SIXNET EtherTRAK Driver

Table of Contents

SIXNET EtherTRAK Driver

Help version 1.016

CONTENTS

Overview

The SIXNET EtherTRAK Driver provides a reliable way to connect SIXNET EtherTRAK devices to OPC client applications; including HMI, SCADA, Historian, MES, ERP and countless custom applications. It is intended for use with SIXNET EtherTRAK devices that support the Modbus Open TCP/UDP protocol. This driver utilizes UDP socket communications to provide maximum performance with minimal overhead.

Setup

Supported Devices

SIXNET EtherTRAK I/O modules (firmware version 2.10 or later)
SIXNET RemoteTRAK I/O connected through an EtherTRAK I/O module.*
SIXNET VersaTRAK RTUs (firmware version 2.12 or later)
SIXNET SIXTRAK gateways (firmware version 2.12 or later)

*Both the RemoteTRAK and EtherTRAK must have firmware version 2.01 or later.

Communication Protocol

Modbus Open Protocol over Ethernet using Winsock V1.1 or higher.

Maximum Number of Channels

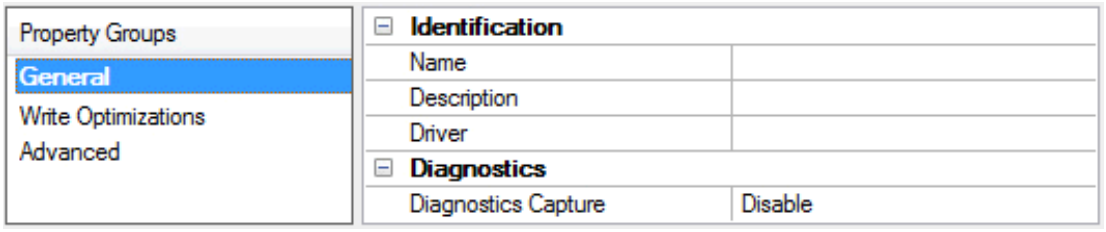
The maximum number of supported channels is 100.

Device ID (EtherTRAK IP Network Address Without RemoteTRAK RS-485 Bridging)

SIXNET EtherTRAK devices are networked using standard IP addressing. Determine and set the IP of the SIXNET EtherTRAK modules using the SIXNET Remote IO Toolkit. In general, the Device ID has the following format: YYY.YYY.YYY.YYY, where YYY designates the device IP address. Each YYY byte should be in the range of 0 to 255. If intending to address RemoteTRAK modules hung from the SIXNET EtherTRAK module's RS-485 port, refer to **Addressing RS-485 RemoteTRAK Devices Over Ethernet**.

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.
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.

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

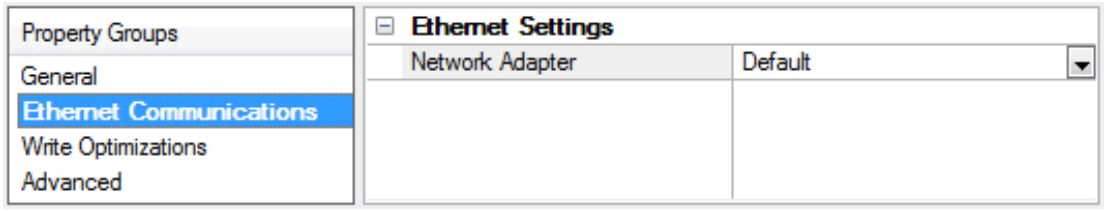
Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is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

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

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

Channel Properties — Ethernet Communications

Ethernet Communication can be used to communicate with devices.

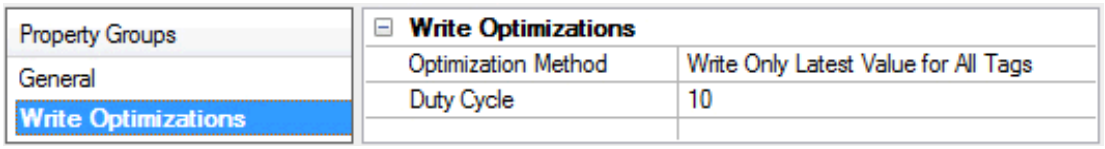


Ethernet Settings

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

Channel Properties — Write Optimizations

As with any OPC 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.

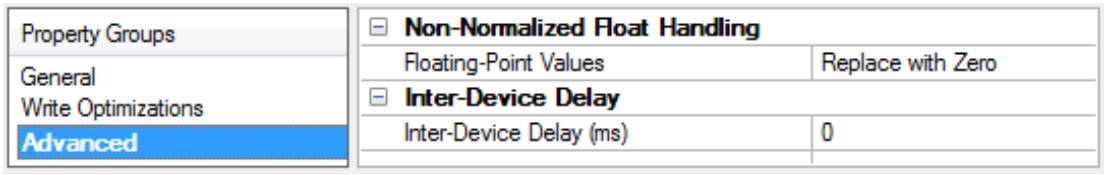
  Note: This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second 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.

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 disabled 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.

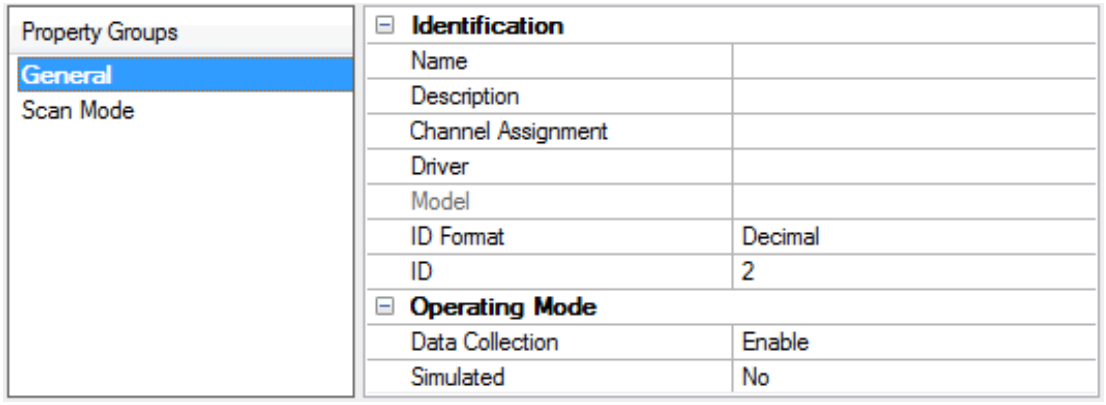
For more information on the floating point values, refer to "How To ... Work with Non-Normalized Floating Point Values" in the server help.

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.

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. This property specifies the driver selected during channel creation. It is disabled in the channel properties.

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 station / node / identity / address. The type of ID entered depends on the communications driver being used. For many 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 ID format can be Decimal, Octal, and Hexadecimal. 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.

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.

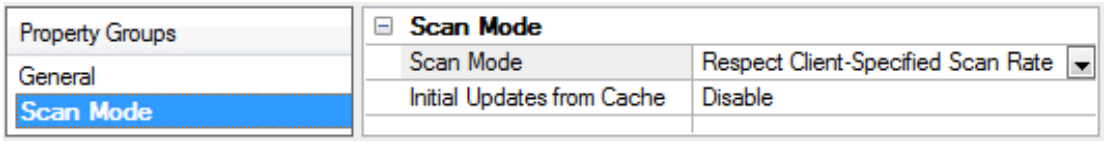
Notes:

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.



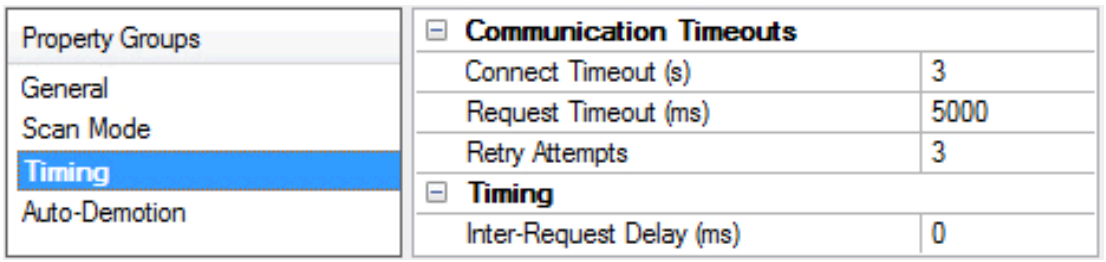
Scan Mode: specifies how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

- **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 maximum scan rate to be used. 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: 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.

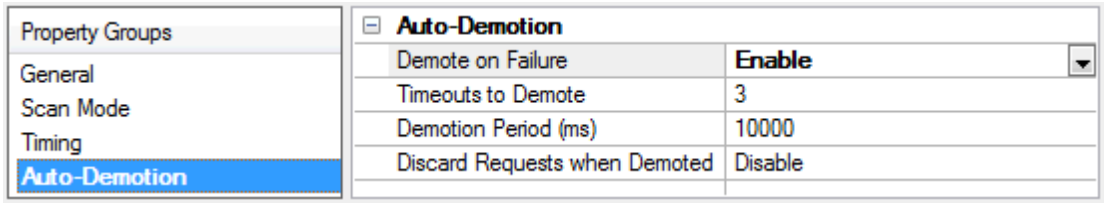
Timing

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 — 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.

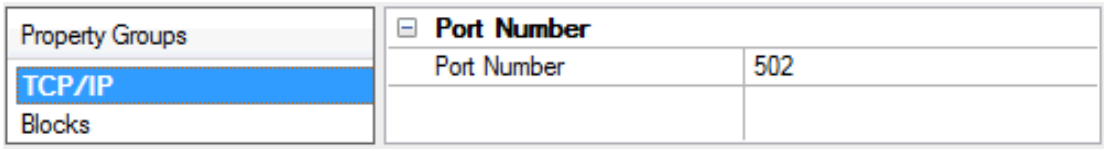
Tip: Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

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.

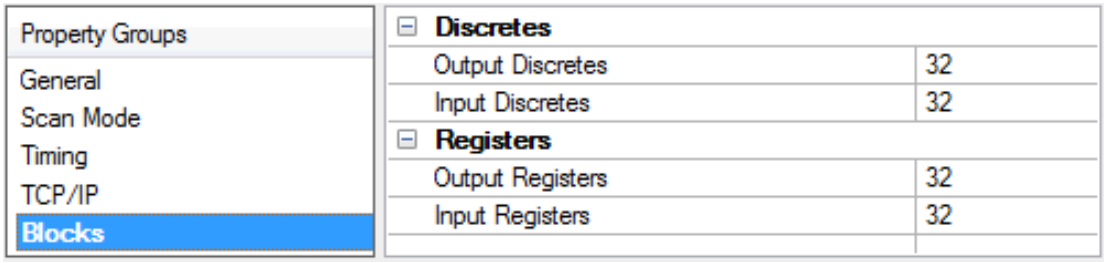
Discard Requests when Demoted: Select whether or not write requests should be attempted during the off-scan period. Disable to always send write requests regardless of the demotion period. Enable to discard writes; the server automatically fails any write request received from a client and does not post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

Device Properties - TCP/IP



Port Number: Specify the TCP/IP port number that the remote device is configured to use. The default setting is 502.

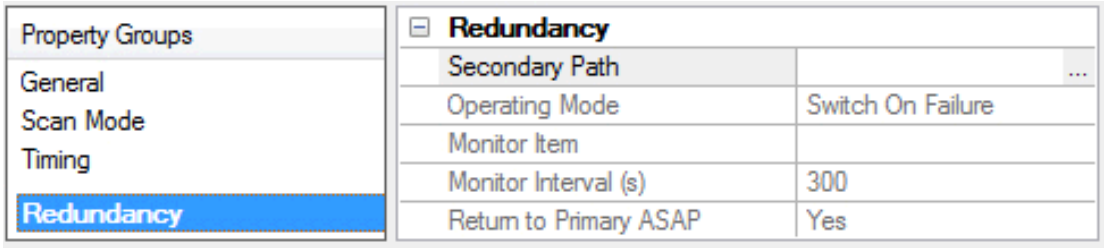
Device Properties - Blocks



Discretes: Coils 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. If data needs to be read from non-contiguous locations within the device, the block size can be reduced. The default setting is 32.

Registers: Registers 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. If data needs to be read from non-contiguous locations within the device, the block size can be reduced. The default setting is 32.

Device Properties — Redundancy

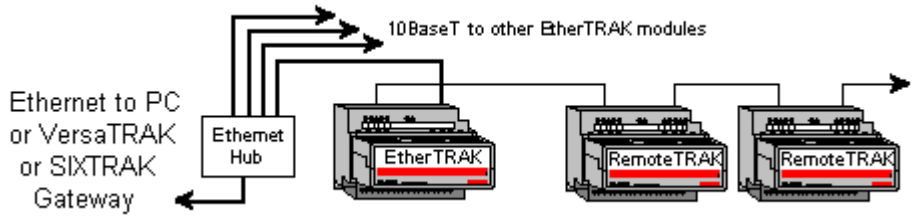


Redundancy is available with the Media-Level Redundancy Plug-In.

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

Addressing RS-485 RemoteTRAK Devices Over the Ethernet

The SIXNET EtherTRAK Driver can address RemoteTRAK I/O connected on a RS485 party-line by connecting through the SIXNET ET-GT-485-1 Ethernet to Modbus RS485 gateway. Any SIXNET EtherTRAK I/O module messages are passed through to the RS485 port.



The intelligent Modbus interfaces accept Modbus Ethernet commands and convert them to traditional Modbus messages. They then pass the command along to the RS485 port on the Ethernet interface. Replies from the station are returned to Modbus Ethernet format and passed back on the Ethernet network to the SIXNET EtherTRAK OPC server.

Device ID (EtherTRAK IP Network Address with RemoteTRAK RS-485 Bridging)

SIXNET EtherTRAK devices are networked using standard IP addressing. Users can determine or set the IP of the SIXNET EtherTRAK modules using the SIXNET Remote IO Toolkit. In general, the Device ID has the following format YYY.YYY.YYY.YYY, where YYY designates the device IP address. Each YYY byte should be in the range of 0 to 255.

When addressing RemoteTRAK devices via the SIXNET EtherTRAK module's RS-485 port, include the station number of the RemoteTRAK device as part of the SIXNET EtherTRAK IP address. Using the same format as the IP address, adding the RemoteTRAK station number would take the following format: YYY.YYY.YYY.YYY.ZZZ. The normal IP address remains the same as denoted by the YYY.YYY.YYY.YYY; however, the station number of the desired RemoteTRAK unit on the SIXNET EtherTRAK module's RS-485 port is denoted by the .ZZZ. The valid station number range for ZZZ is 1 to 247.

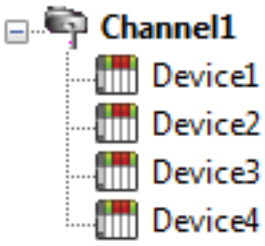
Example

Assume that the SIXNET EtherTRAK module is at IP address 10.1.1.10. To this EtherTRAK module, the user desires to attach four RemoteTRAK modules that have already been configured with Modbus Station numbers of 1, 2, 3 and 4. In the SIXNET EtherTRAK OPC Server, the user would add four separate devices to the SIXNET EtherTRAK project. The first device would have a Device ID of 10.1.1.10.1; the second 10.1.1.10.2; the third 10.1.1.10.3; the forth 10.1.1.10.4. Thus, although each Device ID has the same IP address, the last field contains the actual station number of each RemoteTRAK device attached to the RS-485 port.

Optimizing Communications

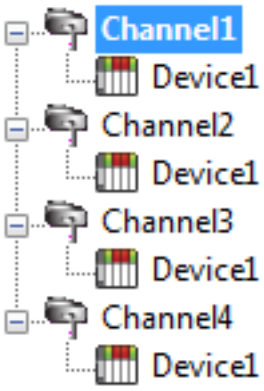
The SIXNET EtherTRAK Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the SIXNET EtherTRAK Driver is fast, there are a couple of guidelines that can be used in order to control and optimize the application and gain maximum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

Our server refers to communications protocols like EtherTRAK 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 EtherTRAK I/O module 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 EtherTRAK 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 EtherTRAK Driver channel. In this configuration, the driver must move from one device to the next as quickly as possible in order 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 EtherTRAK Driver could only define one single channel, then the example shown above would be the only option available; however, the SIXNET EtherTRAK 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 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 than channels, it can be optimized exactly as it is shown here.

The performance will improve even if the application has more devices than channels. While one device per channel is ideal, the application will still benefit from additional channels. Although 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.

Block Size, which is available on each defined device, can also affect the SIXNET EtherTRAK Driver's performance. Block Size refers to the number of bytes that may be requested from a device at one time. To refine the performance of this driver, configure Block Size from 1 to 120 registers per request. The coil block size can be adjusted from 8 to 800. Increase the block size setting for the device if the application consists of large requests for consecutively ordered data.

Data Types Description

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8 bit value bit 0 is the low bit bit 7 is the high bit
Char	Signed 8 bit value bit 0 is the low bit bit 6 is the high bit bit 7 is the sign 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
DWord	Unsigned 32 bit value bit 0 is the low bit bit 31 is the high bit
DWord Example	If register 40001 is specified as a dword, bit 0 of register 40001 would be bit 0 of the 32 bit data type and bit 15 of register 40002 would be bit 31 of the 32 bit data type. The reverse is true when this is not selected.
Long	Signed 32 bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
Long Example	If register 40001 is specified as a long, bit 0 of register 40001 would be bit 0 of the 32 bit data type and bit 15 of register 40002 would be bit 31 of the 32 bit data type. The reverse is true when this is not selected.
BCD	Two byte packed BCD Value range is 0-9999. Behavior is undefined for values beyond this range.
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

Data Type	Description
	register the high word and the first register the low word.
Float Example	If register 40001 is specified as a float, bit 0 of register 40001 would be bit 0 of the 32 bit word, and bit 15 of register 40002 would be bit 31 of the 32 bit word.

Note: The descriptions above assume first word low data handling of 32 bit data types.

Address Descriptions

Access to I/O of different types is supported by the Modbus messaging protocol via address ranges. To access the following SIXNET data types, use the following Modbus address ranges.

SIXNET Data Type	SIXNET Address	Modbus Data Address
Discrete In:	0 .. 9999	10001 .. 20000
Discrete Out:	0 .. 9999	00001 .. 10000
Analog In:	0 .. 2999	30001 .. 33000
Analog Out:	0 .. 2999	40001 .. 43000
Short Integer In	0 .. 1999	33001 .. 35000
Long Integer In	0 .. 1999	35001 .. 37000
Floating Point In:	0 .. 2999	37001 .. 40000
Short Integer Out:	0 .. 1999	43001 .. 45000
Long Integer Out:	0 .. 1999	45001 .. 47000
Floating Point Out:	0 .. 2999	47001 .. 50000

Examples

1. Modbus data address 10001 equates to SIXNET discrete input 0.
2. Modbus data address 30006 equates to SIXNET analog input 5.

Generic Modbus Addressing Decimal Format

The default data types for dynamically defined tags are shown in **bold**.

Address	Range	Data Type	Access
Output Coils [Function Codes (decimal): 01, 05, 15]	000001-065536	**Boolean**, Byte, Char, Word, Short*	Read/Write
Input Coils [Function Code (decimal): 02]	100001-165536	**Boolean**, Byte, Char, Word, Short*	Read Only
Internal Registers [Function Code (decimal): 04]	300001-365536 300001-365535 3xxxxx.0- 3xxxxx.15	**Word**, Short, BCD Float, DWord, Long, LBCD Boolean	Read Only
Holding Registers	400001-465536 400001-465535	**Word**, Short, BCD Float, DWord, Long, LBCD	Read/Write

Address	Range	Data Type	Access
[Function Codes (decimal): 03, 06, 16]	4xxxxx.0-4xxxxx.15	Boolean	

*When accessing coils as a byte or char, the address specified must lie on a byte boundary (such as xxxxx1, xxxxx9, xxxxx17 and so forth). When accessed as a word or short, the address specified must lie on a word boundary (such as xxxxx1, xxxxx17, xxxx33 and so forth).

Examples
1. To access SIXNET Discrete Output 0, enter a Modbus address of 00001.
2. To access SIXNET Analog In 3, enter a Modbus address of 30004.
3. To access SIXNET Analog Out 2, enter a Modbus address of 40003.

Array Support

Arrays are supported for internal and holding register locations for all data types except for Boolean. There are two methods of addressing an array. Examples are given using holding register locations.

4xxxx [rows] [cols]
4xxxx [cols] this method assumes rows is equal to one.

Rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register type. For arrays of 32 bit data types, rows multiplied by cols multiplied by 2 cannot exceed the block size.

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

Winsock initialization failed (OS Error = n)
Winsock V1.1 or higher must be installed to use the SIXNET EtherTRAK device driver

Device Specific Messages

Bad address in block [<start address> to <end address>] on device '<device name>'
Block size mismatch reading [<start address> to <end address>] on device '<device name>'
Block request [<start address> to <end address>] on device '<device name>.' responded with exception = n

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 it is 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 doesn't 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 serial connection between the device and the Host PC is broken.
2. The communications properties for the serial connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.
4. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device property.

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications properties match those of the device.
3. Verify that the Network ID given to the named device matches that of the actual device.
4. Increase the Request Timeout property so that the entire response can be handled.

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

Error Type:
Serious

Possible Cause:

1. The serial connection between the device and the Host PC is broken.
2. The communications properties for the serial connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications properties match those of the device.
3. Verify that the Network ID given to the named device matches that of the actual device.

Winsock initialization failed (OS Error = n)

Error Type:
Fatal

OS Error	Indication	Possible Solution
10091	Indicates that the underlying network subsystem is not ready for network communication.	Wait a few seconds and restart the driver.
10067	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 the SIXNET EtherTRAK device driver

Error Type:
Fatal

Possible Cause:
The version number of the Winsock DLL found on the system is less than 1.1.

Solution:
Upgrade Winsock to version 1.1 or higher.

Bad address in block [<start address> to <end address>] on device '<device name>'

Error Type:
Serious

Possible Cause:
An attempt has been made to reference a nonexistent location in the specified device.

Solution:
Verify the tags assigned to addresses in the specified range on the device and eliminate ones that reference invalid locations.

Block size mismatch reading [<start address> to <end address>] on device '<device name>

Error Type:
Fatal for addresses falling in this block.

Possible Cause:
The driver attempted to read a block of memory in the PLC. The PLC responded with no error, but did not provide the driver with the requested block size of data.

Solution:
Ensure that the range of memory exists for the PLC.

Block request [<start address> to <end address>] on device '<device name>' responded with exception = n

Error Type:
Fatal for addresses falling in this block.

Possible Cause:
The driver attempted to read a block of memory in the PLC. The PLC responded with the exception error shown.

Solution:
Ensure that the range of memory or data type exists for the PLC.

Index

A

B

C

D

E

F

I

L

M

N

O

R

S

T

U

W