this includes handling of Win32 errors as well (such as RPC errors).
- **GetLocaleID**: Returns the default Locale ID for this server/client session.
- **QueryAvailableLocaleIDs**: Returns the available Locale IDs for this server/client session.
- **SetClientName**: Allows the client to optionally register a client name with the server. This is included primarily for debugging purposes. The recommended behavior is that users set the Node name and EXE name here.
- **SetLocaleID**: Sets the default Locale ID for this server/client session. This Locale ID is used by the GetErrorString method on this interface. The default value for the server should be LOCALE_SYSTEM_DEFAULT.

IOPCServer

This is an OPC server's main interface. The OPC server is registered with the operating system as specified in the Installation and Registration Chapter of this specification.

- **AddGroup**: Adds a group to a server. A group is a logical container for a client to organize and manipulate data items.
- **CreateGroupEnumerator**: Creates various enumerators for the groups provided by the server.

- **GetErrorString**: Returns the error string for a server specific error code.
- **GetGroupByName**: Returns an additional interface pointer when given the name of a private group (created earlier by the same client). Use GetPublicGroupByName to attach to public groups. This function can be used to reconnect to a private group for which all interface pointers have been released.
- **GetStatus**: Returns current status information for the server.
- **RemoveGroup** Deletes the group. A group is not deleted when all the client interfaces are released, since the server itself maintains a reference to the group. The client may still call GetGroupByName after all the interfaces have been released. RemoveGroup() causes the server to release its 'last' reference to the group, which results in the group being deleted.

IConnectionPointContainer (Server)

This interface provides the access to the connection point for IOPCShutdown.

- **EnumConnectionPoints**: Creates an enumerator for the connection points supported between the OPC group and the client. OPCServers must return an enumerator that includes IOPCShutdown. Additional vendor specific callbacks are allowed.
- **FindConnectionPoint**: Finds a particular connection point between the OPC server and the client. OPCServers must support IID_IOPCShutdown. Additional vendor specific callbacks are allowed.

IConnectionPoint (Server)

This interface establishes a call back to the client.

- **Advise**: Establishes an advisory connection between the connection point and the caller's sink object.
- **EnumConnections**: Creates an enumerator object for iteration through the connections that exist to this connection point.
- **GetConnectionInterface**: Returns the IID of the outgoing interface managed by this connection point.
- **GetConnectionPointContainer**: Retrieves the IConnectionPointContainer interface pointer to the connectable object that conceptually owns the connection point.
- **Unadvise**: Terminates an advisory connection previously established through the Advise method.
- **ShutdownRequest** Allows the server to request that all clients disconnect from the server.

IOPCBrowse

IOPCBrowse interface provides improved methods for browsing the server address space and for obtaining the item properties.

- **GetProperties**: Returns an array of OPCITEMPROPERTIES, one for each item ID.
- **Browse**: Browses a single branch of the address space and returns zero or more OPCBROWSEELEMENT structures.

IOPCBrowseServerAddressSpace

This interface provides a way for clients to browse the available data items in the server, giving the user a list of the valid definitions for an item ID. It allows for either flat or hierarchical address spaces and is designed to work well over a network. It also insulates the client from the syntax of a server vendor specific item ID.

- **BrowseAccessPaths**: Provides a way to browse the available AccessPaths for an item ID.
- **BrowseOPCItemIDs**: Returns an IENUMString for a list of item IDs as determined by the passed properties. The position from which the browse is made can be set in ChangeBrowsePosition.
- **ChangeBrowserPosition**: Provides a way to move up, down or to in a hierarchical space.

- **GetItemID**: Provides a way to assemble a fully qualified item ID in a hierarchical space. This is required since the browsing functions return only the components or tokens that make up an item ID and do not return the delimiters used to separate those tokens. Also, at each point one is browsing just the names below the current node (e.g. the units in a cell).
- **QueryOrganization**: Provides a way to determine if the underlying system is inherently flat or hierarchical and how the server may represent the information of the address space to the client. Flat and hierarchical spaces behave somewhat different. If the result is flat, the client knows that there is no need to pass the Branch or Leaf flags to BrowseOPCItem IDs or to call ChangeBrowsePosition.

IOPCItemProperties

This interface can be used to browse the available properties associated with an item ID as well as to read the properties' current values.

- **GetItemProperties**: Returns a list of the current data values for the passed ID codes.
- **LookUpItemIDs**: Returns a list of item IDs for each of the passed ID codes if any are available. These indicate the item ID which could be added to an OPC group and used for more efficient access to the data corresponding to the item properties.
- **QueryAvailableProperties**: Returns a list of ID codes and descriptions for the available properties for this item ID. This list may differ for different item IDs. This list is expected to be relatively stable for a particular item ID, although it could be affected from time to time by changes to the underlying system's configuration. The item ID is passed to this function because servers are allowed to return different sets of properties for different item IDs.

IOPCItemIO

The purpose of this interface is to provide an easy way for basic applications to obtain OPC data.

- **Read**: Reads one or more values, qualities, and timestamps for the items specified. This is functionally similar to the IOPCSyncIO::Read method.
- **WriteVQT**: Writes one or more values, qualities, and timestamps for the items specified. This is functionally similar to the IOPCSyncIO2::WriteVQT except that there is no associated group. If a client attempts to write VQ, VT, or VQT it should expect that the server will write them all or none at all.

Group

The client calls CoCreateInstance to create the server object and the initial interface.

- **QueryInterface**: The client can ask the object whether it supports any outgoing interfaces by calling QueryInterface for IConnectionPointContainer. If the object answers "yes" by handing back a valid pointer, the client knows it can attempt to establish a connection.
- **AddRef**: Increments the reference count for an interface on an object. It should be called for every new copy of a pointer to an interface on a given object.
- **Release**: Decreases the reference count of the interface by 1.

IOPCGroupStateMgt

IOPCGroupStateMgt allows the client to manage the overall state of the group. Primarily, this accounts for changes made to the group's update rate and active state.

- **CloneGroup**: Creates a second copy of a group with a unique name.
- **GetState**: Gets the current state of the group. This function is typically called to obtain the current values of this information prior to calling SetState. This information was all supplied by or returned to the client when the group was created.

- **SetName**: Changes the name of a private group. The name must be unique. The name cannot be changed for public groups. Group names are required to be unique with respect to an individual client to server connection.
- **SetState**: Sets various properties of the group. This represents a new group which is independent of the original group.

IOPCGroupStateMgt2

This interface was added to enhance the existing IOPCGroupStateMgt interface.

- **SetKeepAlive**: Causes the server to provide client callbacks on the subscription when there are no new events to report. Clients can be assured of the health of the server and subscription without resorting to pinging the server with calls to GetStatus().
- **GetKeepAlive**: Returns the currently active keep-alive time for the subscription.

IOPCItemMgt

This interface allows a client to add, remove and control the behavior of items is a group.

- **AddItems**: Adds one or more items to a group. It is acceptable to add the same item to the group more than once, generating a second item with a unique ServerHandle.
- **CreateEnumerator**: Creates an enumerator for the items in the group.
- **RemoveItems**: Removes items from a group. Removing items from a group does not affect the address space of the server or physical device. It indicates whether or not the client is interested in those particular items.
- **SetActiveState**: Sets one or more items in a group to active or inactive. This controls whether or not valid data can be obtained from read cache for those items and whether or not they are included in the IAdvise subscription to the group. Deactivating items does not result in a callback, since by definition callbacks do not occur for inactive items. Activating items generally results in an IAdvise callback at the next UpdateRate period.
- **SetClientHandles**: Changes the client handle for one or more items in a group. In general, it is expected that clients set the client handle when the item is added and not change it later.
- **SetDataTypes**: Changes the requested data type for one or more items in a group. In general, it is expected that clients set the requested data type when the item is added and not change it later.
- **ValidateItems**: Determines if an item is valid and could be added without error. It also returns information about the item such as canonical datatype. It does not affect the group in any way.

IOPCItemDeadbandMgt

Force a callback to IOPCDataCallback::OnDataChange for all active items in the group, whether they have changed or not. Inactive items are not included in the callback. The MaxAge value determines where the data is obtained. There is only one MaxAge value, which determines the MaxAge for all active items in the group. This means some of the values may be obtained from cache while others could be obtained from the device, depending on the "freshness" of the data in the cache.

- **SetItemDeadband**: Overrides the deadband specified for the group for each item.
- **GetItemDeadband**: Gets the deadband values for each of the requested items.
- **ClearItemDeadband**: Clears the individual item PercentDeadband, effectively reverting them back to the deadband value set in the group.

IOPCItemSamplingMgt

This optional interface allows the client to manipulate the rate at which individual items within a group are obtained from the underlying device. It does not affect the group update rate of the callbacks for OnDataChange.

- **SetItemSamplingRate**: Sets the sampling rate on individual items. This overrides the update rate of the group as far as collection from the underlying device is concerned. The update rate associated with individual items does not affect the callback period.
- **GetItemSamplingRate**: Gets the sampling rate on individual items, which was previously set with SetItemSamplingRate.
- **ClearItemSamplngRate**: Clears the sampling rate on individual items, which was previously set with SetItemSamplingRate. The item reverts to the update rate of the group.
- **SetItemBufferEnable**: Requests that the server turns on or off, depending on the value of the Enable property, the buffering of data for the identified items, which are collected for items that have an update rate faster than the group update rate.
- **GetItemBufferEnable**: Queries the current state of the servers buffering for requested items.

IOPCSyncIO

IOPCSyncIO allows a client to perform synchronous read and write operations to a server. The operations run to completion.

- **Read**: Reads the value, quality and timestamp information for one or more items in a group. The function runs to completion before returning. The data can be read from cache in which case it should be accurate to within the UpdateRate and percent deadband of the group. The data can be read from the device, in which case an actual read of the physical device must be performed. The exact implementation of cache and device reads are not defined by the specification.
- **Write**: Writes values to one or more items in a group. The function runs to completion. The values are written to the device, meaning that the function should not return until it verifies that the device has actually accepted or rejected the data. Writes are not affected by the active state of the group or item.

IOPCSyncIO2

This interface was added to enhance the existing IOPCSyncIO interface.

- **ReadMaxAge**: Reads one or more values, qualities and timestamps for the items specified. This is functionally similar to the OPCSyncIO::Read method except no source is specified (device or cache). The server determines whether the information is obtained from the device or cache. This decision is based on the MaxAge property. If the information in the cache is within the MaxAge, the data is obtained from the cache; otherwise, the server must access the device for the requested information.
- **WriteVQT**: Writes one or more values, qualities and timestamps for the items specified. This is functionally similar to the IOPCSyncIO::Write except that Quality and Timestamp may be written. If a client attempts to write VQ, VT or VQT it should expect that the server will write to all or none.

IOPCAsyncIO

IOPCAsyncIO allows a client to perform asynchronous read and write operations to a server. The operations are queued and the function returns immediately so that the client can continue to run. Each operation is treated as a transaction and is associated with a Transaction ID. As the operations are completed, a callback is made to the IAdvise Sink in the client (if one is established). The information in the callback indicates the Transaction ID and the error results. By convention, 0 is an invalid Transaction ID.

- **Cancel**: Requests that the server cancel an outstanding transaction.
- **Read**: Reads one or more items in a group. The results are returned via the IAdvise Sink connection established through the IDataObject. For cache reads the data is only valid if both the group and the item are active. Device reads are not affected by the active state of the group or item.

- **Refresh**: Forces a callback for all active items in the group, whether they have changed or not. Inactive items are not included in the callback.
- **Write**: Writes one or more items in a group. The results are returned via the IAdviseSink connection established through the IDataObject.

IDataObject

IDataObject is implemented on the OPCGroup rather than on the individual items. This allows the creation of an Advise connection between the client and the group using the OPC Data Stream Formats for the efficient data transfer.

- **DAdvise**: Creates a connection for a particular stream format between the OPC group and the client.
- **DUnadvise**: Terminates a connection between the OPC group and the client.

IAdviseSink

The client only has to provide a full implementation of OnDataChange.

- **OnDataChange**: This method is provided by the client to handle notifications from the OPC group for exception based data changes, Async reads and Refreshes and Async Write Complete.

IAsyncIO2

This interface is similar to IOPCAsync(OPC 1.0) and is intended to replace IOPCAsyncIO. It was added in OPC 2.05.

- **Cancel2**: Requests that the server cancel an outstanding transaction.
- **GetEnable**: Retrieves the last Callback Enable value set with SetEnable.
- **Read**: Reads one or more items in a group. The results are returned via the client's IOPCDataCallback connection established through the server's IConnectionPointContainer. Reads are from device and are not affected by the active state of the group or item.
- **Refresh2**: Forces a callback to IOPCDataCallback::OnDataChange for all active items in the group, whether they have changed or not. Inactive items are not included in the callback.
- **SetEnable**: Controls the operation of OnDataChange. Setting Enable to False disables any OnDataChange callbacks with a transaction ID of 0 (not the result of a Refresh). The initial value of this variable when the group is created is True; OnDataChange callbacks are enabled by default.
- **Write**: Writes one or more items in a group. The results are returned via the client's IOPCDataCallback connection established through the server's IConnectionPointContainer.

IAsyncIO3

This interface was added to enhance the existing IOPCAsyncIO2 interface.

- **ReadMaxAge**: Reads one or more values, qualities and timestamps for the items specified. This is functionally similar to the OPCSyncIO::Read method except it is asynchronous and no source is specified (device or cache). The server determines whether the information is obtained from the device or cache. This decision is based on the MaxAge property. If the information in the cache is within the MaxAge, the data is obtained from the cache; otherwise, the server must access the device for the requested information.
- **WriteVQT**: Writes one or more values, qualities and timestamps for the items specified. The results are returned via the client's IOPCDataCallback connection established through the server's IConnectionPointContainer. This is functionally similar to the IOPCAsyncIO2::Write except that Quality and Timestamp may be written. If a client attempts to write VQ, VT or VQT it should expect that the server will write them all or none at all.

- **RefreshMaxAge**: Forces a callback to IOPCDataCallback::OnDataChange for all active items in the group, whether or not they have changed. Inactive items are not included in the callback. The MaxAge value determines where the data is obtained. There is only one MaxAge value, which determines the MaxAge for all active items in the group. This means some of the values may be obtained from cache while others can be obtained from the device, depending on the type of the data in the cache.

IConnectionPointContainer (Group)

This interface provides functionality similar to the IDataObject but is easier to implement and to understand. It also provides the functionality missing from the IDataObject interface. The client must use the new IOPCAsyncIO2 interface to communicate via connections established with this interface. The old IOPCAsnyc continues to communicate via IDataObject connections as in the past.

- **EnumConnectionPoints**: Creates an enumerator for the connection points supported between the OPC group and the client.
- **FindConnectionPoint**: Finds a particular connection point between the OPC group and the client.

IConnectionPoint (Group)

This interface establishes a call back to the client.

- **Advise**: Establishes an advisory connection between the connection point and the caller's sink object.
- **EnumConnections**: Creates an enumerator object for iteration through the connections that exist to this connection point.
- **GetConnectionInterface**: Returns the IID of the outgoing interface managed by this connection point.
- **GetConnectionPointContainer**: Retrieves the IConnectionPointContainer interface pointer to the connectable object that conceptually owns the connection point.
- **Unadvise**: Terminates an advisory connection previously established through the Advise method.

IOPCDataCallback

To use connection points, the client must create an object that supports both the IUnknown and IOPCDataCallback interface.

- **OnDataChange**: This method is provided by the client to handle notifications from the OPC group for exception based data changes and Refreshes.
- **OnReadComplete**: This method is provided by the client to handle notifications from the OPC group on completion of Async reads.
- **OnWriteComplete**: This method is provided by the client to handle notifications from the OPC group on completion of AsyncIO2 Writes.
- **OnCancelComplete**: This method is provided by the client to handle notifications from the OPC group on completion of Async cancel.

IEnumOPCItemAttributes

IEnumOPCItemAttributes allows clients to find out the contents of a group and the attributes of those items. Most of the returned information is either supplied by or returned to the client at the time it called AddItem.

- **Clone**: Creates a second copy of the enumerator. The new enumerator is initially in the same state as the current enumerator.
- **Next**: Fetches the next 'celt' items from the group.
- **Reset**: Resets the enumerator back to the first item.
- **Skip**: Skips over the next 'celt' attributes.

For more information on the general principles of connection points, refer to Microsoft documentation.

OPC UA Services

For more information on a specific OPC Diagnostic Event, select a link from the list below.

AttributeServiceSet
DiscoveryServiceSet
MonitoredItemServiceSet
OtherServices
SecureChannelServiceSet
SessionServiceSet
SubscriptionServiceSet
ViewServiceSet

AttributeServiceSet

This service set provides services to access attributes that are part of nodes.

- **Read**: This service is used to read one or more attributes of one or more nodes. For constructed attribute values whose elements are indexed, such as an array, this service allows clients to read the entire set of indexed values as a composite, to read individual elements or to read ranges of elements of the composite.
- **Write**: This service is used to write values to one or more attributes of one or more nodes. For constructed attribute values whose elements are indexed, such as an array, this service allows clients to write the entire set of indexed values as a composite, to write individual elements or to write ranges of elements of the composite.

DiscoveryServiceSet

This service set defines services used to discover the endpoints implemented by a server and to read the security configuration for those endpoints.

- **FindServers**: This service returns the servers known to a server or discovery server.
- **GetEndpoints**: This service returns the endpoints supported by a server and all of the configuration information required to establish a secure channel and session.

MonitoredItemServiceSet

This service set allows clients to define monitored items to subscribe to data and events. Each monitored item identifies the item to be monitored and the subscription to use to send notifications. The item to be monitored may be any node attribute.

- **CreateMonitoredItems**: This service is used to create and add one or more MonitoredItems to a Subscription. A MonitoredItem is deleted automatically by the server when the Subscription is deleted.
- **DeleteMonitoredItems**: This service is used to remove one or more MonitoredItems of a Subscription. When a MonitoredItem is deleted, its triggered item links are also deleted.
- **ModifyMonitoredItems**: This service is used to modify MonitoredItems of a Subscription. Changes to the MonitoredItem settings are immediately applied by the server.
- **SetMonitoringMode**: This service is used to set the monitoring mode for one or more MonitoredItems of a Subscription. Setting the mode to disabled causes all queued notifications to be deleted.

- **SetTriggering**: This service is used to create and delete triggering links for a triggering item. Triggered items and their links cause a monitored item to report samples when their monitoring mode doesn't allow for that by default.

OtherServices

OtherServices represents miscellaneous services and notifications.

- **ServiceFault**: This response is provided any time a service fails.
- **Unsupported**: These services are not supported by this server.

SecureChannelServiceSet

This service set defines services used to open a communication channel that ensures the confidentiality and integrity of all messages exchanged with the server.

- **CloseSecureChannel**: This service is used to terminate a SecureChannel.
- **OpenSecureChannel**: This services is used to open or renew a SecureChannel that can be used to ensure confidentiality and integrity for message exchange during a session. This service requires the communication stack to apply the various security algorithms to the messages as they are sent and received.

SessionServiceSet

This service set defines services for an application layer connection establishment in the context of a session.

- **ActivateSession**: This service is used by the client to specify the identity of the user associated with the session.
- **Cancel**: This service is used to cancel any outstanding service requests. Successfully cancelled service requests shall respond with Bad_RequestCancelledByClient ServiceFaults.
- **CloseSession**: This service is used to terminate a session.
- **CreateSession**: This service is used by the client to create a Session and the server returns two values which uniquely identify the Session. The first value is the sessionId which is used to identify the Session in the Server's AddressSpace. The second is the authenticationToken which is used to associate an incoming request with a Session.

SubscriptionServiceSet

Subscriptions are used to report notifications from MonitoredItems to a client.

- **CreateSubscription**: This service is used to create a subscription. Subscriptions monitor a set of MonitoredItems for Notifications and return them to the client in response to Publish requests.
- **DeleteSubscriptions**: This service is invoked to delete one or more subscriptions that belong to the client's session. Successful completion of this service causes all MonitoredItems that use the Subscription to be deleted.
- **ModifySubscription**: This service is used to modify a subscription.
- **Publish**: This service is used for two purposes. First, it is used to acknowledge the receipt of NotificationMessages for one or more Subscriptions. Second, it is used to request the server to return a NotificationMessage or a keep-alive message. Since Publish requests are not directed to a specific Subscription, they may be used by any Subscription.

- **Republish**: This service requests the Subscription to republish a NotificationMessage from its retransmission queue.
- **SetPublishingMode**: This service is used to enable or disable sending of notifications on one or more subscriptions.
- **TransferSubscriptions**: This service is used to transfer a subscription and its MonitoredItems from one Session to another.

ViewServiceSet

Clients use the browse services of this service set to navigate through the AddressSpace.

- **Browse**: This service is used to discover the References of a specified Node. The browse service also supports a primitive filtering capability.
- **BrowseNext**: This service is used to request the next set of Browse or BrowseNext response information that is too large to be sent in a single response. "Too large" in this context means that the server is not able to return a larger response or that the number of results to return exceeds the maximum number of results to return that was specified by the client in the original browse request.
- **RegisterNodes**: This service can be used by clients to register the Nodes that they know they will access repeatedly (e.g. Write, Read). It allows Servers to set up anything needed so that the access operations will be more efficient.
- **TranslateBrowsePathsToNodeIds**: This service is used to request that the server translates one or more browse paths to NodeIds. Each browse path is constructed of a starting Node and a RelativePath. The specified starting Node identifies the Node from which the RelativePath is based. The RelativePath contains a sequence of ReferenceTypes and BrowseNames.
- **UnregisterNodes**: This service is used to unregister NodeIds that have been obtained via the RegisterNodes service.

For more information on the general principles of connection points, refer to Microsoft documentation.

Communication Diagnostics

The server's diagnostic features provide real-time information on the communication driver's performance. All read and write operations can be viewed in the Diagnostics Viewer or tracked directly in the OPC client application with built-in Diagnostics tags. The Diagnostic Viewer also provides a real-time protocol view, which is useful when making changes to key communication parameter settings (such as baud rate, parity, or device IDs). The changes' effects are displayed in real-time. Once the correct communication and device settings are set, the data exchange with the device is visible.

Enabling Communication Diagnostics

To enable Communication Diagnostics, right-click on the channel in the Project View and click **Properties | Enable Diagnostics**. Alternatively, double-click on the channel and select **Enable Diagnostics**. Users may enable diagnostics after channel creation.

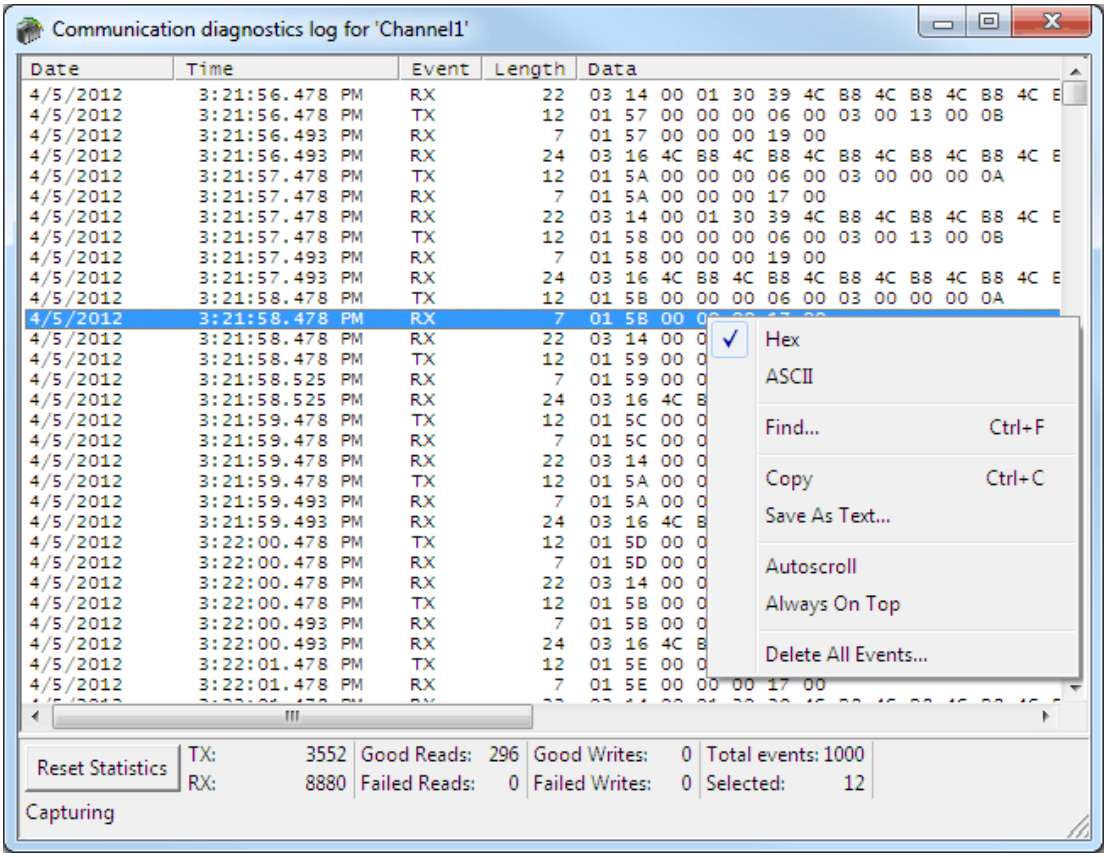
See Also: *Channel Properties — General*

Accessing the Communication Diagnostics Viewer

To access the Communication Diagnostics Viewer, right-click on the channel or device in the Project View and select **Diagnostics**. Alternatively, select the channel or device and click **View** | **Communication Diagnostics**. The Communication Diagnostics Viewer operates in a mode-less form that allows it to exist while other dialogs in the server are open. Once the viewer is open, it should begin capturing the real-time

protocol data. If communications are occurring properly, there is a stream of communications messages between the server and the device. Users should be able to view the TX and RX events, as well as the Total Event count.

Note: Although the Communication Diagnostics Viewer can be opened when capture is disabled, there are no diagnostics until it is enabled. When enabled, the viewer displays "Capturing". When disabled, the viewer displays "Diagnostics capture disabled".



Reset Statistics

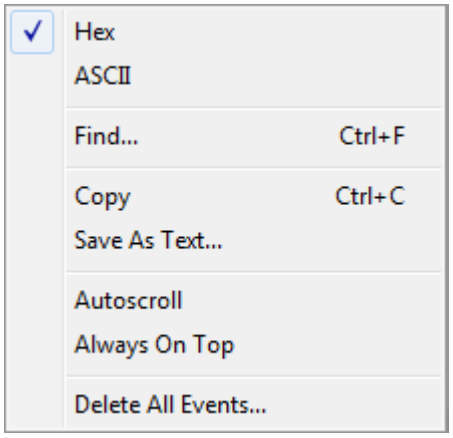
Clicking Reset Statistics sets the counts for TX, RX, Good Reads, Failed Reads, Good Writes, and Failed Writes to zero. Total Events are not set to zero because it specifies the actual number of events in the viewer.

For information on the log settings, refer to ***Settings - Event Log****.*

Accessing the Context Menu

If communications do not appear to be working normally, users can access the channel properties and modify the communications parameters. The Diagnostic Window remains displayed even after the channel properties are displayed, allowing users to change the properties and monitor their effect. The Diagnostic Window must be displayed before any dialogs are accessed.

If a communications problem persists, right-click in the Diagnostic Window to invoke the context menu. Then, use the available selections to tailor the Diagnostic Window's operation.

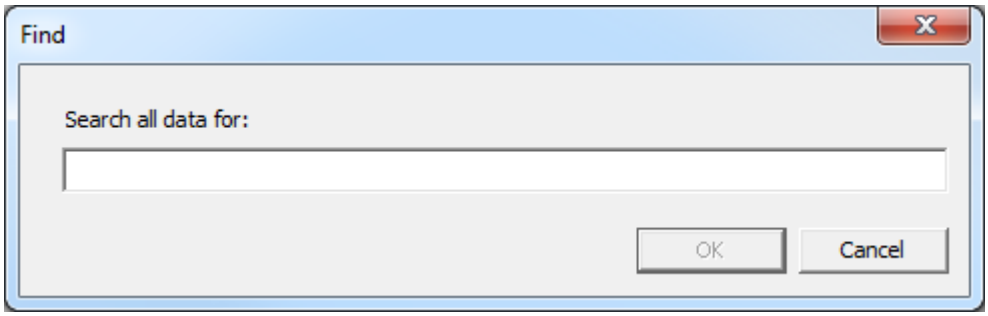


Descriptions of the options are as follows:

- **Hex** When enabled, the TX/RX details are formatted using hexadecimal notation.
- **ASCII** When enabled, the TX/RX details are formatted using ASCII notation.
- **Find** This option invokes a dialog for entering a search string to be applied to the event details. For more information, refer to **Find**.
- **Copy**: This option formats the protocol capture buffer's contents as text for easy "cut and paste" into an email or fax message. This information helps Technical Support analyze and diagnose many communications issues.
- **Save as Text File**: This option saves all the events in the view to a specified file name (as text).
- **Autoscroll**: This option scrolls the display as new events are received to ensure that the most recent one is visible. It is turned off when users manually select an event (or when a selection is made by Find/Find Next).
- **Always on Top**: This option forces the Diagnostics Window to remain on the top of all other application windows. This is the default setting.
- **Delete All Events:** This option clears the log being maintained by the Event Log and results in the deletion of data.

Find

This dialog searches the Diagnostics View for key information transferred between the client and server.



Description of the property is as follows:

- **Search all data for** This field specifies the search criteria.

Note: When an event or detail with the specified text is found, the line containing the text is highlighted. To perform a Find Next operation (and look for the next occurrence of the specified text), press "F3". When the last occurrence is found, a message box is displayed indicating this condition. Users can change the search criteria at any time by pressing "Ctrl+F".

Event Log Messages

The following information concerns messages posted to the Event Log pane in the main user interface. Consult the server help on filtering and sorting the Event Log detail view. Server help contains many common messages, so should also be searched. Generally, the type of message (informational, warning) and troubleshooting information is provided whenever possible.

Server Summary Information

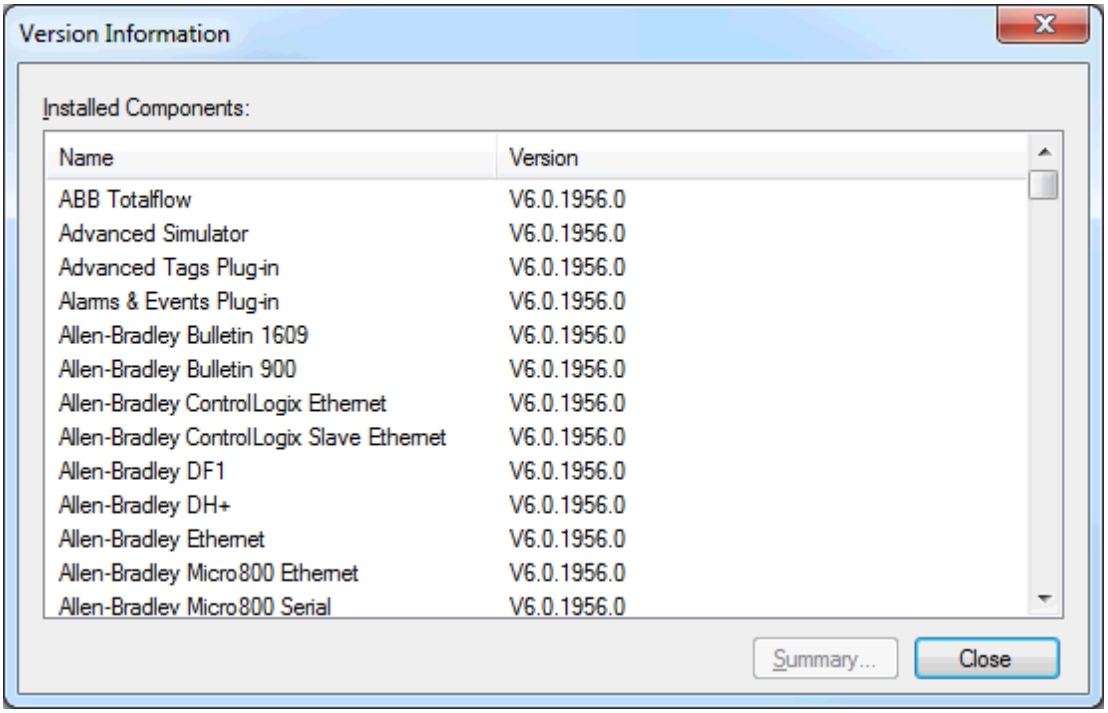
The server provides basic summary information about itself and any drivers and plug-ins that are currently installed.

About the Server

The server version is readily available for review and provides a way to find driver-specific information. To access, click **Help | Support Information** in the server Configuration. To display the version information of all installed components, click **Versions**.

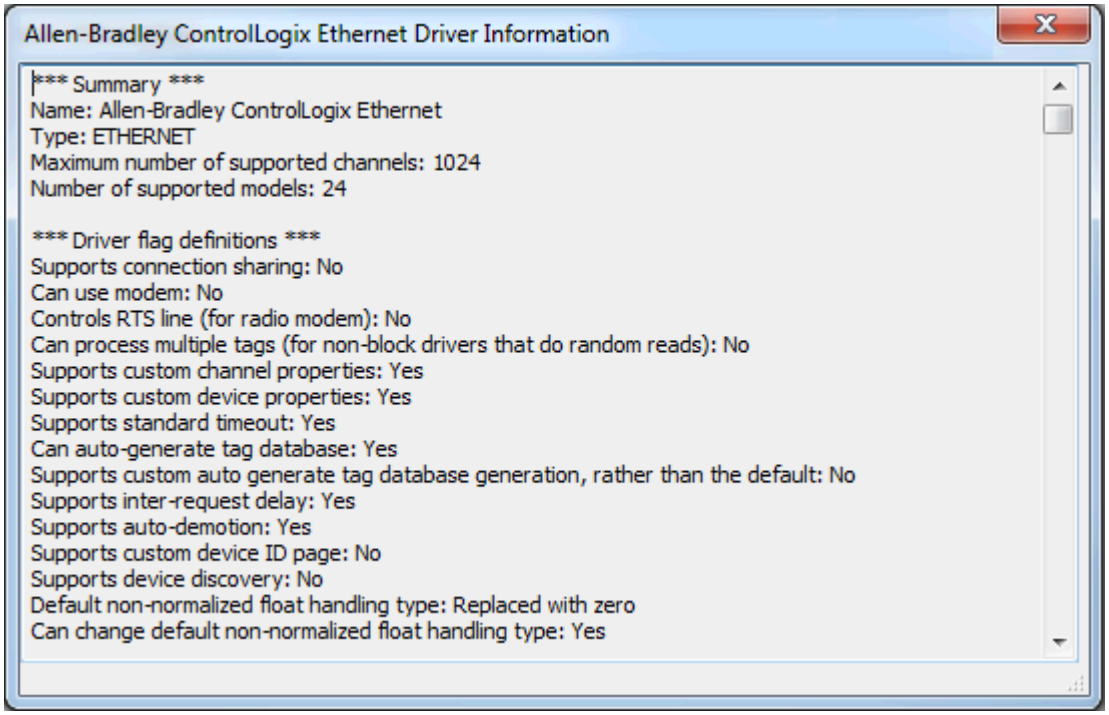
Component Version Information

The Version Information window displays all installed drivers and plug-ins along with their version numbers. For driver-specific information, select a component and click **Summary**.



Driver Information

The Driver Information window provides a summary of the driver's default settings. For example, each driver displays its maximum number of supported channels.



Descriptions of the sections of information available is as follows:

Summary provides the driver name and type, the maximum number of supported channels, and the number of models in the driver.

COMM Defaults displays the driver's default settings, which may or may not match the settings of the device being configured.

Driver flag definitions displays the driver library functions and indicates whether they have been enabled in the driver.

Model Information displays device-specific addressing and features. It lists the name for each supported model in addition to its addressing values and other features.

The <name> device driver was not found or could not be loaded.

Error Type:

Error

Possible Cause:

1. If the project has been moved from one PC to another, the required drivers may have not been installed yet.
2. The specified driver may have been removed from the installed server.
3. The specified driver may be the wrong version for the installed server version.

Possible Solution:

1. Re-run the server install and add the required drivers.
2. Re-run the server install and re-install the specified drivers.
3. Ensure that a driver has not been placed in the installed server directory (which is out of sync with the server version).

Unable to load the '<name>' driver because more than one copy exists ('<name>' and '<name>'). Remove the conflicting driver and restart the application.

Error Type:
Error

Possible Cause:
Multiple versions of the driver DLL exist in the driver's folder in the server.

Possible Solution:
1. Re-run the server install and re-install the specified drivers.
2. Contact Technical support and verify the correct version. Remove the driver that is invalid and restart the server and load the project.

Invalid project file.

Error Type:
Error

Failed to open modem line '<line>' [TAPI error = <code>].

Error Type:
Error

Possible Cause:
TAPI attempted to open the modem line for the server and encountered an error.

Possible Solution:
Correct the condition for the specified error. Then re-attempt to open the modem line.

Unable to add channel due to driver-level failure.

Error Type:
Error

Possible Cause:
Attempt failed due to issues in the driver.

Possible Solution:
Refer to the additional messages about the driver error and correct related issues.

Unable to add device due to driver-level failure.

Error Type:
Error

Possible Cause:
Attempt failed due to issues in the driver.

Possible Solution:
Refer to the additional messages about the driver error and correct related issues.

Version mismatch.

Error Type:
Error

Invalid XML document:

Error Type:
Error

Possible Cause:
The server is unable to parse the specified XML file.

Possible Solution:
If the server project was edited using a third-party XML editor, verify that the format is correct via the schemas for the server and drivers.

Unable to load project <name>:

Error Type:
Error

Possible Cause:
The project was created in a server version that is not compatible with the version trying to load it.

Possible Solution:
Typically this happens when a project was created in a newer version of the server and it is being opened in an older version.

Note:
Every attempt is made to ensure backwards compatibility in the server so that projects created in older versions may be loaded in newer versions. However, since new versions of the server and driver may have properties and configurations that do not exist in older versions, it may not be possible to open or load a older project in a newer version.

Unable to backup project file to '<path>' [<reason>]. The save operation has been aborted. Verify the destination file is not locked and has read/write access. To continue to save this project without a backup, deselect the backup option under Tools | Options | General and re-save the project.

Error Type:
Error

Possible Cause:

1. The destination file may be not locked by another application.
2. The destination file or the folder where it is located does not allow read/write access.

Possible Solution:

1. Ensure that the destination file is not locked by another application, unlock the file, or close the application.
2. Ensure that the destination file and with the folder where it is located allow read and write access.

<feature name> was not found or could not be loaded.

Error Type:
Error

Possible Cause:
The feature is not installed or is not in the expected location.

Possible Solution:
Re-run the server install and select the specified feature for installation.

Unable to save project file <name>:

Error Type:
Error

Device discovery has exceeded <count> maximum allowed devices. Limit the discovery range and try again.

Error Type:
Error

<feature name> is required to load this project.

Error Type:
Error

The current language does not support loading XML projects. To load XML projects, change the product language selection to English in Server Administration.

Error Type:
Error

Possible Cause:
Loading XML projects file allowed only in English environment.

Possible Solution:
Change the product language selection to English in Server Administration and try again.

Unable to load the project due to a missing object. | Object = '<object>'.

Error Type:
Error

Possible Cause:
Editing the JSON project file may have left it in an invalid state.

Possible Solution:
Revert any changes made to the JSON project file.

Invalid Model encountered while trying to load the project. | Device = '<device>'.

Error Type:
Error

Possible Cause:
The specified device has a model that is not supported in this version of the server.

Possible Solution:
Open this project with a newer version of the server.

Cannot add device. A duplicate device may already exist in this channel.

Error Type:
Error

Auto-generated tag '<tag>' already exists and will not be overwritten.

Error Type:
Warning

Possible Cause:
Although the server is regenerating tags for the tag database, it has been set not to overwrite tags that already exist.

Possible Solution:

If this is not the desired action, change the setting of the "On Duplicate Tag" property for the device.

Unable to generate a tag database for device '<device>'. The device is not responding.

Error Type:

Warning

Possible Cause:

1. The device did not respond to the communications request.
2. The specified device is not on, not connected, or in error.

Possible Solution:

1. Verify that the device is powered on and that the PC is on (so that the server can connect to it).
2. Verify that all cabling is correct.
3. Verify that the device IDs are correct.
4. Correct the device failure and retry the tag generation.

Unable to generate a tag database for device '<device>':

Error Type:

Warning

Possible Cause:

The specified device is not on, not connected, or in error.

Possible Solution:

Correct the device failure and retry the tag generation.

Auto generation produced too many overwrites, stopped posting error messages.

Error Type:

Warning

Possible Cause:

1. To keep from filling the error log, the server has stopped posting error messages on tags that cannot be overwritten during automatic tag generation.
2. Reduce the scope of the automatic tag generation or eliminate problematic tags.

Failed to add tag '<tag>' because the address is too long. The maximum address length is <number>.

Error Type:
Warning

Line '<line>' is already in use.

Error Type:
Warning

Possible Cause:
The target modem line is already open, likely because it is in use by another application.

Possible Solution:
Find the application holding the modem open and close or release it.

Hardware error on line '<line>'.

Error Type:
Warning

Possible Cause:
A hardware error was returned after a request was made for a tag in a device connected to the modem.

Possible Solution:
Disable data collection on the device. Enable it after the modem connects to the destination modem.

Note:
The error occurs on first scan and is not repeated.

No comm handle provided on connect for line '<line>'.

Error Type:
Warning

Possible Cause:
An attempt was made to connect to the modem line with no specified COMM handle.

Possible Solution:
Verify the modem is installed and initialized correctly.

Unable to dial on line '<line>'.

Error Type:
Warning

Possible Cause:
The modem is not in a state that allows dialing.

Possible Solution:

To dial a number, the line must be idle. Monitor the _Mode Modem tag and dial when it indicates an idle state.

Unable to use network adapter '<adapter>' on channel '<name>'. Using default network adapter.

Error Type:
Warning

Possible Cause:
The network adapter specified in the project does not exist on this PC. The server uses the default network adapter.

Possible Solution:
Select the network adapter to use for the PC and save the project.

See Also:
Channel Properties - Network Interface

Rejecting attempt to change model type on a referenced device '<channel device>'.

Error Type:
Warning

TAPI line initialization failed: <code>.

Error Type:
Warning

Possible Cause:
The telephony service is not required to be running for the Runtime to start. When the service is disabled and a serial driver is added to the project, this error message is reported.

Possible Solution:

1. If modem communication is not used, no action is required.
2. If modem communications are required, the telephony service must be started on the PC.

Validation error on '<tag>': <error>.

Error Type:
Warning

Possible Cause:
An attempt was made to set invalid parameters on the specified tag.

Unable to load driver DLL '<name>'.

Error Type:

Warning

Possible Cause:
The specified driver could not be loaded when the project started.

Possible Solution:

1. Verify the version of the installed driver. Check the website to see if the driver version is correct for the server version installed.
2. If the driver corrupted, delete it and re-run the server install.

Note:
This problem is usually due to corrupted driver DLLs or drivers that are not compatible with the server version.

Validation error on '<tag>': Invalid scaling parameters.

Error Type:
Warning

Possible Cause:
An attempt was made to set invalid scaling parameters on the specified tag.

See Also:
Tag Properties - Scaling

Unable to apply modem configuration on line '<line>'.

Error Type:
Warning

Possible Cause:
TAPI Manager was unable to apply configuration changes to the server.

Possible Solution:

1. Verify the cabling to the modem.
2. Verify that the modem is set to accept configuration changes.
3. Verify that the modem is not being used by another application.

Device '<device>' has been automatically demoted.

Error Type:
Warning

Possible Cause:
Communications with the specified device failed. The device has been demoted from the poll cycle.

Possible Solution:

1. If the device fails to reconnect, investigate the reason behind the communications loss and correct it.
2. To stop the device from being demoted, disable Auto-Demotion.

See Also:
Auto-Demotion

<Source>: Invalid Ethernet encapsulation IP '<address>'.

Error Type:
Warning

Possible Cause:
The IP address specified for a device on an Ethernet encapsulated channel is not a valid IP address.

Possible Solution:
Correct the IP in the XML file and re-load the project.

Note:
This error can occur when loading XML formatted projects that were created or edited with third-party XML software.

The '<product>' driver does not currently support XML persistence. Save using the default file format.

Error Type:
Warning

Possible Cause:
The specified driver does not support XML formatting.

Possible Solution:
Save the project in .opf format.

Unable to load plug-in DLL '<name>'.

Error Type:
Warning

Possible Cause:
The specified plug-in could not be loaded when the project started.

Possible Solution:

1. Verify the version of the plug-in installed. Check the website to see if the plug-in version is compatible with the server installed. If not, correct the server or re-run the server install.
2. If the plug-in is corrupt, delete it and then re-run the server install.

Note:

This problem is usually due to corrupted plug-in DLLs or plug-ins that are not compatible with the server version.

The time zone set for '<device>' is '<zone>'. This is not a valid time zone for the system. Defaulting the time zone to '<zone>'.

Error Type:
Warning

Unable to load driver DLL '<name>'. Reason:

Error Type:
Warning

Possible Cause:
The specified plug-in could not be loaded when the project started.

Possible Solution:

1. Verify the version of the plug-in installed. Check the website to see if the plug-in version is compatible with the server installed. If not, correct the server or re-run the server install.
2. If the plug-in is corrupt, delete it and then re-run the server install.

Unable to load plug-in DLL '<name>'. Reason:

Error Type:
Warning

Possible Cause:
The specified plug-in could not be loaded when the project started.

Possible Solution:

1. Verify the version of the plug-in installed. Check the website to see if the plug-in version is compatible with the server installed. If not, correct the server or re-run the server install.
2. If the plug-in is corrupt, delete it and then re-run the server install.

Channel requires at least one number in its phonebook for automatic dialing. | Channel = '<channel>'.

Error Type:
Warning

Possible Cause:
The Auto-Dial property is set to Enable and there are no entries in the phonebook.

Possible Solution:
If auto-dialing is desired, add a phone number entry to the phonebook. If auto-dialing is not desired, disable Auto-Dial.

Channel requires Auto-Dial enabled and at least one number in its phone-book to use a shared modem connection. | Channel = '<channel>'.

Error Type:
Warning

Possible Cause:
Channel shares a modem with one or more existing channels and does not have Auto-Dial enabled or a phone number for auto-dialing.

Possible Solution:

1. Enable Auto-Dial on the reported channel.
2. Add a phone number to the phonebook of the reported channel.

The specified network adapter is invalid on channel '%1' | Adapter = '%2'.

Error Type:
Warning

Possible Cause:
The network adapter specified in the project does not exist on this PC.

Possible Solution:
Select the network adapter to use for the PC and save the project.

See Also:
Channel Properties - Network Interface

No tags were created by the tag generation request. See the event log for more information.

Error Type:
Warning

Possible Cause:
The driver produced no tag information but declined to provide a reason why.

Possible Solution:
Event log may contain information that will help troubleshoot the issue.

TAPI configuration has changed, reinitializing...

Error Type:
Informational

<Product> device driver loaded successfully.

Error Type:

Informational

Starting <name> device driver.

Error Type:
Informational

Stopping <name> device driver.

Error Type:
Informational

Dialing '<number>' on line '<modem>'.

Error Type:
Informational

Line '<modem>' disconnected.

Error Type:
Informational

Dialing on line '<modem>' canceled by user.

Error Type:
Informational

Line '<modem>' connected at <rate> baud.

Error Type:
Informational

Remote line is busy on '<modem>'.

Error Type:
Informational

Remote line is not answering on '<modem>'.

Error Type:
Informational

No dial tone on '<modem>'.

Error Type:
Informational

The phone number is invalid (<number>).

Error Type:
Informational

Dialing aborted on '<modem>'.

Error Type:
Informational

Line dropped at remote site on '<modem>'.

Error Type:
Informational

Incoming call detected on line '<modem>'.

Error Type:
Informational

Modem line opened: '<modem>'.

Error Type:
Informational

Modem line closed: '<modem>'.

Error Type:
Informational

<Product> device driver unloaded from memory.

Error Type:
Informational

Line '<modem>' connected.

Error Type:
Informational

Simulation mode is enabled on device '<device>'.

Error Type:
Informational

Simulation mode is disabled on device '<device>'.

Error Type:
Informational

Attempting to automatically generate tags for device '<device>'.

Error Type:
Informational

Completed automatic tag generation for device '<device>'.

Error Type:

Informational

Initiating disconnect on modem line '<modem>'.

Error Type:
Informational

A client application has enabled auto-demotion on device '<device>'.

Error Type:
Informational

Possible Cause:
A client application connected to the server has enabled or disabled Auto Demotion on the specified device.

Possible Solution:
To restrict the client application from doing this, disable its ability to write to system-level tags through the User Manager.

See Also:
User Manager

Data collection is enabled on device '<device>'.

Error Type:
Informational

Data collection is disabled on device '<device>'.

Error Type:
Informational

Object type '<name>' not allowed in project.

Error Type:
Informational

Created backup of project '<name>' to '<path>'.

Error Type:
Informational

Device '<device>' has been auto-promoted to determine if communications can be re-established.

Error Type:
Informational

Failed to load library: <name>.

Error Type:
Informational

Failed to read build manifest resource: <name>.

Error Type:
Informational

The project file was created with a more recent version of this software.

Error Type:
Informational

A client application has disabled auto-demotion on device '<device>'.

Error Type:
Informational

Phone number priority has changed. | Phone Number Name = '<name>', Updated Priority = '<priority>'.

Error Type:
Informational

Tag generation results for device '<device>'. | Tags created = <count>.

Error Type:
Informational

Tag generation results for device '<device>'. | Tags created = <count>, Tags overwritten = <count>.

Error Type:
Informational

Tag generation results for device '<device>'. | Tags created = <count>, Tags not overwritten = <count>.

Error Type:
Informational

Access to object denied. | User = '<account>', Object = '<object path>', Permission =

Error Type:
Security

User moved from user group. | User = '<name>', Old group = '<name>', New group = '<name>'.

Error Type:
Security

User group has been created. | Group = '<name>'.

Error Type:
Security

User added to user group. | User = '<name>', Group = '<name>'.

Error Type:
Security

User group has been renamed. | Old name = '<name>', New name = '<name>'.

Error Type:
Security

Permissions definition has changed on user group. | Group = '<name>'.

Error Type:
Security

User has been renamed. | Old name = '<name>', New name = '<name>'.

Error Type:
Security

User has been disabled. | User = '<name>'.

Error Type:
Security

User group has been disabled. | Group = '<name>'.

Error Type:
Security

User has been enabled. | User = '<name>'.

Error Type:
Security

User group has been enabled. | Group = '<name>'.

Error Type:
Security

Password for user has been changed. | User = '<name>'.

Error Type:
Security

Account '<name>' does not have permission to run this application. Contact the system administrator.

Error Type:
Error

Changing runtime operating mode.

Error Type:
Informational

Runtime operating mode change completed.

Error Type:
Informational

Shutting down to perform an installation.

Error Type:
Informational

OPC ProgID has been added to the ProgID Redirect list. | ProgID = '<ID>'.

Error Type:
Informational

OPC ProgID has been removed from the ProgID Redirect list. | ProgID = '<ID>'.

Error Type:
Informational

The invalid ProgID entry has been deleted from the ProgID Redirect list. | ProgID = '<ID>'.

Error Type:
Informational

Password for administrator was reset by the current user. | Administrator name = '<name>', Current user = '<name>'.

Error Type:
Security

User moved from user group. | User = '<name>', Old group = '<name>', New group '<name>'.

Error Type:
Security

User group has been created. | Group = '<name>'.

Error Type:
Security

User added to user group. | User = '<name>', Group = '<name>'.

Error Type:
Security

User information replaced by import. | File imported = '<absolute file path>'.

Error Type:
Security

User group has been renamed. | Old name = '<name>', New name = '<name>'.

Error Type:
Security

Permissions definition has changed on user group. | Group = '<name>'.

Error Type:
Security

User has been renamed. | Old name = '<name>', New name = '<name>'.

Error Type:
Security

User has been disabled. | User = '<name>'.

Error Type:
Security

User group has been disabled. | Group = '<name>'.

Error Type:
Security

User has been enabled. | User = '<name>'.

Error Type:
Security

User group has been enabled. | Group = '<name>'.

Error Type:
Security

Failed to reset password for administrator. | Administrator name = '<name>'.

Error Type:
Security

Password reset for administrator failed. Current user is not a Windows administrator. | Administrator name = '<name>', Current user = '<name>'.

Error Type:
Security

Password for user has been changed. | User = '<name>'.

Error Type:
Security

General failure during CSV tag import.

Error Type:
Error

Connection attempt to runtime failed. | Runtime host address = '<host address>', User = '<name>', Reason = '<reason>'.

Error Type:
Error

Invalid or missing user information.

Error Type:
Error

Insufficient user permissions to replace the runtime project.

Error Type:
Error

Runtime project update failed.

Error Type:
Error

Failed to retrieve runtime project.

Error Type:
Error

Unable to replace devices on channel because it has an active reference count. | Channel = '<name>'.

Error Type:
Error

Failed to replace existing auto-generated devices on channel, deletion failed. | Channel = '<name>'.

Error Type:
Error

Channel is no longer valid. It may have been removed externally while awaiting user input. | Channel = '<name>'.

Error Type:
Error

No device driver DLLs were loaded.

Error Type:
Error

Device driver was not found or could not be loaded. | Driver = '<name>'.

Error Type:
Error

Error importing CSV data. \n\nField buffer overflow reading identification record.

Error Type:
Error

Error importing CSV data. \n\nUnrecognized field name. | Field = '<name>'.

Error Type:
Error

Error importing CSV data. \n\nDuplicate field name. | Field = '<name>'.

Error Type:
Error

Error importing CSV data. \n\nMissing field identification record.

Error Type:
Error

Error importing CSV record. \n\nField buffer overflow. | Record index = '<number>'.

Error Type:
Error

Error importing CSV record. \n\nInsertion failed. | Record index = '<number>', Record name = '<name>'.

Error Type:
Error

Unable to launch application. | Application = '<path>', OS error = '<code>'.

Error Type:
Error

Error importing CSV record. \n\n'Mapped To' tag address is not valid for this project. | Record index = '<number>', Tag address = '<address>'.

Error Type:
Error

Error importing CSV record. \n\nAlias name is invalid. Names cannot contain double quotations or start with an underscore. | Record index = '<number>'.

Error Type:
Error

Invalid XML document:

Error Type:
Error

Rename failed. There is already an object with that name. | Proposed name = '<name>'.

Error Type:
Error

Failed to start channel diagnostics

Error Type:
Error

Rename failed. Names can not contain periods, double quotations or start with an underscore. | Proposed name = '<name>'.

Error Type:

Error

Synchronization with remote runtime failed.

Error Type:
Error

Account '<name>' does not have permission to run this application. Contact the system administrator.

Error Type:
Error

Error importing CSV record. Tag name is invalid. | Record index = '<number>', Tag name = '<name>'.

Error Type:
Warning

Error importing CSV record. Tag or group name exceeds maximum name length. | Record index = '<number>', Max. name length (characters) = '<number>'.

Error Type:
Warning

Error importing CSV record. Missing address. | Record index = '<number>'.

Error Type:
Warning

Error importing CSV record. Tag group name is invalid. | Record index = '<index>', Group name = '<name>'.

Error Type:
Warning

Close request ignored due to active connections. | Active connections = '<count>'.

Error Type:
Warning

Failed to save embedded dependency file. | File = '<path>'.

Error Type:
Warning

The configuration utility cannot run at the same time as third-party configuration applications. Close both programs and open only the one you want to use. | Product = '<name>'.

Error Type:
Warning

Opening project. | Project = '<name>'.

Error Type:
Informational

Closing project. | Project = '<name>'.

Error Type:
Informational

Virtual Network Mode changed. This affects all channels and virtual networks. See help for more details regarding the Virtual Network Mode. | New mode = '<mode>'.

Error Type:
Informational

Beginning device discovery on channel. | Channel = '<name>'.

Error Type:
Informational

Device discovery complete on channel. | Channel = '<name>', Devices found = '<count>'.

Error Type:
Informational

Device discovery canceled on channel. | Channel = '<name>'.

Error Type:
Informational

Device discovery canceled on channel. | Channel = '<name>', Devices found = '<count>'.

Error Type:
Informational

Unable to begin device discovery on channel. | Channel = '<name>'.

Error Type:
Informational

Shutting down for the purpose of performing an installation.

Error Type:
Informational

Runtime project has been reset.

Error Type:
Informational

Runtime project replaced. | New project = '<path>'.

Error Type:
Informational

Not connected to the event logger service.

Error Type:
Security

Attempt to add item '<name>' failed.

Error Type:
Error

No device driver DLLs were loaded.

Error Type:
Error

Invalid project file: '<name>'.

Error Type:
Error

Could not open project file: '<name>'.

Error Type:
Error

Rejecting request to replace the project because it's the same as the one in use: '<name>'.

Error Type:
Error

Filename must not overwrite an existing file: '<name>'.

Error Type:
Error

Filename must not be empty.

Error Type:
Error

Filename is expected to be of the form <subdir>/<name>.{json,opf,sopf}

Error Type:
Error

Filename contains one or more invalid characters.

Error Type:
Error

Saving project files with Project File Encryption enabled as .OPF file type is not supported. Supported file types are .SOPF and .JSON.

Error Type:
Error

Saving project files with Project File Encryption disabled as .SOPF file type is not supported. Supported file types are .OPF and .JSON.

Error Type:
Error

Account '<name>' does not have permission to run this application. Contact the system administrator.

Error Type:
Error

A password is required for saving encrypted project files (.SOPF).

Error Type:
Error

Saving .OPF and .JSON project files with a password is not supported. To save encrypted project files, use .SOPF.

Error Type:
Error

Addition of object to '<name>' failed: <reason>.

Error Type:
Warning

Move object '<name>' failed: <reason>.

Error Type:

Warning

Update of object '<name>' failed: <reason>.

Error Type:
Warning

Delete object '<name>' failed: <reason>.

Error Type:
Warning

Unable to load startup project '<name>': <reason>.

Error Type:
Warning

Failed to update startup project '<name>': <reason>.

Error Type:
Warning

Runtime project replaced with startup project defined. Runtime project will be restored from '<name>' at next restart.

Error Type:
Warning

Ignoring user-defined startup project because a configuration session is active.

Error Type:
Warning

Write request rejected on read-only item reference '<name>'.

Error Type:
Warning

Unable to write to item '<name>'.

Error Type:
Warning

Write request failed on item '<name>'. The write data type '<type>' cannot be converted to the tag data type '<type>'.

Error Type:
Warning

Write request failed on item '<name>'. Error scaling the write data.

Error Type:

Warning

Write request rejected on item reference '<name>' since the device it belongs to is disabled.

Error Type:
Warning

<Name> successfully configured to run as a system service.

Error Type:
Informational

<Name> successfully removed from the service control manager database.

Error Type:
Informational

Runtime re-initialization started.

Error Type:
Informational

Runtime re-initialization completed.

Error Type:
Informational

Updated startup project '<name>'.

Error Type:
Informational

Runtime service started.

Error Type:
Informational

Runtime process started.

Error Type:
Informational

Runtime performing exit processing.

Error Type:
Informational

Runtime shutdown complete.

Error Type:
Informational

Shutting down to perform an installation.

Error Type:
Informational

Runtime project replaced from '<name>'.

Error Type:
Informational

Missing application data directory.

Error Type:
Informational

Runtime project saved as '<name>'.

Error Type:
Informational

Runtime project replaced.

Error Type:
Informational

Configuration session started by <name> (<name>).

Error Type:
Security

Configuration session assigned to <name> has ended.

Error Type:
Security

Configuration session assigned to <name> promoted to write access.

Error Type:
Security

Configuration session assigned to <name> demoted to read only.

Error Type:
Security

Permissions change applied on configuration session assigned to <name>.

Error Type:
Security

Failed to start Script Engine server. Socket error occurred binding to local port. | Error = <error>, Details = '<information>'.

Error Type:
Error

Possible Cause:
The port conflicts with another application.

Possible Solution:
Use the server administration settings to update the Script Engine port.

An unhandled exception was thrown from the script. | Function = '<function>', error = '<error>'.

Error Type:
Error

Possible Cause:
An exception was thrown from the script.

Possible Solution:
Correct the condition that lead to the exception, or update the script logic.

Script Engine service stopping.

Error Type:
Informational

Script Engine service starting.

Error Type:
Informational

The Config API SSL certificate contains a bad signature.

Error Type:
Error

The Config API is unable to load the SSL certificate.

Error Type:
Error

Unable to start the Config API Service. Possible problem binding to port.

Error Type:
Error

Possible Cause:
The HTTP or HTTPS port specified in the Config API settings is already bound by another application.

Possible Solution:
Change the configuration of the Config API or blocking application to use a different port, or stop the application blocking the port.

The Config API SSL certificate has expired.

Error Type:
Warning

The Config API SSL certificate is self-signed.

Error Type:
Warning

Configuration API started without SSL on port <port number>.

Error Type:
Informational

Configuration API started with SSL on port <port number>.

Error Type:
Informational

The OPC .NET server failed to start. Please see the windows application event log for more details. Also make sure the .NET 3.5 Framework is installed. | OS Error = '<error reason>'.

Error Type:
Error

The OPC .NET server failed to start because it is not installed. Please rerun the installation.

Error Type:
Error

Timed out trying to start the OPC .NET server. Please verify that the server is running by using the OPC .NET Configuration Manager.

Error Type:
Warning

Missing server instance certificate '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate.

Error Type:
Error

Failed to import server instance cert: '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate.

Error Type:
Error

Possible Cause:
1. The file containing the server instance certificate does not exist or is inaccessible.
2. Certificate decryption failed.

Possible Solution:
1. Verify the file references a valid instance certificate to which the user has permissions.
2. Re-issue the certificate to refresh the encryption.
3. Import a new certificate.

The UA server certificate is expired. Please use the OPC UA Configuration Manager to reissue the certificate.

Error Type:
Error

Possible Cause:
The validity period of the certificate is before the current system date.

Possible Solution:
1. Import a non-expired certificate.
2. Re-issue the certificate to generate a new non-expired certificate.

A socket error occurred listening for client connections. | Endpoint URL = '<endpoint URL>', Error = <error code>, Details = '<description>'.

Error Type:
Error

Possible Cause:
The endpoint socket returned an error while listening for client connections.

Possible Solution:
Note the details in the error message to diagnose the problem.

The UA Server failed to register with the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:
Error

Possible Cause:

1. The UA server endpoint URL and the security policy are not supported in the UA Discovery Server.
2. The attempt to register the UA Server with the UA Discovery Server could not complete in the expected manner.

Possible Solution:

Verify the UA Server endpoint URL and the security policy with the UA Discovery Server endpoints.

Unable to start the UA server due to certificate load failure.

Error Type:

Error

Possible Cause:

1. The UA Server application instance certificate validity period occurs before the current system date.
2. The file containing the server instance certificate does not exist or is inaccessible.
3. Certificate decryption failed.

Possible Solution:

1. Import a non-expired certificate.
2. Re-issue the certificate to generate a new non-expired certificate.
3. Verify the file references a valid instance certificate to which the user has permissions.
4. Re-issue the certificate to refresh the encryption.

The UA Server failed to unregister from the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:

Warning

Possible Cause:

1. The UA server endpoint URL and the security policy are not supported in the UA Discovery Server.
2. The attempt to unregister the UA Server from the UA Discovery Server could not complete in the expected manner.

Possible Solution:

Verify the UA Server endpoint URL and the security policy with the UA Discovery Server endpoints.

The UA Server successfully registered with the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:
Informational

The UA Server successfully unregistered from the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:
Informational

The ReadProcessed request timed out. | Elapsed Time = <seconds> (s).

Error Type:
Error

The ReadAtTime request timed out. | Elapsed Time = <seconds> (s).

Error Type:
Error

Attempt to add DDE item failed. | Item = '<item name>'.

Error Type:
Error

DDE client attempt to add topic failed. | Topic = '<topic>'.

Error Type:
Error

Possible Cause:
Topic name is not valid.

Possible Solution:
View the Alias map to correct the reference to a valid topic.

See Also:
Alias Maps

Unable to write to item. | Item = '<item name>'.

Error Type:
Warning

The area specified is not valid. Failed to set the subscription filter. | Area = '<area name>'.

Error Type:
Error

The source specified is not valid. Failed to set the subscription filter. | Source = '<source name>'.

Error Type:
Error

The Config API SSL certificate contains a bad signature.

Error Type:
Error

The Config API is unable to load the SSL certificate.

Error Type:
Error

Unable to start the Config API Service. Possible problem binding to port.

Error Type:
Error

Possible Cause:
The HTTP or HTTPS port specified in the Config API settings is already bound by another application.

Possible Solution:
Change the configuration of the Config API or blocking application to use a different port, or stop the application blocking the port.

The Config API SSL certificate has expired.

Error Type:
Warning

The Config API SSL certificate is self-signed.

Error Type:
Warning

Configuration API started without SSL on port <port number>.

Error Type:
Informational

Configuration API started with SSL on port <port number>.

Error Type:
Informational

Connection to ThingWorx failed. | Platform = <host:port resource>, error = <reason>.

Error Type:

Error

Possible Cause:
The connection to the ThingWorx Platform could not be established.

Possible Solution:

1. Verify that the host, port, resource, and application key are all valid and correct.
2. Verify that the host machine can reach the Composer on the ThingWorx Platform.
3. Verify that the proper certificate settings are enabled if using a self-signed certificate or no encryption.

Error adding item. | Item name = '<item name>'.

Error Type:
Error

Possible Cause:
The item <TagName> could not be added to the server for scanning.

Possible Solution:

1. Verify that the tag exists on a valid channel and device.
2. Verify that the tag may be read using another client, such as the QuickClient.

Failed to trigger the autobind complete event on the platform.

Error Type:
Error

Possible Cause:
The ThingWorx connection was terminated before the autobind process completed.

Possible Solution:
Wait to reinitialize or alter the ThingWorx project properties until after all autobinds have completed.

Connection to ThingWorx failed for an unknown reason. | Platform = <host:port resource>, error = <error>.

Error Type:
Error

Possible Cause:
The connection to the ThingWorx Platform failed.

Possible Solution:

1. Verify that the host, port, resource, and application key are all valid and correct.
2. Verify that the host machine can reach the Composer on the ThingWorx Platform.
3. Verify that the proper certificate settings are enabled if using a self-signed certificate or no encryption.
4. Contact technical support with the error code and an application report.

One or more value change updates lost due to insufficient space in the connection buffer. | Number of lost updates = <count>.

Error Type:
Error

Possible Cause:
Data is being dropped because the ThingWorx Platform is not available or too much data is being collected by the instance.

Possible Solution:
1. Verify that some data is updating on the ThingWorx Platform and that the platform is reachable.
2. Slow down the tag scan rate to move less data into the ThingWorx Platform.

Item failed to publish; multidimensional arrays are not supported. | Item name = '%s'.

Error Type:
Error

Possible Cause:
The item <ItemName> references a tag whose data is a multidimensional array.

Possible Solution:
Modify the item to reference a tag with a supported datatype.

Store and Forward datastore unable to store data due to full disk.

Error Type:
Error

Possible Cause:
The disk being used to store updates has been filled to within 500 MiB.

Possible Solution:
1. Free up some space on the disk being used to store updates.
2. Delete the data stored in the datastore using the _DeleteStoredData system tag.
3. Replace the disk being used to store data with a larger disk.

Store and Forward datastore size limit reached.

Error Type:
Error

Possible Cause:
The ThingWorx Interface is not able to send updates to the platform as fast as the updates are being generated.

Possible Solution:

1. Verify that the ThingWorx Interface can connect to the ThingWorx Platform.
2. Reduce the rate of updates being collected by the ThingWorx Interface.

Connection to ThingWorx was closed. | Platform = <host:port resource>.

Error Type:
Warning

Possible Cause:
The connection was closed. The service was stopped or the interface is no longer able to reach the platform.

Possible Solution:

1. Verify that the native interface is enabled in the project properties.
2. Verify that the host machine can reach the Composer on the ThingWorx Platform.

Failed to autobind property. | Name = '<property name>'.

Error Type:
Warning

Possible Cause:
A property with this name already exists under this Thing.

Possible Solution:

1. Check the property to see if data is current.
2. If data is not current, delete the property under the Thing and run the AddItem service once again.

Failed to restart Thing. | Name = '<thing name>'.

Error Type:
Warning

Possible Cause:
When the AddItem service is complete, a restart service is called on the Thing. This allows the Composer to visualize the changes. Data changes are sent to the platform even when this error has been presented.

Possible Solution:
Relaunch the Composer to restart the Thing.

Write to property failed. | Property name = '<name>', reason = <reason>.

Error Type:
Warning

Possible Cause:
Unable to write to a tag due to a conversion issue.

Possible Solution:

1. Verify that the data type of the tag in KEPServerEX, as well as in the ThingWorx Platform, is correct and consistent.
2. Verify that the value to be written is within the appropriate range for the data type.

ThingWorx request to add item failed. The item was already added. | Item name = '<name>'.

Error Type:
Warning

Possible Cause:
The tag had already been added to this Thing.

Possible Solution:

1. Check the property to see if data is current.
2. If data is not current, delete the property under the Thing and run the AddItem service once again.

ThingWorx request to remove item failed. The item doesn't exist. | Item name = '<name>'.

Error Type:
Warning

Possible Cause:
The tag was already removed from the Thing or no such tag exists.

Possible Solution:
If the tag still shows under the properties of the Thing, delete that property in the ThingWorx Composer.

The server is configured to send an update for every scan, but the push type of one or more properties are set to push on value change only. | Count = <count>.

Error Type:

Warning

Possible Cause:
The push type in the ThingWorx Platform is set to change only for some items. This push type only updates data on the platform when the data value changes.

Possible Solution:
To use the Send Every Scan option, set this value to Always.

The push type of one or more properties are set to never push an update to the platform. | Count = <count>.

Error Type:
Warning

Possible Cause:
The push type in the ThingWorx Platform is set to Never for some items, which prevents any data changes from being automatically updated on the platform.

Possible Solution:
If this is not the desired behavior, change the push type in the ThingWorx Platform.

ThingWorx request to remove an item failed. The item is bound and the force flag is false. | Item name = '<name>'.

Error Type:
Warning

Possible Cause:
The RemoveItems service could not remove the item because it is bound to a property and the Force Flag is not set to True.

Possible Solution:
Re-run the service, explicitly calling the ForceRemove flag as True.

Write to property failed. | Thing name = '<name>', property name = '<name>', reason = <reason>.

Error Type:
Warning

Possible Cause:
Unable to write to a tag due to a conversion issue.

Possible Solution:

1. Verify that the data type of the tag in KEPServerEX, as well as in the ThingWorx Platform, is correct and consistent.
2. Verify that the value to be written is within the appropriate range for the data type.

Error pushing property updates to thing. | Thing name = '<name>'.

Error Type:
Warning

Possible Cause:
Property updates for the named thing were not successfully published to the platform.

Possible Solution:
Check the platform's log for an indication of why property updates are failing, such as a permissions issue.

Unable to connect or attach to Store and Forward datastore. Using in-memory store. | In-memory store size (updates) = <count>.

Error Type:
Warning

Possible Cause:
1. The Store and Forward service is not running.
2. The service does not have access to the specified storage directory.
3. There is a port conflict that prevents the Store and Forward service from accepting connections.

Possible Solution:
1. Restart the server runtime.
2. Verify the specified storage location is accessible by the Store and Forward service.
3. Resolve the port conflict by configuring a new port for Store and Forward in the server administration.

Store and Forward datastore reset due to file IO error or datastore corruption.

Error Type:
Warning

Possible Cause:
1. The datastore was corrupted by a user or another program.
2. The datastore was corrupted by a hardware error.
3. An error occurred while attempting to read data from disk, possibly due to a hardware issue.

Possible Solution:
1. Use User Access Controls to limit the which users have access to the datastore location.
2. Move the datastore to another disk.

Unable to apply settings change initiated by the Platform. Permission Denied. | User = '<user name>'.

Error Type:
Warning

Possible Cause:
The user group "ThingWorx Interface Users" has the permissions "Project Modification:Servermain.Project" set to "Deny".

Possible Solution:
Set the permission "Project Modification:Servermain.Project" on the user group "ThingWorx Interface Users" to "Allow".

Configuration Transfer to ThingWorx Platform failed.

Error Type:
Warning

Configuration Transfer to ThingWorx Platform failed. | Reason = '<reason>'

Error Type:
Warning

Possible Cause:

1. Refer to reason text for more information.
2. The runtime project is locked because a user is editing it.
3. The ThingWorx Interface user account does not have sufficient privelages to perform the operation.

Failed to delete stored updates in the Store and Forward datastore.

Error Type:
Warning

Possible Cause:
A hardware or operating system error prevented the operation from completing.

Possible Solution:
Restart the machine and try again.

Configuration Transfer from ThingWorx Platform failed.

Error Type:
Warning

Configuration Transfer from ThingWorx Platform failed. | Reason = '<reason>'

Error Type:
Warning

Possible Cause:
1. Refer to reason text for more information.
2. The runtime project is locked because a user is editing it.
3. The ThingWorx Interface user account does not have sufficient privelages to perform the operation.

Check that your Application Key is properly formatted and valid.

Error Type:
Warning

Possible Cause:
The connection to the ThingWorx Platform failed due to bad authorization.

Possible Solution:
1. Verify that application key has not expired.
2. Verify that application key is properly formatted.
3. Verify that application key was inputted correctly.

Connected to ThingWorx. | Platform = <host:port resource>, Thing name = '<name>'.

Error Type:
Informational

Possible Cause:
A connection was made to the ThingWorx Platform.

Reinitializing ThingWorx connection due to a project settings change initiated from the platform.

Error Type:
Informational

Possible Cause:
When using the SetConfiguration service, this message informs an operator viewing the KEPServerEX event log that a change was made.

Dropping pending autobinds due to interface shutdown or reinitialize. | Count = <count>.

Error Type:
Informational

Possible Cause:
A server shutdown or initialization was called while auto-binding was in process from an AddItems service call.

Possible Solution:
Any Items not auto bound need to be manually created and bound in the ThingWorx Composer.

Serviced one or more autobind requests. | Count = <count>.

Error Type:
Informational

Possible Cause:
Part of the AddItems service is the autobind action. This action may take more time than the actual adding of the item. This message alerts the operator to how many items have been autobound.

Reinitializing ThingWorx connection due to a project settings change initiated from the Configuration API.

Error Type:
Informational

Possible Cause:
When using the Configuration API, this message informs an operator viewing the KEPServerEX event log that a change was made.

Resumed pushing property updates to thing: the error condition was resolved. | Thing name = '<name>'.

Error Type:
Informational

Configuration transfer from ThingWorx initiated.

Error Type:
Informational

Configuration transfer from ThingWorx aborted.

Error Type:
Informational

Initialized Store and Forward datastore. | Datastore location: '<location>'.

Error Type:
Informational

Possible Cause:
ThingWorx Native Interface is configured to use Store and Forward.

Successfully deleted stored data from the Store and Forward datastore.

Error Type:
Informational

Possible Cause:
A client used the _DeleteStoredData system tag to delete data cached for ThingWorx Interface in the Store and Forward datastore.

Store and Forward mode changed. | Forward Mode = '<mode>'.

Error Type:
Informational

Possible Cause:
The _ForwardMode system tag was written to by a connected client and the value of the write caused a settings change.

Initialized Store and Forward datastore. | Forward Mode = '<mode>' | Datastore location = '<location>'.

Error Type:
Informational

Possible Cause:
ThingWorx Native Interface is configured to use Store and Forward.

Error attaching to datastore due to an invalid datastore path. | Path = '<path>'

Error Type:
Error

Possible Cause:
The path specified by the component using Store and Forward is invalid. Refer to that component's documentation and the validation error contained in the message's body for more information.

Possible Solution:
Correct the error noted in the message.

Failed to start Store and Forward server. Socket error occurred binding to local port. | Error = <error>, Details = '<information>'.

Error Type:
Error

Possible Cause:
The port conflicts with another application.

Possible Solution:
Use the server administration settings to update the Store and Forward port.

Store and Forward service stopping.

Error Type:
Informational

Store and Forward service starting.

Error Type:
Informational

File corruption encountered when attaching to datastore; datastore recreated. | Datastore path = '<path>'.

Error Type:
Informational

Possible Cause:
A file used by the datastore was corrupted by the system, another application, or a user.

Possible Solution:

1. The old datastore is automatically replaced, no user action is needed.
2. If this problem occurs repeatedly, consider changing the datastore directory to a location that cannot be accessed by other applications or users.

Datastore overwritten due to a configuration change. | Datastore path = '<path>'.

Error Type:
Informational

Possible Cause:
The datastore size parameter was changed.

Note:
Changing the datastore size results in all of the datastore's files being recreated. Unless data was actively being stored in the datastore due to a disconnect from the ThingWorx Platform, it is unlikely that data was lost.

Unable to attach to existing datastore because that datastore was created with an older version of the server. Datastore recreated. | Datastore path = '<path>'.

Error Type:
Informational

Possible Cause:
The server was upgraded to a version which uses a newer datastore format.

Possible Solution:
The old datastore was replaced with a new version datastore; no user action is needed.

Com port is in use by another application. | Port = '<port>'.

Error Type:
Error

Possible Cause:
The serial port assigned to a device is being used by another application.

Possible Solution:

1. Verify that the correct port has been assigned to the channel.
2. Verify that only one copy of the current project is running.

Unable to configure com port with specified parameters. | Port = COM<number>, OS error = <error>.

Error Type:
Error

Possible Cause:
The serial parameters for the specified COM port are not valid.

Possible Solution:
Verify the serial parameters and make any necessary changes.

Driver failed to initialize.

Error Type:
Error

Unable to create serial I/O thread.

Error Type:
Error

Possible Cause:

The server process has no resources available to create new threads.

Possible Solution:
Each tag group consumes a thread. The typical limit for a single process is about 2000 threads. Reduce the number of tag groups in the project.

Com port does not exist. | Port = '<port>'.

Error Type:
Error

Possible Cause:
The specified COM port is not present on the target computer.

Possible Solution:
Verify that the proper COM port is selected.

Error opening com port. | Port = '<port>', OS error = <error>.

Error Type:
Error

Possible Cause:
The specified COM port could not be opened due an internal hardware or software problem on the target computer.

Possible Solution:
Verify that the COM port is functional and may be accessed by other applications.

Connection failed. Unable to bind to adapter. | Adapter = '<name>'.

Error Type:
Error

Possible Cause:
Since the specified network adapter cannot be located in the system device list, it cannot be bound to for communications. This can occur when a project is moved from one PC to another (and when the project specifies a network adapter rather than using the default). The server reverts to the default adapter.

Possible Solution:
Change the Network Adapter property to Default (or select a new adapter), save the project, and retry.

Winsock shut down failed. | OS error = <error>.

Error Type:
Error

Winsock initialization failed. | OS error = <error>.

Error Type:
Error

Possible Solution:

1. The underlying network subsystem is not ready for network communication. Wait a few seconds and restart the driver.
2. The limit on the number of tasks supported by the Windows Sockets implementation has been reached. Close one or more applications that may be using Winsock and restart the driver.

Winsock V1.1 or higher must be installed to use this driver.

Error Type:
Error

Possible Cause:
The version number of the Winsock DLL found on the system is older than 1.1.

Possible Solution:
Upgrade Winsock to version 1.1 or higher.

Socket error occurred binding to local port. | Error = <error>, Details = '<information>'.

Error Type:
Error

Device is not responding.

Error Type:
Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communication parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect device ID.
4. The response from the device took longer to receive than allowed by the Request Timeout device setting.

Possible Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications parameters match those of the device.
3. Verify that the device ID for the named device matches that of the actual device.
4. Increase the Request Timeout setting to allow the entire response to be handled.

Device is not responding. | ID = '<device>'.

Error Type:
Warning

Possible Cause:
1. The network connection between the device and the host PC is broken.
2. The communication parameters configured for the device and driver do not match.
3. The response from the device took longer to receive than allowed by the Request Timeout device setting.

Possible Solution:
1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications parameters match those of the device.
3. Increase the Request Timeout setting to allow the entire response to be handled.

Serial communications error on channel. | Error mask = <mask>.

Error Type:
Warning

Possible Cause:
1. The serial connection between the device and the host PC is broken.
2. The communications parameters for the serial connection are incorrect.

Possible Solution:
1. Investigate the error mask code and the related information.
2. Verify the cabling between the PC and the PLC device.
3. Verify that the specified communication parameters match those of the device.

See Also:
Error Mask Codes

Invalid array size detected writing to tag <device name>.<address>.

Error Type:
Warning

Possible Cause:
Client trying to write before being updated.

Possible Solution:
Perform a read on the array before attempting a write.

Unable to write to address on device. | Address = '<address>'.

Error Type:
Warning

Possible Cause:
1. The connection between the device and the host PC is broken.
2. The communications parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect device ID.

Possible Solution:
1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communication parameters match those of the device.
3. Verify that the device ID given to the named device matches that of the actual device.

Items on this page may not be changed while the driver is processing tags.

Error Type:
Warning

Possible Cause:
An attempt was made to change a channel or device configuration while data clients were connected to the server and receiving data from the channel/device.

Possible Solution:
Disconnect all data clients from the server before making changes.

Specified address is not valid on device. | Invalid address = '<address>'.

Error Type:
Warning

Possible Cause:
A tag address has been assigned an invalid address.

Possible Solution:
Modify the requested address in the client application.

Address '<address>' is not valid on device '<name>'.

Error Type:
Warning

This property may not be changed while the driver is processing tags.

Error Type:
Warning

Unable to write to address '<address>' on device '<name>'.

Error Type:
Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communications parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect device ID.

Possible Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communication parameters match those of the device.
3. Verify that the device ID given to the named device matches that of the actual device.

Socket error occurred connecting. | Error = <error>, Details = '<information>'.

Error Type:
Warning

Possible Cause:
Communication with the device failed during the specified socket operation.

Possible Solution:
Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred receiving data. | Error = <error>, Details = '<information>'.

Error Type:
Warning

Possible Cause:
Communication with the device failed during the specified socket operation.

Possible Solution:
Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred sending data. | Error = <error>, Details = '<information>'.

Error Type:
Warning

Possible Cause:
Communication with the device failed during the specified socket operation.

Possible Solution:
Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred checking for readability. | Error = <error>, Details = '<information>'.

Error Type:
Warning

Possible Cause:
Communication with the device failed during the specified socket operation.

Possible Solution:
Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred checking for writability. | Error = <error>, Details = '<information>'.

Error Type:
Warning

Possible Cause:
Communication with the device failed during the specified socket operation.

Possible Solution:
Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

%s |

Error Type:
Informational

<Name> Device Driver '<name>'

Error Type:
Informational

Resources

In addition to this user manual, there are a variety of resources available to assist customers, answer questions, provide more detail about specific implementations, or help with troubleshooting specific issues.

Knowledge Base
Whitepapers
Connectivity Guides
Technical Notes
Training Programs
Training Videos
Kepware Technical Support
PTC Technical Support

Index

%

<

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W