SIXNET UDR Driver

Table of Contents

SIXNET UDR Driver

Help version 1.025

CONTENTS

Overview

The SIXNET UDR Driver provides a reliable way to connect SIXNET UDR devices to OPC Client applications, including HMI, SCADA, Historian, MES, ERP and countless custom applications. It is intended for use with all SIXNET Controllers, RTUs and I/O devices that support the SIXNET UDR (Universal DRiver) protocol. This includes all VersaTRACK, SixTRACK, EtherTRACK, RemoteTRAK, and SiteTRAK products.

This driver was created in partnership with SIXNET. SIXNET endorses the use of this driver as the SIXNET official OPC interface for their products. This driver uses SIXNET supplied communication software modules that enable it to run simultaneously with the SIXNET I/O Tool Kit and ISaGRAF Workbench. This unique capability allows users to perform configuration changes, Datalog file transfers, analog calibrations, and a full suite of other system functions while I/O updating continues in real-time. This is true not only of Ethernet connections but of serial communications as well.

Note: This multitasking capability works over phone lines and wireless radio connections.

Unsolicited Communication-Report by Exception

Multiple virtual UDR server devices can be created to accept and process unsolicited commands as though they were actual I/O devices on the network. Solicited communication (where the I/O device in the field is a UDR server) and unsolicited communication (where the I/P device in the field is the UDR client) can be performed simultaneously. In other words, UDR client and server operation may occur simultaneously over separate channels in the same SIXNET driver. *For more information, refer to* ***UDR Server Devices****.*

External Dependencies

This driver has external dependencies. It uses two SIXNET Universal Driver (UDR) communications library components supplied by SIXNET. *For more information, refer to* ***SIXNET Software Components: Universal Driver Components****.*

Setup

The SIXNET UDR Driver makes use of some of the same software components used by SIXNET applications, such as the Remote I/O Tool Kit. This permits the simultaneous operation of these applications and this driver. The OPC server setup will install these components along with this driver. *For more information regarding these components, refer to* ***SIXNET Software Components: Universal Driver Components****.*

For recommendations on optimizing configuration, refer to ***Optimizing Your SIXNET UDR Communications****.*

Supported Devices

All SIXNET Controllers, RTUs and I/O devices that support the SIXNET UDR protocol. This includes the all of the following products:
VersaTRACK
SixTRACK
EtherTRACK
RemoteTRAK
SiteTRAK

Supported Protocol

SIXNET UDR protocol over serial lines and Ethernet using UDP.

TCP/IP

TCP/IP must be properly installed to use this driver with Ethernet devices. For more information, refer to the Windows documentation on setting up TCP/IP. *For more information, refer to* ***Setup****.*

Channel and Device Limits

A channel represents a serial line connected to one of the computer's COM ports or an Ethernet network connected to the computer's default **Network Interface Card (NIC)**. Channel properties are used to specify what type of connection is desired and what other properties are shared by devices on that network. The maximum number of channels supported by this driver is 100, including multiple channels that connect to the same COM port or Ethernet adapter.

Each physical device to be polled must be represented by a device object in the OPC server. UDR server device objects may be created to act as virtual I/O devices on the network. Each device on a network must be configured with a unique station number. Devices on separate, unconnected networks may have the same station number. Devices that act as gateways to other networks, such as an EtherTRAK unit connected to a network of RemoteTRAK units on an RS-485 party line, must be configured to operate in pass-thru mode. The maximum number of devices supported by this driver is 8192 per channel. The number of these devices that may operate as UDR servers is limited to 64 per COM port plus 64 over Ethernet. *For more information, refer to* ***SIXNET Software Components: Universal Driver Components****.*

Tip: Controller and RTU units may be programmed to send unsolicited requests to the driver's UDR server devices, although not all UDR commands are supported. *For more information, refer to* ***UDR Server Devices****.*

Networking

This driver supports communications over serial lines and Ethernet using UDP. Ethernet performance is not inhibited by the simultaneous use of serial communication. Devices on multi-layered networks may be addressed. For more information, refer to **Network**.

SIXNET Software Components: Universal Driver Components

This driver uses two SIXNET Universal Driver (UDR) communications library components supplied by SIXNET: Six32com.exe and Udrcom32.dll. These files will be placed in the server installation folder when the driver is installed. If copies of these files were previously installed in the Windows system directory with a SIXNET application, they will not be affected by the driver installation.

Limitations

A session is created each time an application makes a connection to a given COM port or to the default Ethernet adapter through UDR components. There is a limit of 64 sessions per COM port plus 64 for the Ethernet adapter. This driver will start a session for each channel, plus an additional session for each UDR server device. Channel sessions will not be started until the scanning of tags belonging to its UDR client devices has begun. Likewise, UDR server device sessions are not started until scanning of its tags has begun. A tag is scanned as long as a client that references that tag is connected to the server. Once a session is started by this driver, it is not ended until the server is shutdown or a new project is loaded. SIXNET applications may start one or more sessions and contribute to the total serviced by these components. The limit of 64 sessions per port / Ethernet should not be prohibitive unless users need a large number of UDR server devices on a single serial or Ethernet line.

Performance

There is a minimum turnaround time of approximately 15 ms per transaction imposed by the UDR components. This limits the performance of the driver to roughly 64 transactions per second for a given channel. Both these components and the driver support multi-threading, however, so users may be able to improve performance by using multiple channels. If the device and network is fast enough, users may be able to nearly double the total transaction rate by creating a duplicate device on a second channel and dividing the tag blocks equally between those devices. *For more information, refer to **Optimizing Your SIXNET UDR Communications**.*

INI File

The UDR components read serial port configuration parameters (such as parity and baud rate) from a Windows application initialization file. This file is called sixtrack.ini and is located in the Windows folder. Both this driver and SIXNET applications will create this file if needed and modify its contents according to user input.

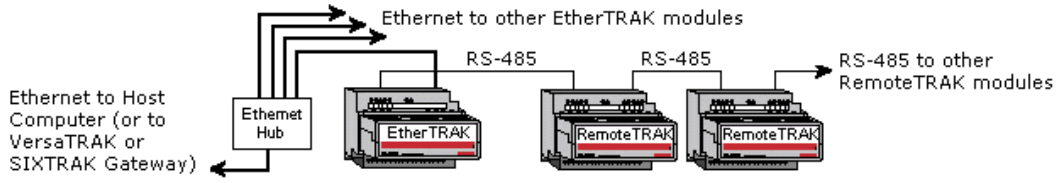
Important: If this driver is used along with SIXNET applications on the same computer, the serial port configurations must be the same. If the settings conflict, it is likely that communication problems will arise because the settings received by the UDR components cannot be predicted.

Uninstalling SIXNET Applications

If users decide to uninstall SIXNET applications, the operation of the SIXNET UDR Driver will not be affected.

Networks

SIXNET offers a wide range of products, allowing unlimited possibilities for network configuration. One powerful feature of some devices is the ability to pass messages through to other network layers. For example, refer to the image below.



Here, a number of RemoteTRAK modules are connected to an RS-485 serial line. This network layer is connected to an Ethernet network via an EtherTRAK module. The EtherTRAK module is configured to operate in "Pass-thru mode" so that it will forward messages to and from the RemoteTRAK modules. If the EtherTRAK module receives a request with a station number matching its own, it will process the request. If the station number does not match, it will repeat the message out its RS-485 port. One of the RemoteTRAK modules will respond if its station number matches that in the request.

To communicate with all of the modules in this setup, a channel that uses this driver must be defined in the OPC server. Next, a device must be added to the channel for each of the modules. Tags can then be added to each device to access the I/O of the associated module.

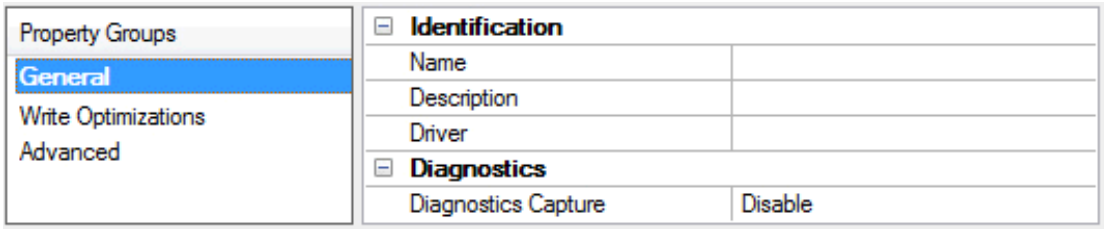
Since the Host PC is directly connected to the Ethernet network in this example and not to the RS-485 serial line, the channel must be configured to use Ethernet. For more information, refer to **Communications Parameters**.

The device associated with the EtherTRAK module must be configured to use the IP address and station number of the EtherTRAK module. The devices associated with the RemoteTRAK modules must use the IP address of the EtherTRAK module, and the station number of the target RemoteTRAK module. Similar principles hold when dealing with other network configurations.

See Also: Station

Channel Properties — General

This server supports the use of multiple simultaneous 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: Specify the 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: Specify user-defined information about this channel.
Many of these properties, including Description, have an associated system tag.

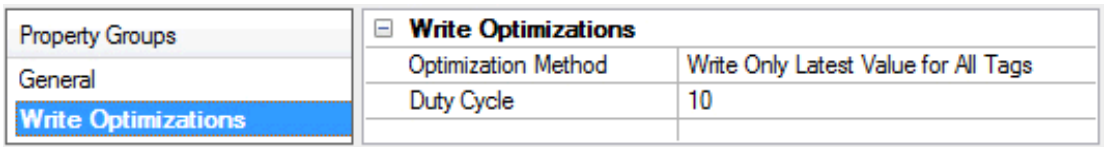
Driver: Specify the protocol / driver for this channel. This property specifies the device driver that was selected during channel creation. It is a disabled setting in the channel properties. The property is required for creating a channel.
Note: With the server's online full-time operation, these properties 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 proper user role and privilege management to prevent operators from changing properties or accessing server features.

Diagnostics

Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications allows the usage of statistics tags that provide feedback to client applications regarding the operation of the channel. 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" and "Statistics Tags" in the server help.

Channel Properties — Write Optimizations

The server must ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties 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.

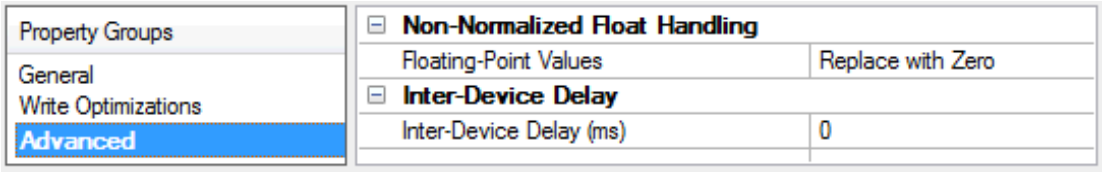
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write 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.

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.

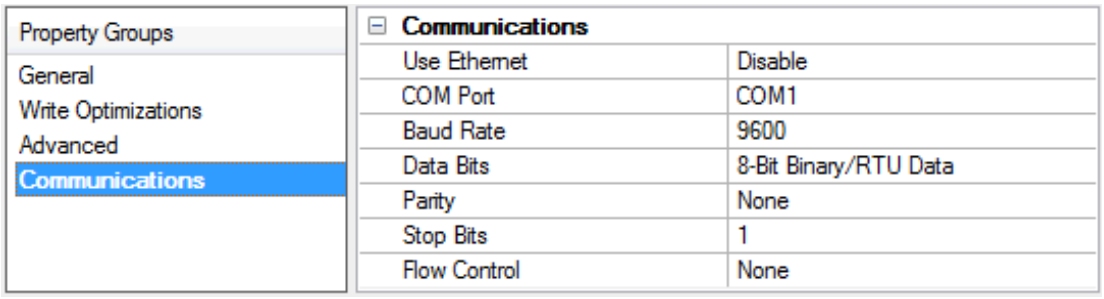
Note: This property is not available if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example,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



Use Ethernet

This property specifies whether the channel is to use serial (default) or Ethernet communication.

COM Port

This property specifies the COM port that will be used for serial communications. The range is 1 to 255. The default setting is COM1.

Baud Rate

This property specifies the baud rate that should be used to configure the selected COM port. Supported baud rates are as follows: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400 and 57600. The default setting is 9600.

Data Bits

This property specifies the number of data bits per data word (7 or 8) and the data format (binary or ASCII Hex) used with serial communication. For example, a value of 10 is sent as a single byte 0x0A using binary format, and as two bytes 0x31 (ASCII "1") 0x30 (ASCII "0") using ASCII Hex format. Supported data bit and format combinations are: 8-bit binary/RTU data, 8-bit ASCII (hex) data and 7-bit ASCII (hex) data. The default setting is 8-bit binary/RTU data.

Note: Binary format is always used with Ethernet communications.

Parity

This property specifies the type of parity the data should use. Choose from: None, Even, Odd, Mark and Space. The default setting is None.

Stop Bits

This property specifies the number of stop bits per data word. Choose between 1 or 2. The default setting is 1.

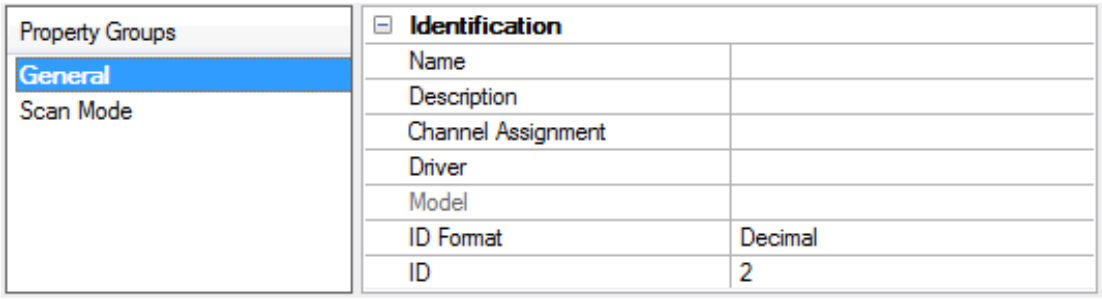
Flow Control

This property specifies how the RTS and DTR control lines should be utilized. Choose from None, Hardware and Xon/Xoff. The default setting is None.

See Also: SIXNET Software Components: Universal Driver Components and Optimizing Your SIXNET UDR Communications.

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: Specify 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: Specify the user-defined information about this device.
Many of these properties, including Description, have an associated system tag.

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

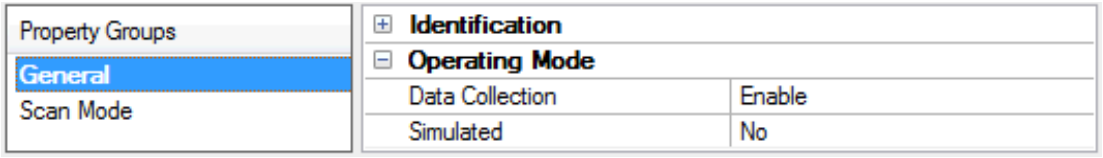
Driver: Selected protocol driver for this device.

Model: Specify the 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: Specify 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: Place the device into or out of 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.

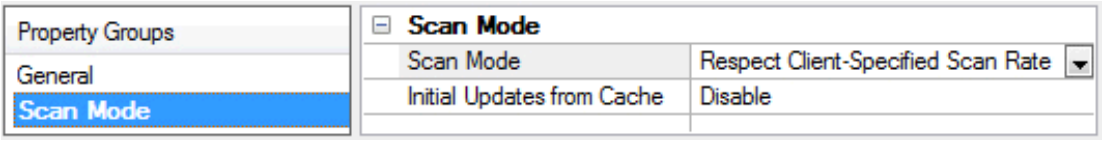
Notes:

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.



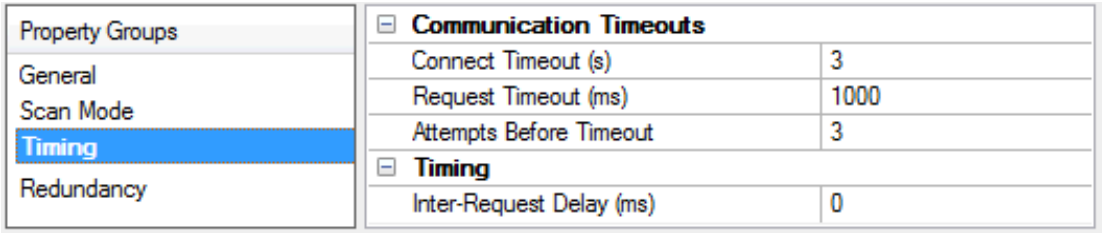
Scan Mode: Specify how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

- **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 — 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: Specify 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: Specify 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: Specify 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 — Tag Generation

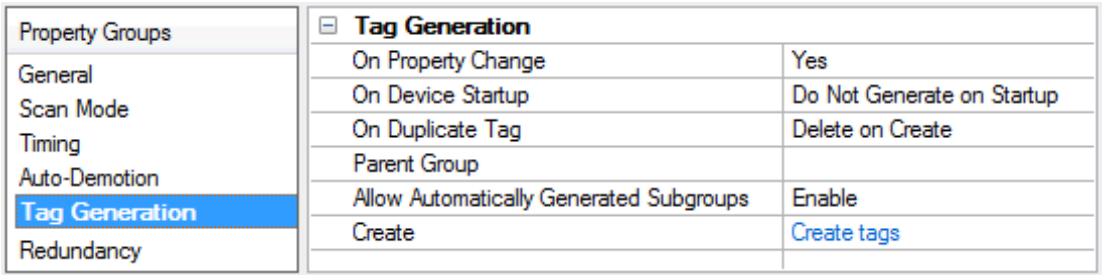
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the 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. To invoke via the Configuration API service, access /config/v1/project/channels/{name}/devices/{name}/services/TagGeneration.

On Device Startup: Specify 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.

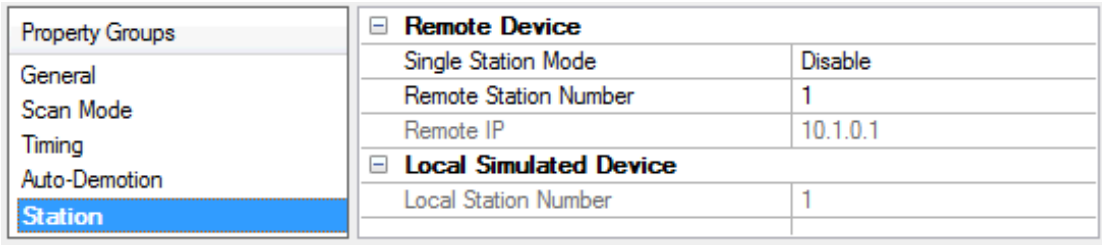
Note: Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags. Users should avoid adding tags to the server using names that may match tags that are automatically generated by the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



Remote Device Settings (UDR Client Mode)

The Remote Device Settings refer to the physical device on a network with which the driver device object communicates. They must be set when this device is set to operate in UDR client mode. The remote device settings will be disabled when UDR server mode is selected.

Single Station Mode

Enable this property if there is only one device on the network and it is configured to use the special single station mode station number. The default setting is disabled.

Remote Station Number

This property specifies the station number of the device to poll. Valid station numbers range from 0 to 15999. The default setting is 1.
Note: In some devices, the station number 0 has special significance. *For more information, refer to the device's documentation*.

Remote IP

This property specifies either the IP address of the device to poll or the IP address of a gateway device that gives access to the specified station. The default IP address is 10.1.0.1. *For more information, refer to* ***Networks***.

Local Simulated Device Settings (UDR Server Mode)

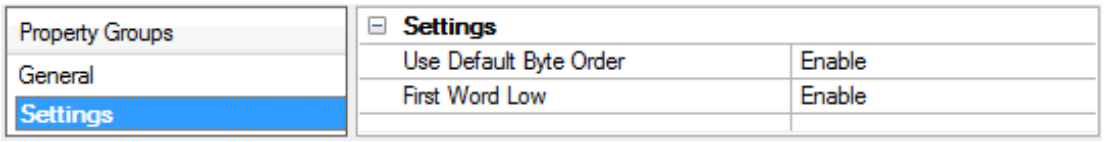
The Local Simulated Device Settings refer to a simulated I/O device created by the driver. They must be set when this device is set to operate in UDR server mode. When UDR client mode is selected, these settings are disabled.

Local Station Number

This property specifies a unique station number for the UDR server device. Valid station numbers range from 0 to 15999. The default setting is 1.

Device Properties — Settings

Multi-byte data can be stored in device registers using a number of byte and word order formats, depending on both the device programming and the format used by external devices. These settings can be used to specify the byte order.



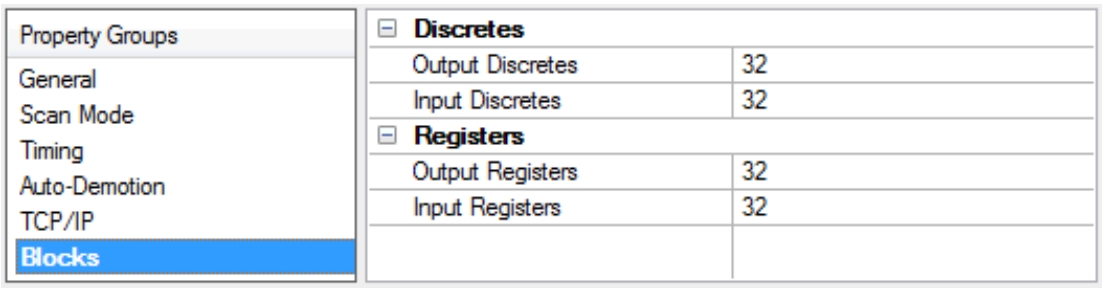
Use Default Byte Order

Data is stored in the device using Motorola (Big-Endian) byte order by default.

First Word Low in 32 Bit Data Types

Two consecutive 16-bit registers are used to store 32-bit values. By default, the register with the lower address will contain the low word. Byte and word order settings do not apply to 32-bit types.

Device Properties — Block Sizes



Digital Data

Digital data can be read from 8 to 800 points (bits) at a time. A higher block size means more points will be read from the device in a single request. Block size can be reduced if data needs to be read from non-contiguous locations within the device. Digital input and output block sizes can be set in steps of 8.

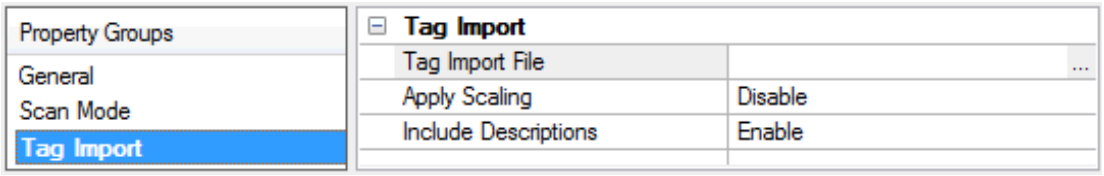
Analog Data

Analog data can be read from 1 to 120 locations (words) at a time. A higher block size means more register values will be read from the device in a single request. Block size can be reduced if data needs to be read from non-contiguous locations within the device.

Notes:

1. These settings refer to the binary representation of the data. If the ASCII Hex format is used, twice the specified number of bytes will be read. For more information, refer to **Communications Parameters**.
2. The request size for 32-bit types is half that for 16-bit types; meaning, it is the same number of bytes but half the number of registers. The request size for 32-bit types is specified in Device Properties.

Device Properties — Tag Import



Tag Import File

This property specifies the exact location of the SIXNET I/O Tool Kit tag export file that the driver should use when Automatic Tag Database Generation is enabled for the device.

Apply Scaling

Enable this property to have generated tags use the scaling settings provided in the import file. Scaling is applicable to analog input (AX) and outputs (AY) only. Linear scaling (with a scaled data type of float) and no clamping is assumed. These and other scale settings may be adjusted manually after tags are generated.

Include Descriptions

Enable this property to include the tag descriptions in generated tags. The tag descriptions are given in the export file.

Notes:

1. Although tags for multiple stations may be included in a single export file, the server must import tags for each of its devices individually. The driver selects a tag for import when the station number specified in the file is the same as the station number configured for the device. For more information, refer to **Station**.
2. f no tag name is specified in the export file, the driver will generate a name of the form "Unnamed_x", where x is an integer. For more information on configuring the automatic tag database generation feature (and how to create a tag import file) refer to **Automatic Tag Database Generation**.

UDR Server Devices

A device can be configured to operate in UDR server mode. Each UDR server device allocates memory for the following I/O:

I/O Type	Address Range	OPC Client Access	UDR Client Access
Digital Input	X0-X32767	Read/Write	Read/Write
Digital Output	Y0 -Y32767	Read/Write	Read/Write
Analog Input	AX0-AX32767	Read/Write	Read/Write
Analog Output	AY0-AY32767	Read/Write	Read/Write
Long Input	LX0-LX1023	Read/Write	Read/Write
Long Output	LY0-LY1023	Read/Write	Read/Write
Float Input	FX0-FX1023	Read/Write	Read/Write
Float Output	FY0-FY1023	Read/Write	Read/Write

Note: All memory is initialized to zero.

UDR server devices support the following UDR commands:

Command	Code (hex)	Description
GETD	0x0A	Read digital data (X, Y)
PUTD	0x0E	Write digital data (X, Y)
GETA	0x0C	Read analog data (AX, AY)
PUTA	0x10	Write analog data (AX, AY)
GETB	0x0B	Read byte data (LX, LY, FX, FY)
PUTB	0x0F	Write byte data (LX, LY, FX, FY)
IOXCHG	0x20	I/O Exchange (X, Y, AX, AY)

Notes:

- Digital data types may be 0 for inputs and 1 for outputs.
- Binary and Hex formats are supported, as well as CRC and non-CRC modes. Extended addresses (2-byte) may be used. Both data byte order options are supported.
- If a request can not be processed, due to invalid address or unsupported command for example, a NAK message will be sent.
- These commands are used automatically when "I/O transfers" and "Remote I/O Links" are configured in the controller. Users may also explicitly issue these commands from device programming and custom UDR drivers. In theses cases, refer to the "SIXNET Universal Protocol" document for protocol specification. This document comes with the SIXNET UDR Driver kit, which may be found on the support CD that came with the hardware.

Data Types Description

Data Type	Description
Boolean	Single bit
Word	Unsigned 16-bit value bit 0 is the low bit bit 15 is the high bit
Short	Signed 16-bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
BCD	Two byte packed BCD Value range is 0-9999. Behavior is undefined for values beyond this range.
DWord	Unsigned 32-bit value bit 0 is the low bit bit 31 is the high bit
Long	Signed 32-bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
LBCD	Four byte packed BCD Value range is 0-99999999. Behavior is undefined for values beyond this range.
Float	32-bit floating point value The driver interprets two consecutive registers as a floating-point value by making the second register the high word and the first register the low word.

Address Descriptions

Address specifications vary depending on the model in use. Select a link from the following list to obtain specific address information for the model of interest.

Client Open Model ddressing
Server Model Addressing

Client Open Model Addressing

Open Addressing allows a full range of addresses to accommodate a wide range of SIXNET devices. For the actual range available in the device, refer to the SIXNET documentation. The default data types for dynamically defined tags are shown in **bold**.

Address	Range	Data Type	Access
Digital Input	X0-X65535	**Boolean**	Read Only
Digital Output	Y0-Y65535	**Boolean**	Read/Write
Analog Input	AX0-AX65535 AX0-AX65534 AX0.0-AX0.15 ... AX65535.0-AX65535.15	Word, **Short**, BCD DWord, Long, Float, LBCD **Boolean**	Read Only
Analog Output	AY0-AY65535 AY0-AY65534 AY0.0-AY0.15 ... AY65535.0-AY65535.15	Word, **Short**, BCD DWord, Long, Float, LBCD **Boolean**	Read/Write
Long Input	LX0-LX65535	DWord, **Long**, Float, LBCD	Read Only
Long Output	LY0-LY65535	DWord, **Long**, Float, LBCD	Read/Write
Float Input	FX0-FX65535	DWord, Long, **Float**, LBCD	Read Only
Float Output	FY0-FY65535	DWord, Long, **Float**, LBCD	Read/Write

Array Support

Arrays are supported for all data types except Boolean. There are two methods of addressing an array. Examples are given using Analog Output memory locations.

AYxxxxx [rows] [cols]
AYxxxxx [cols] – (this method assumes "rows" is equal to one)

Rows multiplied by cols multiplied by data size in bytes cannot exceed the block size. The range of address represented by the array cannot exceed the valid address range of the device.

Note: Use caution when modifying 32-bit analog values (AX and AY with a data type of DWord, Long, LBCD or Float). Each address, for which these data types are allowed, starts at a word offset within the device. Therefore, DWords AX0 and AX1 overlap at word AX1. Thus, writing to AX0 will also modify the value held in AX1. It is recommended that users use these data types so that overlapping does not occur. As an example, when using DWords, use AX0, AX2, AX4 and so on in order to prevent overlapping Words.

See Also: *Block Sizes*

Server Model Addressing

UDR server addressing is for driver UDR server devices. The default data types for dynamically defined tags are shown in **bold**.

For more information, refer to Server Devices.

Address	Range	Data Type	Access
Digital Input	X0-X32767	**Boolean**	Read/Write
Digital Output	Y0-Y32767	**Boolean**	Read/Write
Analog Input	AX0-AX32767 AX0-AX32766 AX0.0-AX0.15 ... AX32767.0-AX32767.15	Word, **Short**, BCD DWord, Long, Float, LBCD **Boolean**	Read/Write
Analog Output	AY0-AY32767 AY0-AY32766 AY0.0-AY0.15 ... AY32767.0-AY32767.15	Word, **Short**, BCD DWord, Long, Float, LBCD **Boolean**	Read/Write
Long Input	LX0-LX1023	DWord, **Long**, Float, LBCD	Read/Write
Long Output	LY0-LY1023	DWord, **Long**, Float, LBCD	Read/Write
Float Input	FX0 -FX1023	DWord, Long, **Float**, LBCD	Read/Write
Float Output	FY0-FY1023	DWord, Long, **Float**, LBCD	Read/Write

Array Support

Arrays are supported for all data types except Boolean. There are two methods of addressing an array. Examples are given using Analog Output memory locations.

AYxxxxx [rows] [cols]
AYxxxxx [cols] – (this method assumes "rows" is equal to one)

Rows multiplied by cols multiplied by data size in bytes cannot exceed the block size. The range of address represented by the array cannot exceed the valid address range of the device.

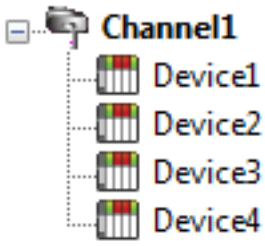
Note: Use caution when modifying 32-bit analog values (AX and AY with a data type of DWord, Long, LBCD or Float). Each address, for which these data types are allowed, starts at a word offset within the device. Therefore, DWords AX0 and AX1 overlap at word AX1. Thus, writing to AX0 will also modify the value held in AX1. It is recommended that users use these data types so that overlapping does not occur. As an example, when using DWords, use AX0, AX2, AX4 and so on in order to prevent overlapping Words.

See Also: *Block Sizes*

Optimizing Communications

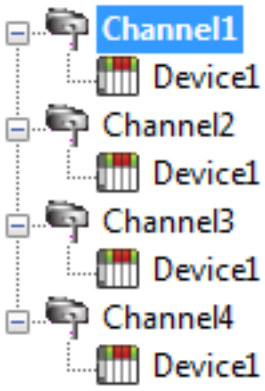
The SIXNET UDR Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the SIXNET UDR Driver is fast, there are a couple of guidelines that can be used to control and optimize the application and gain maximum performance.

Our server refers to communications protocols like SIXNET UDR Driver 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 then be defined under that channel. Each of these devices represents a single SIXNET UDR controller from which data will be collected. While this approach to defining the application will provide a high level of performance, it won't take full advantage of the SIXNET UDR Driver 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 SIXNET UDR Driver 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 SIXNET UDR Driver could only define one single channel, then the example shown above would be the only option available; however, the SIXNET UDR Driver can define up to 100 channels. Using multiple channels distributes 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 can be defined under its own channel. In this configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has 100 or fewer devices, it can be optimized exactly how it is shown here.

The performance will improve even if the application has more devices. While fewer devices may be ideal, the application will still benefit from additional channels. Although by spreading the device load across all channels will cause the server to move from device to device again, it can now do so with far less devices to process on a single channel.

The technique described above is generally applicable to Ethernet devices only because most serial drivers impose a limit of one channel per COM port. Such drivers have already optimized serial communications though single channels as much as possible. This driver is different because it bypasses the normal low level serial communications software components and uses components supplied by SIXNET. These components remove the one channel per COM port restriction. Though single channel per COM port performance will remain very good using these components, it is possible in some cases to improve overall performance by creating two or more channels per COM port, especially if high baud rates are used.

Users are not limited to a one-to-one server-device/physical-device relationship. If the device and network are fast enough, additional performance gains may be made by creating a duplicate device on another channel and then dividing tag blocks equally between them. Similarly, users are free to mix UDR server and client

devices on the same channel. Faster UDR server tag updates may result if UDR client and server devices are placed on separate channels.

Block Size, which is available on each defined device, can also affect the SIXNET UDR Driver performance. Block Size refers to the number of bytes that may be requested from a device at one time. To refine the driver's performance, configure Block Size to 1 to 120 registers and 8 to 800 bits. Unless highly scattered addresses are being read, larger block sizes generally result in better performance.

See Also: SIXNET Software Components: Universal Driver Components

Automatic Tag Database Generation

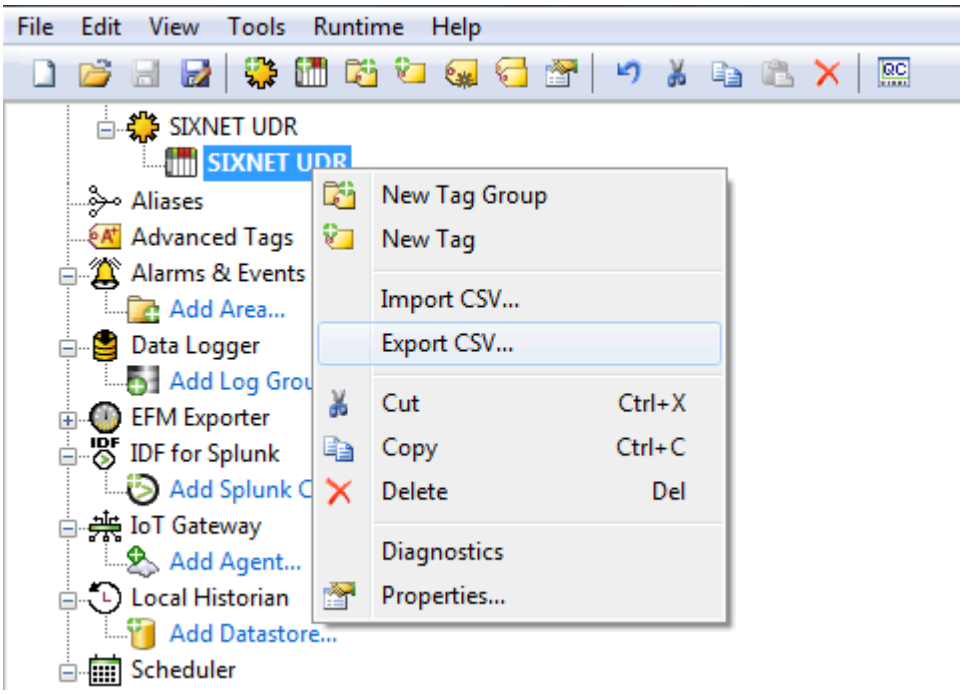
Overview

This driver makes use of the OPC Server's Automatic Tag Database Generation feature. This feature enables drivers to automatically create tags to access data used in a device configuration. Although it is sometimes possible to query a device for the information needed to build a tag database, this driver must use a **Tag Import File** instead. Tag import files can be generated using the SIXNET I/O Tool Kit device configuration application.

Creating the Tag Import File

The tag import file can be created from the SIXNET I/O Tool Kit application.

1. After a device configuration has been configured with the tool kit, right-click on the device and select **Export CSV...** to export the tag data.



2. Click **Browse...** to specify the destination of the export file. Choose **Mode to overwrite current file** or **Append new tags to existing file**.

3. Attaching a station prefix to the tag names largely depends on the application. A prefix can be separated from the tag name with an underscore or period. Since tag names with periods are not allowed by the OPC Server, the driver will automatically replace periods with underscores.

Note: The period separator is allowed here for compatibility with other applications.

4. For convenience, export the tags for all of the stations configured in the tool kit project into a single file. Tags will need to be imported into each OPC Server device individually.
5. Allow the tool kit to export unnamed tags. The driver will generate names for these tags during import. If planning to manually edit the export file, users may wish to have the tool kit sort the exported data. The driver does not require the data to be sorted.
6. Once all of the export options have been specified, click **Export** to create the file.

OPC Server Configuration

The automatic tag database generation feature is customizable. The primary control options can be set during the Database Creation step of the Device Wizard or later by selecting the Database Creation tab of the Device Properties. For more information, refer to the OPC Server's Help documentation.

The SIXNET UDR Driver requires specification of the tag import file's name and location in addition to these basic settings, which are common to all drivers that support automatic tag database generation. The tag name information can be specified during the Tag Import step of the Device Wizard or later by selecting the Tag Import tab of the Device Properties. For more information, refer to **Tag Import**.

Operation

Depending on the configuration, tag generation may either start automatically when the OPC Server project starts or be initiated manually at some other time. The OPC Server's event log will show when the tag generation process started, any errors that occurred while processing the variable import file and when the process completed.

Although tags for multiple stations may be included in a single export file, the server must import tags for each of its devices individually. The driver selects a tag for import when the station number specified in the file is the same as the station number configured for the device. For more information, refer to **Station**.

If no tag name is specified in the export file, the driver will generate a name of the form **Unnamed_x**, where x is an integer.

Error Descriptions

The following error/warning messages may be generated. Click on the link for a description of the message.

Address Validation

Missing address
Device address '<address>' contains a syntax error
Address '<address>' is out of range for the specified device or register
Data Type '<type>' is not valid for device address '<address>'
Device address '<address>' is Read Only
Array size is out of range for address '<address>'
Array support is not available for the specified address: '<address>'

Device Status Messages

Device '<device name>' is not responding
Unable to write to '<address>' on device '<device name>'

Driver Error Messages

Could not allocate memory for UDR server device '<device name>'
Failed to create UDR client SIXNET interface for channel '<channel>'
Failed to create SIXNET interface for UDR server device '<station number>' on channel '<channel>'
Failed to open UDR client session for channel '<channel>'
Failed to open session for UDR server device '<station number>' on channel '<channel>'
Failed to build request for device '<station number>' on channel '<channel>'
Failed to send request for device '<station number>' on channel '<channel>'
Failed to build ACK for device '<station number>' on channel '<channel>'
Failed to send ACK for device '<station number>' on channel '<channel>'
Failed to build NAK for device '<station number>' on channel '<channel>'
Failed to send NAK for device '<station number>' on channel '<channel>'

Automatic Tag Database Generation Messages

Tag import failed due to low memory resources
File exception encountered during tag import
Imported tag name changed from '<old name>' to '<new name>' (record: <record>)
Tag '<name>' (record: <record>) could not be imported due to name conflict
Tag not imported due to unknown I/O type (record: <record>)
Tag could not be imported due to unsupported data type (record: <record>)

Missing address

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically has no length.

Solution:
Re-enter the address in the client application.

Device address '<address>' contains a syntax error

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically contains one or more invalid characters.

Solution:
Re-enter the address in the client application.

Address <address>' is out of range for the specified device or register

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically references a location that is beyond the range of supported locations for the device.

Solution:
Verify that the address is correct; if not, re-enter it in the client application.

Data Type '<type>' is not valid for device address '<address>'

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically has been assigned an invalid data type.

Solution:
Modify the requested data type in the client application.

Device address '<address>' is Read Only

Error Type:
Warning

Possible Cause:

A tag address that has been specified dynamically has a requested access mode that is not compatible with what the device supports for that address.

Solution:
Change the access mode in the client application.

Array size is out of range for address '<address>'

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically is requesting an array size that is too large for the address type or block size of the driver.

Solution:
Re-enter the address in the client application to specify a smaller value for the array or a different starting point.

Array Support is not available for the specified address: '<address>'

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically contains an array reference for an address type that does not support arrays.

Solution:
Re-enter the address in the client application to remove the array reference or correct the address type.

Device <Device name>' is not responding

Error Type:
Serious

Possible Cause:

1. The connection between the device and the Host PC is broken.
2. The IP address assigned to the device is incorrect.
3. In some Sixnet models, this error message will be recieved if there is an attempt to access data out-side of the address memory range. In this situation, the server does not receive a NAK from the device and assumes that connection is lost.
4. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify the IP address given to the named device matches that of the actual device.
3. Verify that there is no attempt to ask for data outside the address memory range.
4. Increase the Request Timeout setting so that the entire response can be handled.

Unable to write to '<address>' on device '<device name>'

Error Type:
Serious

Possible Cause:

1. The connection between the device and the Host PC is broken.
2. The named device may have been assigned an incorrect IP address.

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify the IP address given to the named device matches that of the actual device.

Could not allocate memory for UDR server device '<device name>'

Error Type:
Warning

Possible Cause:
Low memory resources available on the Host PC.

Solution:
Shut down all unnecessary applications and retry.

Failed to create SIXNET interface for UDR client devices on channel '<channel>'

Error Type:
Warning

Possible Cause:
Low memory resources available on the host PC.

Solution:
Shut down all unnecessary applications and retry.

Failed to create SIXNET interface for UDR server device '<station number>' on channel '<channel>'

Error Type:

Warning

Possible Cause:
Low memory resources available on the host PC.

Solution:
Shut down all unnecessary applications and retry.

Failed to open session for devices on channel '<channel>'

Error Type:
Warning

Possible Cause:

1. Session limit exceeded.
2. Invalid COM port or Ethernet adapter specified.
3. Low level SIXNET communication component Six32com.exe or Udrcom32.dll not found.
4. Low computer resources.

Solution:

1. SIXNET software components used by this driver and SIXNET applications limit the number of sessions that can be open a time.
2. Reduce the number of sessions used.
3. Verify specified COM port or Ethernet adapter card exist on the host PC.
4. Make sure Six32com.exe and Udrcom32.dll are present in the system folder.
5. Shut down all unnecessary applications and retry.

See Also:
SIXNET Software Components: Universal Driver Components

Failed to open session for UDR server device '<station number>' on channel '<channel>'

Error Type:
Warning

Possible Cause:

1. Session limit exceeded.
2. Invalid COM port or Ethernet adapter specified.
3. Low level SIXNET communication component Six32com.exe or Udrcom32.dll not found.
4. Low computer resources.

Solution:

1. SIXNET software components used by this driver and SIXNET applications limit the number of sessions that can be open a time.
2. Reduce the number of sessions used.
3. Verify specified COM port or Ethernet adapter card exist on the host PC.
4. Make sure Six32com.exe and Udrcom32.dll are present in the system folder.
5. Shut down all unnecessary applications and retry.

See Also:

SIXNET Software Components: Universal Driver Components

Failed to build request for device '<station number>' on channel '<channel>'

Error Type:

Warning

Possible Cause:

SIXNET communication software component could not construct UDR (Universal DRiver protocol) request from data provided by driver. This problem occurred when attempting to build a read or write request for a UDR client device object.

Solution:

Document the precise circumstances that lead to the error and then contact Technical Support.

Failed to send request for device '<station number>' on channel '<channel>'

Error Type:

Warning

Possible Cause:

SIXNET communication software component could not processes a send request from data provided by driver. This problem occurred when attempting to send a read or write request for a UDR client device object.

Solution:

Document the precise circumstances that lead to the error and then contact Technical Support.

Failed to build ACK for device '<station number>' on channel '<channel>'

Error Type:

Warning

Possible Cause:

SIXNET communication software component could not construct UDR (Universal DRiver protocol) request from data provided by driver. This problem occurred when attempting to build a acknowledged (ACK) response to a successfully processed unsolicited message.

Solution:
Document the precise circumstances that lead to the error and then contact Technical Support.

Failed to send ACK for device '<station number>' on channel '<channel>'

Error Type:
Warning

Possible Cause:
SIXNET communication software component could not processes a send request from data provided by driver. This problem occurred when attempting to send a acknowledged (ACK) response to a successfully processed unsolicited message.

Solution:
Document the precise circumstances that lead to the error and then contact Technical Support.

Failed to build NAK for device '<station number>' on channel '<channel>'

Error Type:
Warning

Possible Cause:
SIXNET communication software component could not construct UDR (Universal DRiver protocol) request from data provided by driver. This problem occurred when attempting to build a not-acknowledged (NAK) response to an unsolicited message.

Solution:
Document the precise circumstances that lead to the error and then contact Technical Support.

Failed to send NAK for device '<station number>' on channel '<channel>'

Error Type:
Warning

Possible Cause:
SIXNET communication software component could not processes a send request from data provided by driver. This problem occurred when attempting to build a not-acknowledged (NAK) response to an unsolicited message.

Solution:
Document the precise circumstances that lead to the error and then contact Technical Support.

Tag import failed due to low memory resources

Error Type:
Serious

Possible Cause:
The driver could not allocate memory required to process tag import file.

Solution:
Shutdown all unnecessary applications and retry.

File exception encountered during tag import

Error Type:
Serious

Possible Cause:
The tag import file could not be read.

Solution:
Regenerate the tag import file and retry.

Imported tag name changed from '<old name>' to '<new name>' (record: <record>)

Error Type:
Warning

Possible Cause:
The tag name encountered in the tag import file contained invalid characters.

Solution:
The driver will construct a valid name based on the one from the tag import file. To prevent this error in the future, and to maintain name consistency, change the tag name in the device configuration if possible.

Tag '<name>' (record: <record>) could not be imported due to name conflict

Error Type:
Warning

Possible Cause:
The tag name encountered in the tag import file contained invalid characters. The driver could not modify the name in such a way that is would be unique and valid.

Solution:
Change the tag name in the device configuration if possible, change the name in the export file using a text editor or manually define the tag in the OPC server.

Tag not imported due to unknown I/O type (record: <record>)

Error Type:
Warning

Possible Cause:

The driver recognizes the following I/O types only:

0 (Analog Input)
1 (Analog Output)
10 (Digital Input)
11 (Digital Output)
20 (Long Input)
21 (Long Output)
22 (Float Input)
23 (Float Output)

Solution:
Correct the record in the export file if in error.

Tag could not be imported due to unsupported data type (record: <record>)

Error Type:
Warning

Possible Cause:
The driver recognizes the following I/O types only:

1. Discrete (bit)
2. Short (signed 16-bit integer)
3. Ushort (unsigned 16-bit integer)
4. Long (signed 32-bit integer)
5. Ulong (unsigned 32-bit integer)
6. Float (32-bit IEEE floating point)

Solution:
Correct the record in the export file if in error or choose an alternate, compatible data type.

Index

1

3

A

B

C

D

E

F

G

I

M

N

O

P

R

S

T

U

X