# SIXNET UDR Driver Help

# Table of Contents

# SIXNET UDR Driver Help

Help version 1.016

## CONTENTS



## Overview

The SIXNET UDR Driver provides an easy and 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 slave 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 slave) and unsolicited communication (where the I/P device in the field is the master) can be performed simultaneously. In other words, master and slave operation may occur simultaneously over separate channels in the same SIXNET driver. For more information, refer to **Slave Devices**.

## Device Setup

### 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 in order 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 **Driver Setup**.

### Device IDs

This driver is limited to 8192 devices. The number of these devices that may operate as slaves is limited to 64 per COM port plus 64 over Ethernet. For more information, refer to **SIXNET Software Components-Universal Driver Components**.

### Device Configuration

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 Ether-TRAK unit connected to a network of RemoteTRAK units on an RS-485 party line, must be configured to operate in pass-thru mode.

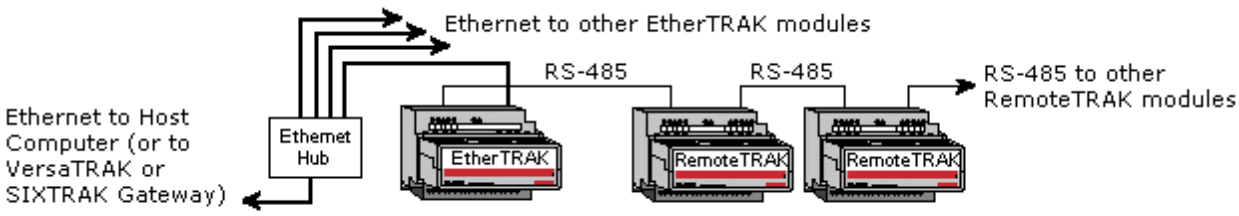
Controller and RTU units may be programmed to send unsolicited requests to the driver's slave devices, although not all UDR commands are supported. For more information, refer to **Slave Devices**.

### 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**.

## 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.

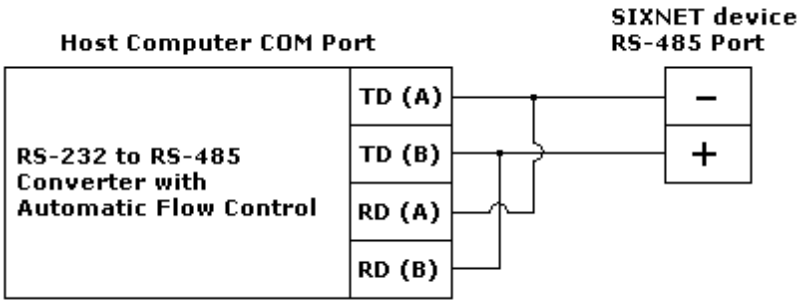
Since the Host PC is directly connected to the Ethernet network in this example and not to the RS-485 serial line, the channel must be configured to use Ethernet. For more information, refer to **Communications Parameters**.

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.

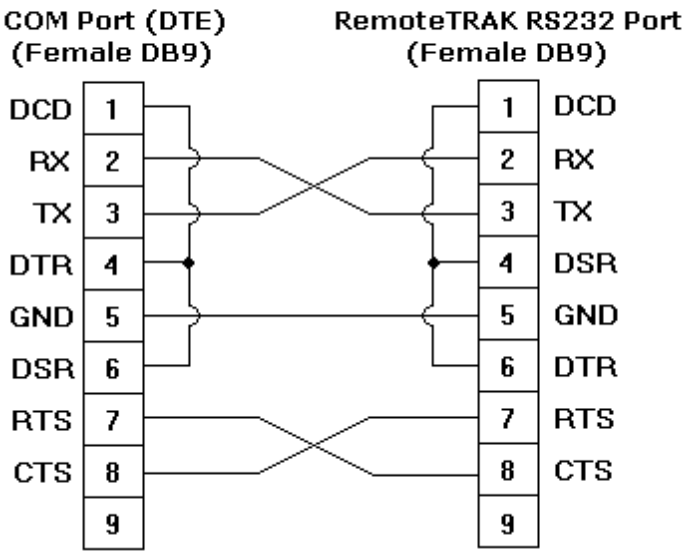
**See Also:** Station

## Cable Diagrams

### Serial Connections




ST-CABLE-PF Cable for PC COM Port
to RM-232-SETUP or RM-232-485-4U




### Ethernet Connections

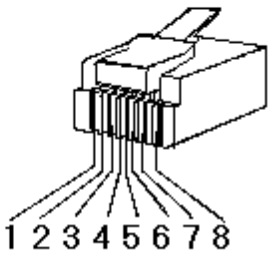
Patch Cable (Straight Through)

| | | | | |
|---|---|---|---|---|
| TD + 1 | OR/WHT | OR/WHT | 1 | TD + |
| TD - 2 | OR | OR | 2 | TD - |
| RD + 3 | GRN/WHT | GRN/WHT | 3 | RD + |
| 4 | BLU | BLU | 4 | |
| 5 | BLU/WHT | BLU/WHT | 5 | |
| RD - 6 | GRN | GRN | 6 | RD - |
| 7 | BRN/WHT | BRN/WHT | 7 | |
| 8 | BRN | BRN | 8 | |
| RJ45 | | | RJ45 | |

Crossover Cable

| | | | | |
|---|---|---|---|---|
| TD + 1 | OR/WHT | GRN/WHT | 1 | TD + |
| TD - 2 | OR | GRN | 2 | TD - |
| RD + 3 | GRN/WHT | OR/WHT | 3 | RD + |
| 4 | BLU | BLU | 4 | |
| 5 | BLU/WHT | BLU/WHT | 5 | |
| RD - 6 | GRN | OR | 6 | RD - |
| 7 | BRN/WHT | BRN/WHT | 7 | |
| 8 | BRN | BRN | 8 | |
| RJ45 | | | RJ45 | |

10 BaseT




8-pin RJ45

## Driver 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**.

**Note 1:** For driver-specific channel and device properties, refer to **Channel Properties** and **Device Properties**.

**Note 2:** For recommendations on optimizing configuration, refer to **Optimizing Your SIXNET UDR Communications**.

## SIXNET Software Components-Universal Driver Components

### Overview

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 slave device. Channel sessions will not be started until the scanning of tags belonging to its master devices has begun. Likewise, slave 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 slave 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.

## Channel Properties

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. For more information on this driver's specific Channel Properties, refer to **Communications Parameters**.

**Note:** Up to 32 channels can be created, including multiple channels that connect to the same COM port or Ethernet adapter.

## Communications Parameters

### Use Ethernet

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

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

### Flow Control
This parameter 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

Each physical device to be polled must be represented by a device object in the server. Slave device objects may be created to act as virtual I/O devices on the network. To enable Slave mode, click the **Device Properties** | **General** and then select the **Slave Model**. The following topics describe how to configure device objects:



**Note:** This driver is limited to 8192 devices. The number of these devices that may operate as slaves is limited to 64 per COM port plus 64 over Ethernet. For more information, refer to **SIXNET Software Components-Universal Driver Components**.

## Station

### Remote Device Settings (Master 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 master mode. The remote device settings will be disabled when slave mode is selected.

### Single Station Mode
Check this box if there is only one device on the network and it is configured to use the special single station mode station number. This box is unchecked by default.

#### Station Number
This parameter 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 Help documentation.

**IP**
This parameter 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 (Slave 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 slave mode. When master mode is selected, these settings will be disabled.

**Station Number**
This parameter specifies a unique station number for the slave device. Valid station numbers range from 0 to 15999. The default setting is 1.

## 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.

## 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.

**Note 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**.

**Note 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.

## Tag Import
### Tag Import File
This parameter 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
Check this box 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
Check this box to include the tag descriptions in generated tags. The tag descriptions are given in the export file.

**Note 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**.

**Note 2:** 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. 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**.

## Slave Devices

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

| I/O Type | Address Range | Client Access | Master 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.

Slave 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:**

1. Digital data types may be 0 for inputs and 1 for outputs.

2. 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.

3. If a request can not be processed, due to invalid address or unsupported command for example, a NAK message will be sent.

4. 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<br><br>bit 0 is the low bit<br>bit 15 is the high bit |
| Short | Signed 16 bit value<br><br>bit 0 is the low bit<br>bit 14 is the high bit<br>bit 15 is the sign bit |
| BCD | Two byte packed BCD<br><br>Value range is 0-9999. Behavior is undefined for values beyond this range. |
| DWord | Unsigned 32 bit value<br><br>bit 0 is the low bit<br>bit 31 is the high bit |
| Long | Signed 32 bit value<br><br>bit 0 is the low bit<br>bit 30 is the high bit<br>bit 31 is the sign bit |
| LBCD | Four byte packed BCD<br><br>Value range is 0-99999999. Behavior is undefined for values beyond this range. |
| Float | 32 bit floating point value<br><br>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.

**Master Open Addressing**
**Slave Addressing**

## Master Open 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 Help 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<br><br>AX0-AX65534<br><br>AX0.0-AX0.15 ... AX65535.0-AX65535.15 | Word, **Short**, BCD<br><br>DWord, Long, Float, LBCD<br><br>**Boolean** | Read Only |
| Analog Output | AY0-AY65535<br><br>AY0-AY65534<br><br>AY0.0-AY0.15 ... AY65535.0-AY65535.15 | Word, **Short**, BCD<br><br>DWord, Long, Float, LBCD<br><br>**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**

## Slave Addressing

Slave Addressing is for driver slave devices. The default data types for dynamically defined tags are shown in **bold**.

**Note:** For more information, refer to **Slave Devices**.

| Address | Range | Data Type | Access |
|---|---|---|---|
| Digital Input | X0-X32767 | **Boolean** | Read/Write |
| Digital Output | Y0-Y32767 | **Boolean** | Read/Write |
| Analog Input | AX0-AX32767<br><br>AX0-AX32766 | Word, **Short**, BCD<br><br>DWord, Long, Float, LBCD | Read/Write |

| | | | |
|---|---|---|---|
| | AX0.0-AX0.15 ... AX32767.0-AX32767.15 | **Boolean** | |
| Analog Output | AY0-AY32767<br><br>AY0-AY32766<br><br>AY0.0-AY0.15 ... AY32767.0-AY32767.15 | Word, **Short**, BCD<br><br>DWord, Long, Float, LBCD<br><br>**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.

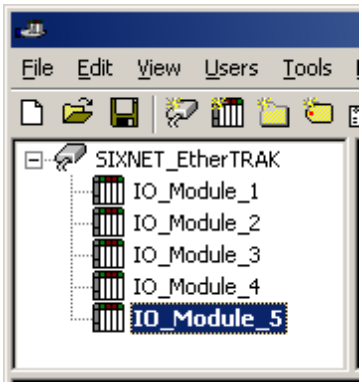
**Note:** Use caution when modifying 32 bit analog values (AX and AY with a data type of DWord, Long, LBCD or Float). Each address, for which these data types are allowed, starts at a word offset within the device. Therefore, DWords AX0 and AX1 overlap at word AX1. Thus, writing to AX0 will also modify 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 Your SIXNET UDR Communications

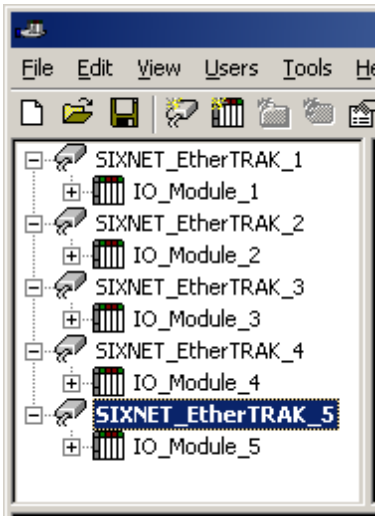
The SIXNET UDR driver driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the SIXNET UDR driver driver is fast, there are a couple of guidelines that can be used in order to control and optimize the application and gain maximum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

Our server refers to communications protocols like 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 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 UDR driver driver could only define one single channel, then the example shown above would be the only option available; however, the SIXNET UDR driver driver can define up to 32 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 32 or fewer devices, it can be optimized exactly how it is shown here.

The performance will improve even if the application has more than 32 devices. While 32 or 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 slave and master devices on the same channel, faster slave tag updates may result if master and slave devices are placed on separate channels.

Block Size, which is available on each defined device, can also affect the SIXNET UDR driver's 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

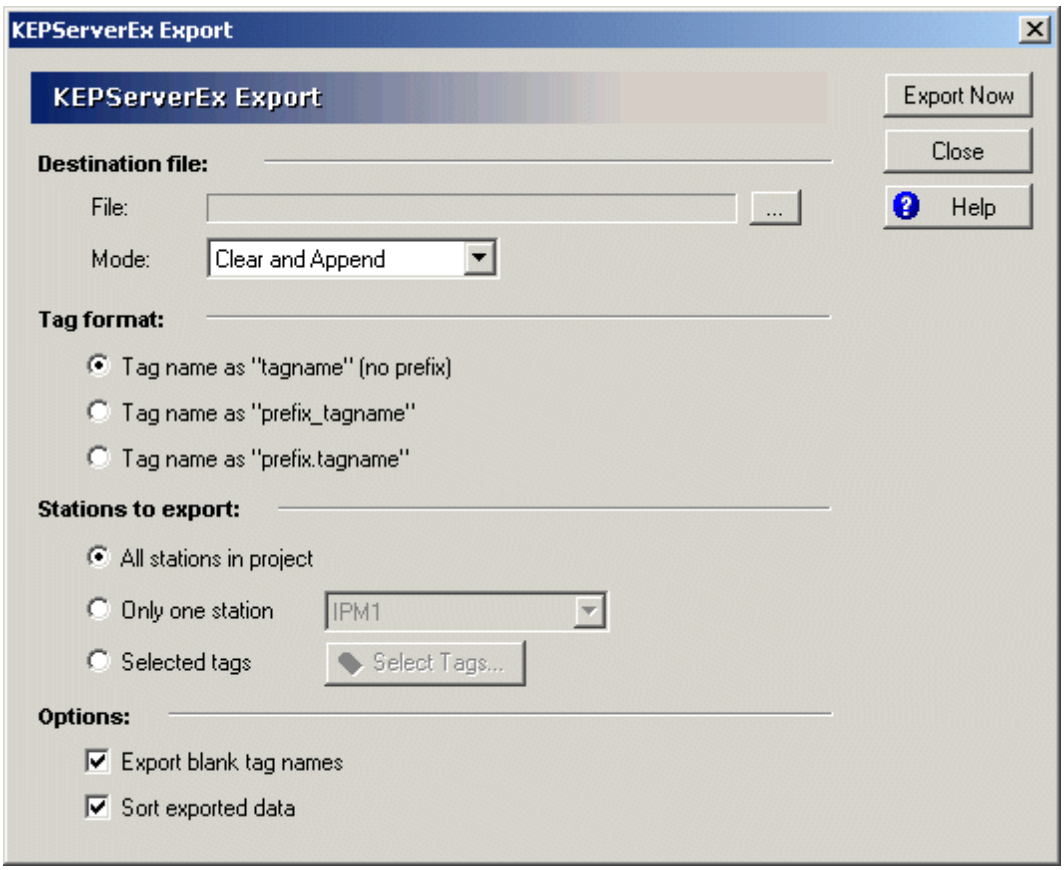
### Overview

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, select **File** | **Export**. Then, select **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 slave device '<device name>'
Failed to create master SIXNET interface for channel '<channel>'
Failed to create SIXNET interface for slave device '<station number>' on channel '<channel>'
Failed to open master session for channel '<channel>'
Failed to open session for slave 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
Error parsing import file record number <record>, field <field>
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>)

## Address Validation

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>'

## 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 Status Messages

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

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

## 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 outside 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.

## Driver Error Messages

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

**Driver Error Messages**
**Could not allocate memory for slave device '<device name>'**
**Failed to create master SIXNET interface for channel '<channel>'**
**Failed to create SIXNET interface for slave device '<station number>' on channel '<channel>'**
**Failed to open master session for channel '<channel>'**
**Failed to open session for slave 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>'**

## Could not allocate memory for slave 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 master SIXNET interface for 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 slave 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 master session for 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 slave 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 master 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 master 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.

## Automatic Tag Database Generation Messages

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

**Automatic Tag Database Generation Messages**
**Tag import failed due to low memory resources**
**File exception encountered during tag import**
**Error parsing import file record number <record>, field <field>**
**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>)**

## 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.

## Error parsing import file record number <record>, field <field>

**Error Type:**
Warning

**Possible Cause:**
The specified field in the tag import file could not be parsed because it is longer than expected or invalid.

**Solution:**
Edit the tag import file to change the offending field if possible.

## 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

## D



## E



## F

## G



## I



## M



## N



## O

## P



## R



## S

## T



## U



## W



## X