AutomationDirect EBC Driver

Table of Contents

AutomationDirect EBC Driver

Help version 1.022

CONTENTS

Overview

The AutomationDirect EBC Driver provides a reliable way to connect AutomationDirect EBC controllers to OPC Client applications, including HMI, SCADA, Historian, MES, ERP and countless custom applications. It is intended for use with Koyo Direct/PLC Direct/AutomationDirect Logic Programmable Logic Controllers that may be accessed via an EBC or GS-EDRV Ethernet module.

Setup

Supported Devices

Terminator I/O
H2-EBC, H2-EBC-F, H4-EBC, H4-EBC-F
GS1, GS2 and GS3 Drives via GS-EDRV Ethernet module

Communication Protocol

Ethernet using Winsock V1.1 or higher.

Connection Timeout

Specify the time that the driver will wait for a connection to be made with a device. Depending on network load, the connect time may vary with each connection attempt. The default setting is 3 seconds. The valid range is 1 to 60 seconds.

Request Timeout

Specify the time that the driver will wait on a response from the device before giving up and going on to the next request. Longer timeouts only affect performance if a device is not responding. The default setting is 250 milliseconds. The valid range is 50 to 9999 milliseconds.

Retry Attempts

Specify the number of times the driver will retry a message before giving up and going on to the next message. The default setting is 3 retries. The valid range is 1 to 10.

Channel and Device Limits

The maximum number of channels supported by this driver is 100. The maximum number of devices supported by this driver is 1024 per channel.
Each device on the channel must be uniquely identified by its own IP address. The IP address is different from the Unit ID that is configurable on the I/O base unit. In general, the Device ID has the format YYY.YYY.YYY.YYY where YYY designates the device IP address. Each YYY byte should be in the range of 0 to 255.

The IP address for an EBC module can be determined using NetEdit, which is an AutomationDirect device configuration utility. NetEdit can be launched by selecting the Device ID Wizard on the General property group in Device Properties.

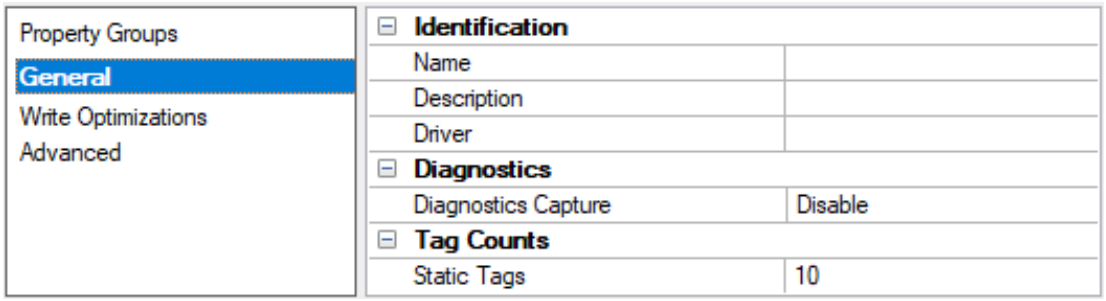
NetEdit provides the ability to query the network, configure network properties (IP Address) and update firmware for EBC devices.

Configuration

The H2-EBC(-F) and Terminator I/O EBC's automatically detect I/O modules in the base on power up. The H4-EBC(-F), however, only detects discrete modules. When using analog modules, NetEdit should be used to manually configure the H4-EBC(-F) base.

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. Specify 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. 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. 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.

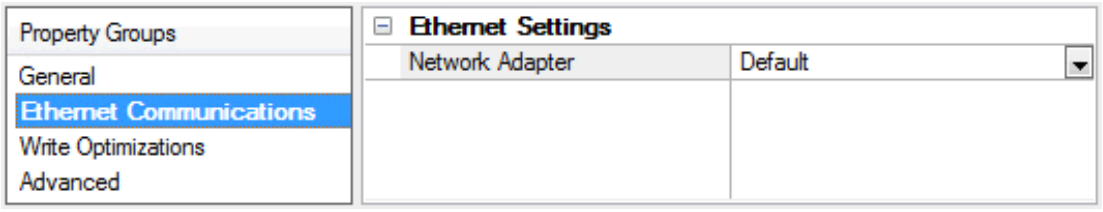
For more information, refer to "Communication Diagnostics" and "Statistics Tags" in the server help.

Tag Counts

Static Tags: Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

Channel Properties — Ethernet Communications

Ethernet Communication can be used to communicate with devices.

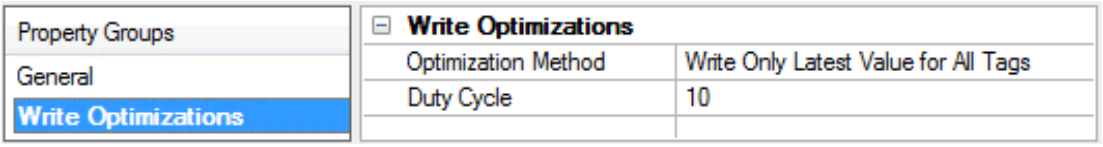


Ethernet Settings

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

Channel Properties — 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

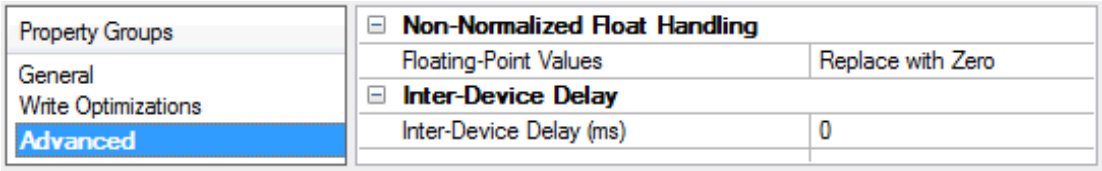
value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

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 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.

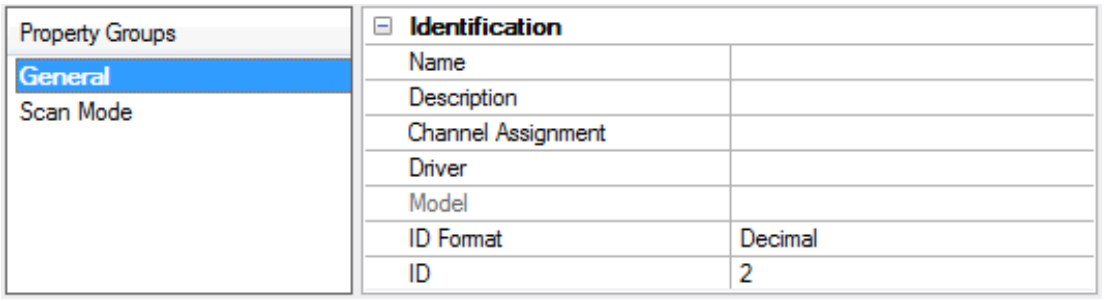
For more information on the floating-point values, refer to "How To ... Work with Non-Normalized Floating-Point Values" in the server help.

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: 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.

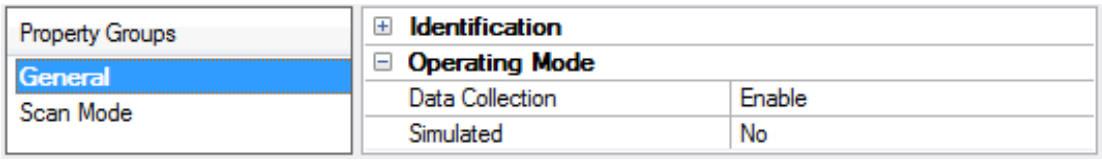
Channel Assignment: Specify the user-defined name of the channel to which this device currently belongs.

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 depend 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.

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.

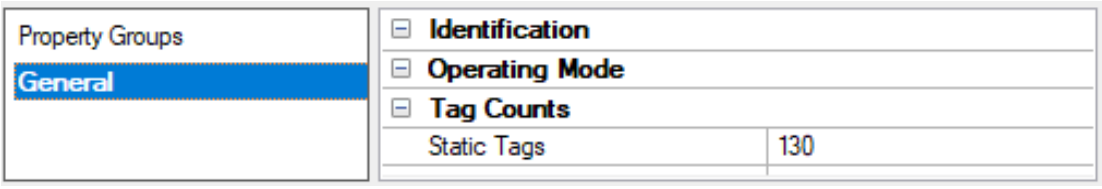
Simulated: Place the device into or out of Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client 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 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. When a device is simulated, updates may not appear faster than one (1) second client.

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

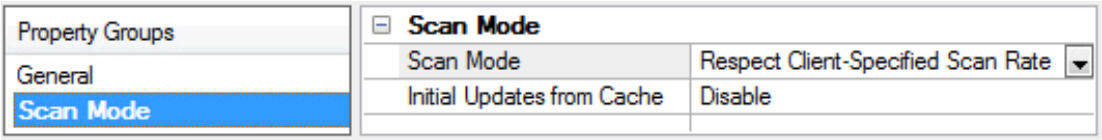
Tag Counts



Static Tags: Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

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.



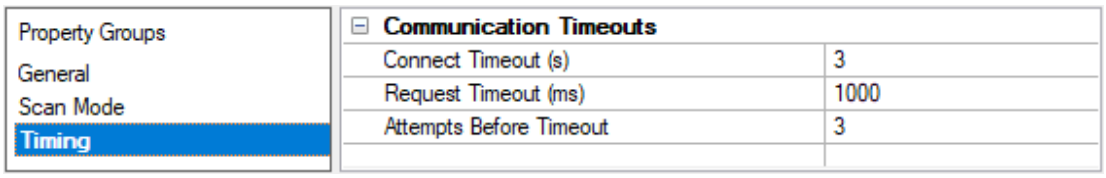
Scan Mode: Specify how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

- **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 OPC 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 9999 milliseconds (167 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most

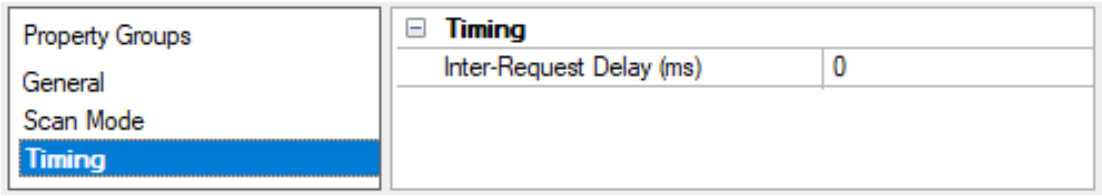
serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

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

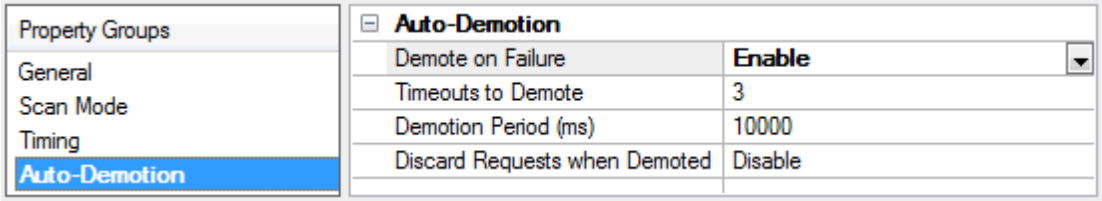
Inter-Request Delay: Specify how long the driver waits before sending the next request to the target device. It overrides the normal polling frequency of tags associated with the device, as well as one-time reads and writes. This delay can be useful when dealing with devices with slow turnaround times 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 — 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 — Tag Generation

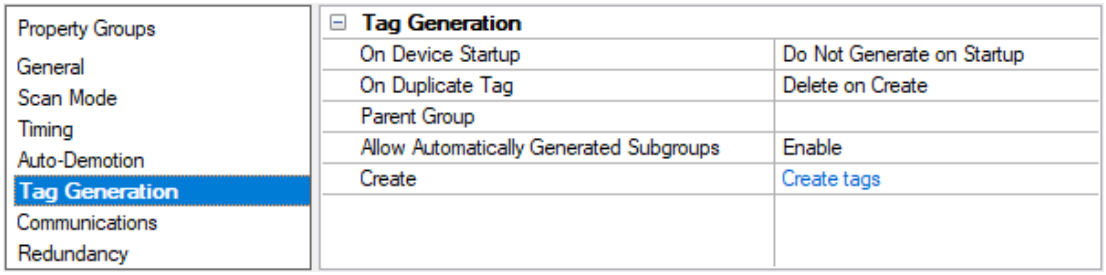
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the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

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

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

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

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



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

On Device Startup: 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.

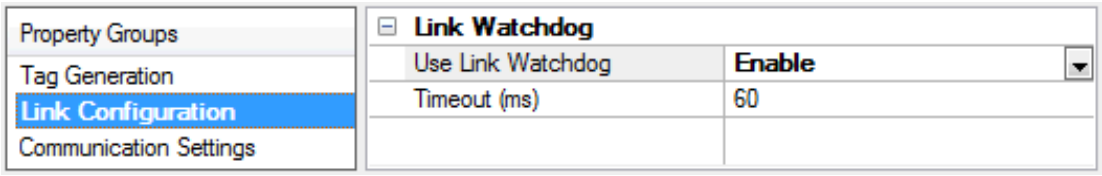
Parent Group: This property keeps automatically generated tags from mixing with tags that have been entered manually by specifying a group to be used for automatically generated tags. The name of the group can be up to 256 characters. This parent group provides a root branch to which all automatically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



Use Link Watchdog - EBC Only

The communications link between the PC and the EBC device is vital for the PC and EBC to exchange information. It is possible for this communication link to "break" during its use; meaning, there is a loss in (or lack of) communication that is either permanent or temporary. For example, a loss in communication would be a physical break in the communication network. A lack of communication would be a significant interval between PC-to-EBC transactions. This interval would have to exceed a set timeout before being considered a break in communication.

The client application can act on a loss of communication by continuously reading the _Error tag for the given EBC device. This is a _System level tag at the OPC server that is 0 when the link is good and 1 when the link is bad. Users must do more than handle operations on the client side during a broken link, however.

If the connection to the EBC is severed, it can no longer be given instructions for proper handling of I/O states (such as digital outputs). Since there are no logic controllers in EBC systems, ladder logic cannot be written to handle the situation. Fortunately, the Link Watchdog mechanism is built into the EBC to safely handle link failure. When a specified amount of time elapses with a broken link, the watchdog wakes up and turns all device I/O outputs off.

The link watchdog (which is also referred to as the link monitor) is configurable through the OPC server. It has two options: Disable and Enable. Descriptions are as follows:

- **Disable:** When the link watchdog is disabled, breaks in the communications link do not affect the device's I/O outputs. The outputs will maintain the state they were assigned prior to the break until communications are restored and outputs are altered via the client application.
- **Enable:** When enabled, the link watchdog continuously monitors the link. If a break in the link occurs, a timer is initiated. When the timer reaches the **Watchdog Timeout** value, all device I/O outputs will be turned off.

Timeout

Values for the watchdog timeout range from 60ms to 32767ms.

Link Maintenance and Device Pings

The watchdog timer is reset every time the device receives a request. To prevent a watchdog timeout, at least one request must be made to the device before the watchdog timer expires. Normally the fastest tag scan rate would determine how often the device is communicated with. Such a dependency on tag scan rate would prove inflexible. To remedy this, each EBC device with the Link Monitor enabled will be pinged intermittently to ensure there are no unnecessary watchdog timeouts. Each ping request is sent independent of tag Read/Write requests. This removes the dependency from the tag scan rate for maintaining the link.

Link Monitor Ping = Low overhead request to the device with no response.
Link Monitor Ping Interval = Watchdog Timeout value / 3

Local PC Usage and Network Activity

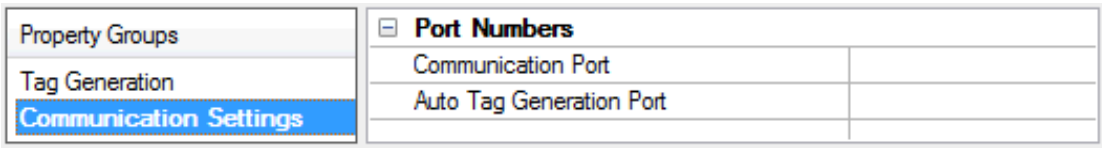
There are two factors that may impact how accurately the Link Monitor Ping request is sent at its scheduled interval: Local PC Usage and Network Activity.

Local PC Usage
In this help file, the local PC is assumed to be the same PC running the OPC server in question. Since the OPC Server shares resources with other applications running on the local PC, there is no guarantee that each Link Monitor Ping request will be sent at its set interval. It is possible that these applications may preempt the OPC server long enough for a Link Monitor Ping request to be made after the watchdog timer expires.

Network Activity
The amount of activity on the network(s) providing access to the given EBC device can greatly affect the delivery of the Link Monitor Ping request. Collisions and load can delay a request long enough for a Link Monitor Ping request to reach the device after the watchdog timer expires.

Note: The Watchdog Timeout may have to be tuned accordingly to ensure spurious watchdog timeouts do not occur.

Device Properties — Communication Settings



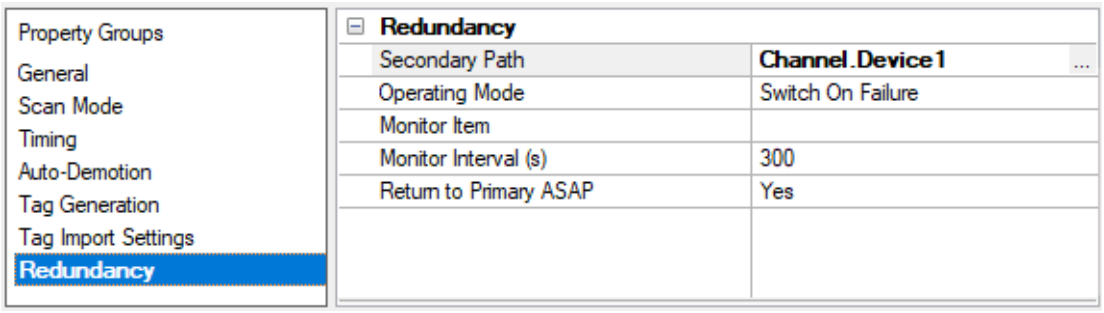
Communication Port: Select the port number to be used by the driver for communicating with the remote device. The default value is 28784 (0x7070) for non-GS device models (H2-EBC, H2-EBC-F, H4-EBC, H4-EBC-F, and Terminator I/O). For GS device models (GS1, GS2 and GS3) the default value is 502. Legal port number values are between 0-65535.

Auto Tag Generation Port: Select the port number to be used by the driver for performing automatic tag database generation from the remote device. This option is available only for GS device models (GS1, GS2 and GS3). The default value is 28784 (0x7070). Legal port number values are between 0-65535.

Notes:

1. For non-GS device models, the Communications Port value is also used for Tag Database Generation. The Tag Database Generation Port option is not visible when a non-GS device model is selected.
2. The Communication Settings can be reached when defining a device in the Device Wizard and afterward by clicking **Device Properties** | **Communication Settings**.

Device Properties — Redundancy



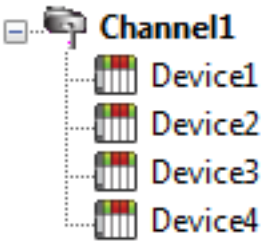
Redundancy is available with the Media-Level Redundancy Plug-In.

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

Optimizing Communications

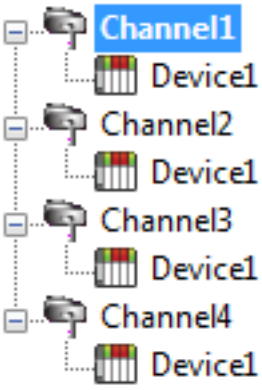
The AutomationDirect EBC Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the AutomationDirect EBC Driver is fast, there are a couple of guidelines that can be used to control and optimize the application and gain maximum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

This server refers to communications protocols like AutomationDirect EBC 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 AutomationDirect 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 AutomationDirect EBC 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 AutomationDirect EBC 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 AutomationDirect EBC Driver could only define one single channel, then the example shown above would be the only option available; however, the AutomationDirect EBC 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 than 100 devices. The application benefits from spreading the device load across channels; while it causes the server to move from device to device, it does so with far fewer devices to process on a each channel.

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
Long	Signed 32-bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
Float	32-bit floating point value* 16-bit floating point value* *

* EBC Only.

* * GS-EDRV Only.

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.

H2-EBC-(-F)

I/O Addressing
Serial Port Addressing

H4-EBC-(-F)

I/O Addressing
Serial Port Addressing

Terminator I/O

I/O Addressing
Serial Port Addressing

GS-EDRV

GS Drive Property Addressing
GS Drive Status Addressing

H2-EBC(-F) Addressing

There are two types of addressing for H2-EBC(-F) models, I/O and serial port. For more information on a specific address type, select a link from the list below.

I/O Addressing
Serial Port Addressing

H4-EBC(-F) Addressing

There are two types of addressing for H4-EBC(-F) models, I/O and serial port. For more information on a specific address type, select a link from the list below.

I/O Addressing
Serial Port Addressing

Terminator I/O Addressing

There are two types of addressing for Terminator I/O models, I/O and serial port. For more information on a specific address type, select a link from the list below.

I/O Addressing
Serial Port Addressing

Status Addressing

Extended Error information is available for referencing in certain modules. The extended error contains module specific error information that a single error code could not define on its own. This data is available regardless of whether or not the module is in an error state.

Only the T1F-14THM module currently supports addressing of Extended Error data. Each Byte represents a Thermocouple Channel. The general form of this Status information is as follows:

S<ss>:STS.EXT<nn>

where ss is the slot number (1 to 93), and nn is the 0-based channel number (0 - 63).

Example

S1:STS.EXT0 = Extended Error for Thermocouple Channel 1.

See Also: Application Notes: T1F-14THM

GS Drive Addressing

There are two types of addressing for GS-EDRV models, property and status variables. For more information on a specific address type, select a link from the list below.

GS Drive Property Addressing
GS Drive Status Addressing

H2-EBC(-F) I/O Addressing

Data Types

The EBC is designed to replace the PLC CPU. As a result, I/O slots must be individually addressed as there is no memory addressing. The general form for H2 addresses is as follows:

S<ss>:<t><nn>

where ss is the slot number (0 to 14), t is the address type (DI, DO, WI, WO) and nn is the address. The address ranges from 0 to an upper limit determined by the module occupying the slot.

Note: The default data types are shown in **bold**.

Address Type	Range	Data Type
Discrete Inputs (X: Point)	X<nn>, DI<nn> nn = Bit Number (decimal)	**Boolean**, Byte, Char, Word, Short, DWord, Long
Discrete Outputs (Y: Point)	Y<nn>, DO<nn> nn = Bit Number (decimal)	**Boolean**, Byte, Char, Word, Short, DWord, Long
Byte Inputs	BI<nn> nn = Byte Number (decimal)	**Byte**, Char
Byte Outputs	BO<nn> nn = Byte Number (decimal)	**Byte**, Char
Word Inputs (K: Analog Channel)	K<nn>, WI<nn> nn = Word Number (decimal)	**Word**, Short
Word Outputs (V: Analog Channel)	V<nn>, WO<nn> nn = Word Number (decimal)	**Word**, Short

Address Type	Range	Data Type
DWord Inputs	DWI<nn> nn = DWord Number (decimal)	**DWord**, Long
DWord Outputs	DWO<nn> nn = DWord Number (decimal)	**DWord**, Long
Float Inputs	FI<nn> nn = Float Number (decimal)	**Float**
Float Outputs	FO<nn> nn = Float Number (decimal)	**Float**
Double Inputs	DBI<nn> nn = Double Number (decimal)	**Float**
Double Outputs	DBO<nn> nn = Double Number (decimal)	**Float**

Example

The following example illustrates a sample arrangement and its corresponding addresses.

	Slot 0 8 Inputs	Slot 1 32 Inputs	Slot 2 4 Analog In	Slot 3 8 Outputs	Slot 4 16 Outputs	Slot 5 8 Analog Out
H2-EBC Module	Addresses S0:X0 | | \/ S0:X7	Addresses S1:X0 | | \/ S1:X31	Addresses S2:K0 | | \/ S2:K3	Addresses S3:Y0 | | \/ S3:Y7	Addresses S4:Y0 | | \/ S4:Y15	Addresses S5:V0 | | \/ S5:V7

Case: Various valid and invalid item entries and their meaning.

S0:X0	Bit 0 of slot 0 input
S3:K4	Word 4 of slot 3 input

Case: 32-point output module in slot 1.

S1:Y0@Byte	Byte 0 of slot 1 output
S1:Y3@Byte	Invalid address must be on even byte boundary
S1:Y8@Byte	Byte 1 of slot 1 output
S1:Y16@Word	Word 1 of slot 1 output

Case: 4-channel in/ 2-channel out module in slot 2.

S2:K3	Word 3 of slot 2 input
S2:V1	Word 1 of slot 2 output

H4-EBC(-F) I/O Addressing

Data Types

The EBC is designed to replace the PLC CPU. As a result, I/O bases and slots must be individually addressed as there is no memory addressing. The general form for H4 addresses is as follows:

B<bb>:S<ss>:<t><nn>

where bb is the base number (0 to 3), ss is the slot number (0 to 14), t is the address type (DI, DO, WI, WO) and nn is the address. The address ranges from 0 to an upper limit determined by the module occupying the slot.

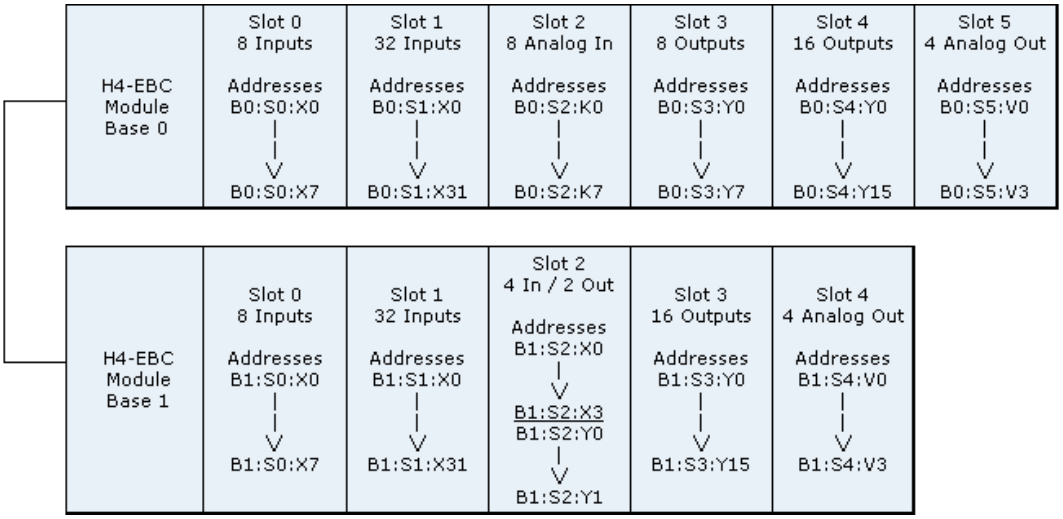
Note: The default data types are shown in **bold**.

Address Type	Range	Data Type
Discrete Inputs (X: Point)	X<nn>, DI<nn> nn = Bit Number (decimal)	**Boolean**, Byte, Char, Word, Short, DWord, Long
Discrete Outputs (Y: Point)	Y<nn>, DO<nn> nn = Bit Number (decimal)	**Boolean**, Byte, Char, Word, Short, DWord, Long
Byte Inputs	BI<nn> nn = Byte Number (decimal)	**Byte**, Char
Byte Outputs	BO<nn> nn = Byte Number (decimal)	**Byte**, Char
Word Inputs (K: Analog Channel)	K<nn>, WI<nn> nn = Word Number (decimal)	**Word**, Short
Word Outputs (V: Analog Channel)	V<nn>, WO<nn> nn = Word Number (decimal)	**Word**, Short
DWord Inputs	DWI<nn> nn = DWord Number (decimal)	**DWord**, Long
DWord Outputs	DWO<nn> nn = DWord Number (decimal)	**DWord**, Long
Float Inputs	FI<nn> nn = Float Number (decimal)	**Float**
Float Outputs	FO<nn> nn = Float Number (decimal)	**Float**
Double Inputs	DBI<nn>	**Float**

Address Type	Range	Data Type
	nn = Double Number (decimal)	
Double Outputs	DBO<nn> nn = Double Number (decimal)	**Float**

Example

The following example illustrates a sample arrangement and its corresponding addresses.



Case: Various valid and invalid item entries and their meaning.

B0:S0:X0	Bit 0 of slot 0 input, base 0
S3:K4	Error, need to specify base

Case: 32-point output module in slot 1, base 3.

B3:S1:Y0@Byte	Byte 0 of slot 1 output
B3:S1:Y3@Byte	Invalid address must be on even byte boundary
B3:S1:Y8@Byte	Byte 1 of slot 1 output
B3:S1:Y16@Word	Word 1 of slot 1 output

Case: 4-channel in/ 2-channel out module in slot 2, base 0.

B0:S2:K3	Word 3 of slot 2 input
B0:S2:V1	Word 1 of slot 2 output

Terminator I/O I/O Addressing

Data Types

The EBC is designed to replace the PLC CPU. As a result, I/O slots must be individually addressed as there is no memory addressing. The general form for Terminator I/O addresses is as follows:

S<ss>:<t><nn>

where ss is the slot number (1 to 93), t is the address type (DI, DO, WI, WO, etc.), and nn is the address. The address ranges from 0 to an upper limit determined by the module occupying the slot.

Note: The default data types are shown in **bold**.

Address Type	Range	Data Type
Discrete Inputs	DI<nn> nn = Bit Number (decimal)	**Boolean**, Byte, Char, Word, Short, DWord, Long
Discrete Outputs	DO<nn> nn = Bit Number (decimal)	**Boolean**, Byte, Char, Word, Short, DWord, Long
Byte Inputs	BI<nn> nn = Byte Number (decimal)	**Byte**, Char
Byte Outputs	BO<nn> nn = Byte Number (decimal)	**Byte**, Char
Word Inputs	WI<nn> nn = Word Number (decimal)	**Word**, Short
Word Outputs	WO<nn> nn = Word Number (decimal)	**Word**, Short
DWord Inputs	DWI<nn> nn = DWord Number (decimal)	**DWord**, Long
DWord Outputs	DWO<nn> nn = DWord Number (decimal)	**DWord**, Long
Float Inputs	FI<nn> nn = Float Number (decimal)	**Float**
Float Outputs	FO<nn> nn = Float Number (decimal)	**Float**
Double Inputs	DBI<nn> nn = Double Number (decimal)	**Float**
Double Outputs	DBO<nn> nn = Double Number (decimal)	**Float**

Example

The following example illustrates a sample arrangement and its corresponding addresses.

Terminator IO EBC Module Slot 0 Serial IO Ports EBC:SP0.Item	Slot 1 8 Digital In Addresses S1:DI0 | | \/ S1:DI7	Slot 2 16 Digital In Addresses S2:DI0 | | \/ S2:DI15	Slot 3 8 Digital Out Addresses S3:DO0 | | \/ S3:DO7	Slot 4 16 Digital Out Addresses S4:DO0 | | \/ S4:DO15	Slot 5 8 Analog In Addresses S5:DWI0 | | \/ S5:DWI7	Slot 6 16 Analog Out Addresses S6:DWO0 | | \/ S6:DWO15

Case: Various valid and invalid item entries and their meaning.

S0:DI0	Discrete input 0 of slot 0 input
S3:WI4	Word 4 of slot 3 input

Case: 32-point output module in slot 1.

S1:DO0@Byte	Byte 0 of slot 1 output
S1:DO3@Byte	Invalid address must be on even byte boundary
S1:DO8@Byte	Byte 1 of slot 1 output
S1:DO16@Word	Word 1 of slot 1 output

Case: 4-channel in/ 2-channel out module in slot 2.

S2:WI3	Word 3 of slot 2 input
S2:WO1	Word 1 of slot 2 output

H2, H4, Terminator I/O Serial Port Addressing

The EBC contains an RS232 serial port. Both the transmit buffer and receive buffer of the driver are 127 bytes in size. Likewise, the corresponding tags can be a maximum of 127 bytes. Incoming bytes are appended to the receive buffer. If the receive buffer is full or additional bytes will overflow the buffer, the buffer will reset with these additional bytes. The H2-SERIO module contains three serial ports each identical to the EBC serial port. The EBC serial port and the H2-SERIO serial ports differ only in the way the ports are addressed.

The following is a list of possible configurations.

1. Baud rates: 9600
2. Data bits: 7 or 8
3. Parity: None, Odd, or Even
4. Stop bits: 1 or 2

Notes:

1. The device will not maintain configuration values while its power is off. Upon power-up, the values will need to be re-entered.
2. An RJ45 connector is required.

Specifying A Port

Port specifiers precede the serial port address and define the port to which the serial port address corresponds. No base or slot information is needed to address the EBC serial port. To define an EBC address, the mnemonic "EBC" is used. The mnemonic SP0 means serial port 0. The EBC serial port is always serial port 0.

For addressing serial ports in modules, a base (if applicable) and slot must be specified. This is similar to I/O addressing. The syntax is described below.

- <bb> is the base number in which the module is located.
- <ss> is the slot number in which the module is located.
- <pp> specifes the port number of interest.

For example, when starting in slot 0 (or 1 for Terminator I/O), the first serial port module encountered is assigned ports 1 - (1 + # module 1 serial ports). Let x = (1 + # module 1 serial ports). The second serial port module encountered is assigned ports (x - (x + # module 2 serial ports). The table below summarizes port specification for both EBC and serial port module addressing.

Location	H2/Terminator I/O Syntax	H4 Syntax
EBC (Base)	EBC:SP0	EBC:SP0
Module	S<ss>:SP<pp>	B<bb>:S<ss>:SP<pp>

Example

EBC = H2
Slot 1 = 3 port serial port module
Slot 5 = 3 port serial port module

To specify the third port in Slot 1, enter:
S1:SP3

To specify the third port in Slot 5, enter:
S5:SP6

Specifying an Address

For the following topics, <port spec> will be a placeholder for the port specifier previously discussed. Examples are given to illustrate the combination of port specifier + serial port address and example values they may undertake.

Serial Data Received
Number of RX Queue Bytes
Flush Data Received
Flush RX Queue
Serial Data To Transmit
Number of TX Queue Bytes
Flush TX Queue
Serial Port Configuration: Baud Rate
Serial Port Configuration: #Data Bits
Serial Port Configuration: Parity
Serial Port Configuration: #Stop Bits
Serial Port Configuration: Mode
Serial Port Configuration: Use RTS Line
Serial Port Configuration: Pre-Transmit Delay
Serial Port Configuration: Post-Transmit Delay

Serial Data Received

To read serial data received, reference address type *<port spec>.DATAIN*. All bytes read will be removed from the serial port's RX queue upon reading *DATAIN*.

Requirements

Serial port configuration addresses must have appropriate values for proper operation.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Number of Bytes Read	Access
<port spec>.DATAIN [r][c]*	**Byte Array**, Char Array	r times c	Read Only
<port spec>.DATAIN	String	127	Read Only

* To access as an array, *[row][column]* form is required. For example, DATAIN [1][24] would display 24 ASCII bytes in array notation: [x1, x2, x3..x24])

Examples

Address	Value	Description
EBC.SP0.DATAIN	"hello"	Port 0 input data viewed as a string.
S3.SP2.DATAIN [2][2]	[105, 105][105, 105]	Port 2 input data in array form. In string form, this would equate to "iiii".

Number of RX Queue Bytes

The number of bytes in the serial port's RX queue can be accessed by referencing address type *RXAVAIL*.

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.RXAVAIL	**Word**, Short	Read Only

Examples

Address	Value	Description
EBC.SP0.RXAVAIL	0	Port 0, no bytes in RX queue.
B0.S1:SP1.RXAVAIL	5	Port 1, 5 bytes in RX queue.

Flush Data Received

To flush the local *DATAIN* buffer, reference *DIFLUSH*.

Values

true = flush DATAIN
false = do nothing

Requirements

Serial port configuration addresses must have appropriate values for proper operation.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.DIFLUSH	**Boolean**	Write Only

Examples

Address	Value	Description
EBC:SP0.DIFLUSH	true	Flush port 0's DATAIN buffer.

Flush RX Queue

To flush the serial port's RX queue, reference *RXFLUSH*. All bytes in the serial port's RX queue will be lost and therefore not accessible on subsequential reads of *DATAIN*.

Values

true = flush queue
false = do nothing

Requirements

Serial port configuration addresses must have appropriate values for proper operation.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.RXFLUSH	**Boolean**	Write Only

Examples

Address	Value	Description
EBC:SP0.RXFLUSH	true	Flush port 0's RX queue.

Serial Data To Transmit

To transmit serial data, reference address type *DATAOUT*. All bytes written will be removed from the serial port's TX queue upon completion of transmission.

Requirements

Serial port configuration addresses must have appropriate values for proper operation.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Number of Bytes Read	Access
<port spec>.DATAOUT [r][c]*	**Byte Array**, Char Array	r times c	Write Only
<port spec>.DATAOUT	String	length of null terminated string	Write Only

* To access as an array, *[row][column]* form is required. For example, DATAOUT [1][24] would send 24 ASCII bytes in array notation: [x1, x2, x3..x24])

Examples

Address	Value	Description
EBC.SP0.DATAOUT	"hello"	Port 0, transmit "hello".
S3.SP2.DATAOUT [2][2]	[105, 105][105, 105]	Port 2 data to transmit in array form. In string form this would equate to "iiii".

Number of TX Queue Bytes

Reference *TXLEFT* to determine how many bytes are left in the TX queue. This value is available after transmission of serial data.

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.TXLEFT	**Word**, Short	Read Only

Examples

Address	Value	Description
EBC.SP0.TXLEFT	0	Port 0, no bytes left in queue, all bytes on last transmission have been sent.
B0.S1:SP1.TXLEFT	10	Port 1, 10 bytes have yet to be sent and reside in the TX queue.

Flush TX Queue

To flush the serial port's TX queue, reference *TXFLUSH*. All bytes in the serial port's TX queue will be lost and therefore not sent to the target serial device. *TXLEFT* will reset upon flushing of the TX queue.

Values

true = flush queue
false = do nothing

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.TXFLUSH	**Boolean**	Write Only

Examples

Address	Value	Description
EBC.SP0:TXFLUSH	true	Flush port 0's TX queue.

Serial Port Configuration: Baud Rate

To configure the baud rate for a serial port, reference *BAUD*.

Values

9600

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.BAUD	**DWord**, Long	Read/Write

Examples

Address	Value	Description
EBC.SP0.BAUD	9600	Port 0, baud = 9600.

Serial Port Configuration: #Data Bits

To configure the number of data bits for a serial port, reference *DATABITS*.

Values

7 or 8

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.DATABITS	**Byte**, Char	Read/Write

Examples

Address	Value	Description
EBC.SP0.DATABITS	7	Port 0 configured for 7 databits.

Serial Port Configuration: Parity

To configure the parity for a serial port, reference *PARITY*.

Values

0, 1 = none
2 = odd
3 = even

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.PARITY	**Byte**, Char	Read/Write

Examples

Address	Value	Description
EBC.SP0.PARITY	0	Port 0 configured for no parity.

Serial Port Configuration: #Stop Bits

To configure the number of stop bits for a serial port, reference *STOPBITS*.

Values

1 or 2

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.STOPBITS	**Byte**, Char	Read/Write

Examples

Address	Value	Description
EBC.SP0.STOPBITS	1	Port 0 configured for 1 databits.

Serial Port Configuration: Mode

To configure the mode of operation for the serial port, reference *MODE*.

Values

0 = KSequence Server (not supported)
1= Proxy (ASCII communications)

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.MODE	**Byte**, Char	Read/Write

Examples

Address	Value	Description
EBC.SP0.MODE	1	Port 0 configured for generic ASCII communications

Serial Port Configuration: Use RTS Line

Flow control may be required to communicate with certain serial devices. A flow control option available is *USERTS*. This specifies whether the RTS line is to be used or not. The RTS line will be high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line will be low. This is normally used with RS232/RS485 converter hardware.

Values

true = Use the RTS line
false = Don't use the RTS line

Requirements

None.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.USERTS	**Boolean**	Read/Write

Examples

Address	Value	Description
EBC.SP0.USERTS	true	Port 0 configured to use the RTS line.

Serial Port Configuration: Pre-Transmit Delay

Reference *PRETXDLY* to configure how long the RTS line is to be high before beginning transmission.

Values

Delay = PRETXDLY value milliseconds times 2.

Requirements

USERTS = true.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.PRETXDLY	**Byte**, Char	Read/Write

Examples

Address	Value	Description
EBC.SP0.PRETXDLY	2	Port 0 configured to delay 4 ms.

Serial Port Configuration: Post-Transmit Delay

Reference *POSTTXDLY* to configure how long the RTS line is to be held high after transmission.

Values
Delay = POSTTXDLY value milliseconds times 2.

Requirements
USERTS = true.

Note: The default data types are shown in **bold**.

Syntax	Data Type	Access
<port spec>.POSTTXDLY	**Byte**, Char	Read/Write

Examples

Address	Value	Description
EBC.SP0.POSTTXDLY	1	Port 0 configured to delay 2 ms.

GS Drive Parameter Addressing
GS Drives are addressed by means of parameters and parameter groups. Both are specific to the drive in question and can be found by consulting the drive's user manual.

Addressing Syntax
Both group and property are 0-based. All properties are Read/Write.

P<group>.<parameter>

Parameter Definitions
Definitions for both parameter groups and properties can be found in the particular drive's user manual.

Drive	Manual
GS1	GS1-M
GS2	GS2-M
GS3	GS3-M

Float Data
Properties with default data type Float also support Word and Short access. Float data is returned from the drives in raw form (no decimal point). Referencing a Float parameter as Word and Short will return this raw value. Referencing it as a Float will return the scaled value in floating point notation.

Examples
1. Drive: GS1

- Parameter Group: Ramps
- Parameter: Acceleration Time 1
- Address: **P1.1**
- Data Type: Float, Value = 10.0
- Data Type: Word, Value = 100

2. Drive: GS2

- Parameter Group: Analog
- Parameter: Analog Input Reverse Motion Enable
- Address: **P4.4**
- Data Type: Word

GS Drive Status Addressing

GS Drives also contain status information much the same way properties are addressed. Available status variables are specific to the drive in question and can be found by consulting the drive's user manual.

Addressing Syntax

Both group and status variable are 0-based. Some status variables are Read Only.

ST<group>.<status variable>

For now, only group 0 is available.

Status Definitions

Group	Status Variable	Model	Label	Data Type
0	0	GS1/GS2/GS3	Status_Monitor_1	Word
0	1	GS1/GS2/GS3	Status_Monitor_2	Word
0	2	GS1/GS2/GS3	Frequency_Command_F	Float
0	3	GS1/GS2/GS3	Output_Frequency_H	Float
0	4	GS1/GS2/GS3	Output_Current_A	Float
0	5	GS1/GS2/GS3	DC_BUS Voltage_U	Float
0	6	GS1/GS2/GS3	Output_Voltage_E	Float
0	7	GS1/GS3	Motor_RPM	Word
0	8	GS1/GS3	Scale_Frequency_Low	Word
0	9	GS1/GS3	Scale_Frequency_High	Word
0	10	GS2/GS3	Power_Factor_Angle	Word
0	11	GS1/GS3	Percent_Load	Word
0	12	GS3	PID_Setpoint	Word
0	13	GS3	PID_Feedback_Signal	Word
0	14	N/A	Reserved	N/A
0	15	N/A	Reserved	N/A
0	16	GS1/GS2/GS3	Software_Version	Word
0	17	N/A	Reserved	N/A
0	18	GS1/GS2/GS3	Serial_Status	Word

Note: Definitions for status variables can be found in the specific driver's user manual.

Drive	Manual

GS1	GS1-M
GS2	GS2-M
GS3	GS3-M

Float Data

Status variables with default data type Float also support Word and Short access. Float data is returned from the drives in raw form (no decimal point). Referencing a Float status variable as Word and Short will return this raw value. Referencing it as a Float will return the scaled value in floating point notation.

Examples

1. Drive: GS1

- Status: Status Monitor 1
- Address: **ST0.0**
- Data Type: Word

2. Drive: GS3

- Status Variable: DC BUS Voltage U
- Address: **ST0.5**
- Data Type: Float, Value = 24.0
- Data Type: Word, Value = 240

Module Hot Swapping

Terminator I/O allows for the Hot Swapping of I/O modules (blue component) while the power is on. I/O bases (black component) are not hot swappable. AutomationDirect.com states the I/O module replacement procedure as follows:

1. Remove the I/O module from base.
2. Install the new I/O module of the same part number.
3. Verify that the Base Controller LEDs have returned to normal.

Note: It is strongly recommended that modules are hot swapped one at a time.

What If I Don't Follow the Preferred Replacement Procedure?

A replacement procedure differing from the above may produce erroneous results from the server. It is very important that the new module is the same size as the previous module. Size refers to the number of Discretes/Bytes/Words/DWords/Floats and etc. If the previous module had 8 DWords and 8 Discretes, the new module must have 8 DWords and 8 Discretes. The safest measure to take is to replace a module with a module of the same part number.

If the sizes do not match, communication errors will occur and OPC data will go Bad. To remedy this problem, cycle the device's power when safe and appropriate. It is possible to remain in an erroneous state after cycling the device's power. This condition may occur if the replacement module has different address types then the original module. In this case, client tags for this module are referencing invalid addresses. The server project and possibly the client project will need editing to correct this problem. Again, this problem will not occur if the replacement module and original module have the same part number.

Error Descriptions

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

Driver Error Messages

Winsock initialization failed (OS Error = n)
Winsock V1.1 or higher must be installed to use the AutomationDirect EBC device driver
Memory allocation error

Driver Warning Messages

Device '<device name>' is not responding
Unable to write to '<address>' on device '<device name>'
Device address '<address>' contains a syntax error
Address '<address>' is out of range for the specified device or register
Device address '<address>' is not supported by model '<model name>'
Device address '<address>' is Read Only
Data type '<type>' is not valid for device address '<address>'
Device '<device name>' returned an error with value = '<error value>' disabling Link Watchdog. All tags will be invalidated
Device '<device name>' returned an error with value = '<error value>' enabling Link Watchdog with timeout = '<timeout>'. All tags will be invalidated

Read Errors

Address '<address>' is out of range for device '<device name>'. Tag Deactivated
Base referenced in address '<address>' on device '<device name>' does not exist. Tag Deactivated
Module referenced in address '<address>' on device '<device name>' does not exist. Tag Deactivated
Invalid address type for address '<address>' on device '<device name>'. Tag Deactivated
Device '<device name>' returned an error code with value = '<value>' reading address '<address>'
Device '<device name>' : Base '<base>' : Slot '<slot>' returned an error with value = '<error value>'
Device '<device name>' : Base '<base>' : Slot '<slot>' returned a warning with value = '<warning value>'
Device '<device name>' : Base '<base>' : Slot '<slot>' returned information with value = '<info value>'
Device '<device name>' : Base '<base>' : Slot '<slot>' returned an internal error with value = '<internal value>'
Device '<device name>' : Base '<base>' : Slot '<slot>' returned extended error: Type = '<ext err type>', Data = '<error bytes>'
Frame received from device '<device name>' contains errors
Frame received from device '<device name>' contains errors. Verify its IP address
Format version of frame received from device '<device name>' is not supported. Tag Deactivated

Model selected for device '<device name>' does not match the actual device model. Verify its IP address
Device '<device name>' block request [<start> to <end>] responded with exception <code>
Bad received length [<start> to <end>] on device '<device name>'

Write Errors

Write rejected. Address '<address>' is out of range for device '<device name>'
Write rejected. Base referenced in address '<address>' on device '<device name>' does not exist
Write rejected. Module referenced in address '<address>' on device '<device name>' does not exist
Write rejected. Invalid address type for address '<address>' on device '<device name>'
Write to address '<address>' failed. Device '<device name>' returned an undefined status code with value = '<value>'
Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned an error with value = '<error value>'
Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned a warning with value = '<warning value>'
Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned information with value = '<info value>'
Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned an internal error with value = '<internal value>'
Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned extended error: Type = '<ext err type>', Data = '<error bytes>'
Write to address '<address> ' failed. Frame received from device '<device name>' contains errors
Write to address '<address>' failed. Frame received from device '<device name>' contains errors. Verify its IP address
Write to address '<address>' failed. Format version of frame received from device '<device name>' is not supported
Write to address '<address>' failed. Model selected for device '<device name>' does not match the actual device model. Verify its IP address
Unable to write to address <address>. Device '<device name>' responded with exception <code>

See Also: EBC Error Codes

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 AutomationDirect EBC 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.

Memory allocation error

Error Type:
Fatal

Possible Cause:
Insufficient system RAM to support the number of tags the driver is being asked to scan.

Solution:
Increase the amount of system memory or reduce the number tags being scanned.

Device '<device name>' is not responding

Error Type:
Serious

Possible Cause:

1. The Ethernet connection between the device and the Host PC is broken.
2. The named device may have been assigned an incorrect IP address.
3. The requested address is not available in the device.
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 EBC device network.
2. Verify the IP address given to the named device matches that of the actual device.
3. Verify that the device supports the requested address.
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 Ethernet connection between the device and the Host PC is broken.
2. The named device may have been assigned an incorrect IP address.
3. The requested address is not available in the device.

Solution:

1. Verify the cabling between the PC and the EBC device network.
2. Verify the IP address given to the named device matches that of the actual device.
3. Verify that the device supports the requested address.

Device address '<address>' contains a syntax error

Error Type:
Warning

Possible Cause:

1. A tag address contains one or more invalid characters.
2. Bit addressing notation conflicts with the assigned data type.

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 this address type on the specified device. This range is generic and is designed to set a

hard upper limit on locations for the specified address type. Each module will impose their own upper limit which is less than or equal to the generic upper limit.

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

Device address '<address>' is not supported by model '<model name>'

Error Type:
Warning

Possible Cause:
A tag address that has been specified dynamically is not supported by the target model.

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

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.

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 <device name> returned an error with value = <error value> disabling Link Watchdog. All tags will be invalidated.

Error Type:
Warning

Possible Cause:
The EBC device generated an error during a Link Watchdog disabling procedure. Due to this failure, it is uncertain whether the Link Watchdog is truly disabled on the device-side. All device tags will be invalidated for precautionary reasons.

Solution:
Such an error state cannot be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Device <device name> returned an error with value = <error value> enabling Link Watchdog with timeout = <timeout>. All tags will be invalidated

Error Type:
Warning

Possible Cause:
The EBC device generated an error during a Link Watchdog enabling procedure. Due to this failure, it is uncertain whether the Link Watchdog is truly enabled on the device-side with the specified timeout. All tags will be invalidated for precautionary reasons.

Solution:
Such an error state cannot be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Address '<address>' is out of range for device '<device name>'. Tag Deactivated

Error Type:
Warning

Possible Cause:
A tag address references a location that is considered out of range by the target module.

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

Tag DeactivatedBase referenced in address '<address>' on device '<device name>' does not exist. Tag Deactivated

Error Type:
Warning

Possible Cause:
A tag address references an extension base that does not exist.

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

Note:
This applies to H4-EBC(-F).

Module referenced in address '<address>' on device '<device name>' does not exist. Tag Deactivated

Error Type:
Warning

Possible Cause:
A tag address references a slot that is not occupied by an I/O module.

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

Address type invalid for address '<address>' on device '<device name>'. Tag Deactivated

Error Type:
Warning

Possible Cause:
A tag references an address using an address type that is not valid for that module.

Solution:
Verify that the address type is correct; if it is not, re-enter it in the client application. Recall that 'X' (inputs) and 'Y' (outputs) are for discrete modules while 'K' (inputs) and 'V' (outputs) are for analog.

Device '<device name>' returned an error code with value = '<value>' reading address '<address>'

Error Type:
Warning

Possible Cause:
Device <device name> returned an error code during a read request to address <address>. Given the address is that of a non-blocking tag (ie. serial port tags), the error will be displayed for each non-block read and then invalidated. If the address is that of a block read (ie. I/O tags), the address will be displayed as <I/O Block>. This means that the error and invalidation applies to all tags for this device (excluding the serial port tags).

Solution:
Such an error state can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Device '<device name>' : Base '<base>' : Slot '<slot>' returned an error with value = '<error value>'

Error Type:
Warning

Possible Cause:
The module in base <base>, slot <slot>, generated an error during its operation. All tags referencing this slot will be invalidated.

Solution:
Such an error state can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Device '<device name>' : Base '<base>' : Slot '<slot>' returned a warning with value = '<warning value>'

Error Type:
Warning

Possible Cause:
The module in base <base>, slot <slot>, generated a warning during its operation.

Solution:
This warning state will be cleared if possible. Check the error/warning/info list for a detailed description of this warning value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Device '<device name>' : Base '<base>' : Slot '<slot>' returned information with value = '<info value>'

Error Type:
Information

Possible Cause:
The module in base <base>, slot <slot>, generated information during its operation.

Solution:
This state will be cleared. Check the error/warning/info list for a detailed description of this information value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Device '<device name>' : Base '<base>' : Slot '<slot>' returned an internal error with value = '<internal value>'

Error Type:
Error

Possible Cause:
The module in base <base>, slot <slot>, generated an internal error during its operation. All tags referencing this slot will be invalidated.

Solution:
Such an error state is fatal and can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this internal error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Device '<device name>' : Base '<base>' : Slot '<slot>' returned extended error: Type = '<ext err type>', Data = '<error bytes>'

Error Type:
Warning

Possible Cause:
The module in base <base>, slot <slot>, generated an extended error of type <ext err type> during its operation. Extended error information is contained within the error bytes. All tags referencing this slot will be invalidated.

<error bytes> is a generic array of bytes whose meaning depends on both the module and the error. In most cases, each byte represents an analog channel.

00 = analog channel good
01 = analog channel in error (such as a blown fuse or broken transmitter).

For example, given a thermocouple module:
"...Data = '00 00 01 00 00 00 00 00 00 00 01 00 00 00 00 00'" means Channel 3 and Channel 11 contain broken transmitters.

Solution:
Such an error state can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If it is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Frame received from device '<device name>' contains errors

Error Type:
Warning

Possible Cause:
The EBC device <device name> responded with incorrect data possibly due to transmission errors or device malfunction.

Solution:
1. Place device on less noisy network if that is the case.
2. Increase the request timeout.

Frame received from device '<device name>' contains errors. Verify its IP address

Error Type:
Warning

Possible Cause:
The EBC device <device name> responded with incorrect data for the specified model but valid for another model.

Solution:
Verify the correct device model and IP address was selected.

Format version of frame received from device '<device name>' is not supported. Tag Deactivated

Error Type:
Warning

Possible Cause:
The EBC device <device name> responded with a frame whose protocol version is not supported. Currently, version 1 and below are supported.

Solution:
The AutomationDirect EBC Driver will not be able to communicate with this device.

Model selected for device '<device name>' does not match the actual device model. Verify its IP address

Error Type:
Warning

Possible Cause:

Terminator I/O EBC's specify its model within the base definition. If a device specified in the server project is given a model that does not match the model specification in the base definition (ie. non-Terminator I/O), then this error will occur. All tags for this device will be invalidated.

Solution:
Verify the model and/or IP address selected for device <device name>.

Device '<device name>' block request [<start> to <end>] responded with exception <code>

Error Type:
Warning

Possible Cause:
Device <device name> returned an error code during a GS Drive block read request starting at address <start> and ending at address <end> Addresses are listed in Modbus address form (4xxxx). Consult the drive's user manual for the corresponding properties for the given start and end Modbus addresses. For example,

4-03 Analog Input Gain
Memory Address 0403H (41208)

In this example, the address 41208 is the Modbus address for parameter P4.3.

Solution:
Such an error state can not be cleared unless the problem is remedied. Contact Technical Support.

Bad received length [<start> to <end>] on device '<device name>'

Error Type:
Warning

Cause:
The driver attempted to read a block of memory in the GS Drive. The Drive 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 GS Drive.

Write rejected. Address '<address>' is out of range for device '<device name>'

Error Type:
Warning

Possible Cause:
A write operation was attempted on 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.

Write rejected. Base referenced in address '<address>' on device '<device name>' does not exist

Error Type:
Warning

Possible Cause:
A write operation was attempted on an extension base that does not exist.

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

Note:
This applies to H4-EBC(-F).

Write rejected. Module referenced in address '<address>' on device '<device name>' does not exist

Error Type:
Warning

Possible Cause:
A write operation was attempted on a slot that is not occupied by an I/O module.

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

Write rejected. Invalid address type for address '<address>' on device '<device name>'

Error Type:
Warning

Possible Cause:
A write tag references an address using an address type that is not valid for that module.

Solution:
Verify that the address type is correct; if it is not, re-enter it in the client application. Recall that 'X' (inputs) and 'Y' (outputs) are for discrete modules while 'K' (inputs) and 'V' (outputs) are for analog.

Write to address '<address>' failed. Device '<device name>' returned an undefined status code with value = '<value>'

Error Type:
Warning

Possible Cause:
Device <device name> returned a status code that is not an error/warning/info/internal/extended error. Write failed.

Solution:
N/A.

Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned an error with value = '<error value>'

Error Type:
Warning

Possible Cause:
The module in base <base>, slot <slot>, generated an error during a write operation to address <address>. Write failed.

Solution:
Such an error state can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned a warning with value = '<warning value>'

Error Type:
Warning

Possible Cause:
The module in base <base>, slot <slot>, generated a warning during a write operation to address <address>. Write may have succeeded.

Solution:
This warning state will be cleared if possible. Check the error/warning/info list for a detailed description of this warning value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned information with value = '<info value>'

Error Type:
Information

Possible Cause:
The module in base <base>, slot <slot>, generated information during a write operation to address <address>. Write may have succeeded.

Solution:
This state will be cleared. Check the error/warning/info list for a detailed description of this information value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned information with value = '<info value>'

Error Type:
Information

Possible Cause:
The module in base <base>, slot <slot>, generated information during a write operation to address <address>. Write may have succeeded.

Solution:
This state will be cleared. Check the error/warning/info list for a detailed description of this information value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned an internal error with value = '<internal value>'

Error Type:
Error

Possible Cause:
The module in base <base>, slot <slot>, generated an internal error during a write operation to address <address>. Write failed.

Solution:
Such an error state is fatal and can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this internal error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Write to address '<address>' failed. Device '<device name>' : Base '<base>' : Slot '<slot>' returned extended error: Type = '<ext err type>', Data = '<error bytes>'

Error Type:
Warning

Possible Cause:
The module in base <base>, slot <slot>, generated an extended error of type <ext err type> during a write operation to address <address>. Extended error information is contained within the error bytes. Write failed.

Solution:
Such an error state can not be cleared unless the problem is remedied. Check the error/warning/info list for a detailed description of this error value. If is not included in the list, contact the device manufacturer for further assistance.

See Also:
EBC Error Codes

Write to address '<address>' failed. Frame received from device '<device name>' contains errors

Error Type:
Warning

Possible Cause:
A write operation was retried the preset number of times and failed each time. This is possibly due to transmission errors or device malfunction.

Solution:

1. Place device on less noisy network if that is the case.
2. Increase the request timeout.

Write to address '<address>' failed. Frame received from device '<device name>' contains errors. Verify its IP address

Error Type:
Warning

Possible Cause:
A write operation was retried the preset number of times and failed each time.

Solution:
Verify the correct device model and IP address was selected.

Write to address '<address>' failed. Format version of frame received from device '<device name>' is not supported

Error Type:
Warning

Possible Cause:
The EBC device <device name> responded with a frame whose protocol version is not supported. Currently, version 1 and below are supported. Write failed.

Solution:
The AutomationDirect EBC Driver will not be able to communicate with this device.

Write to address '<address>' failed. Model selected for device '<device name>' does not match the actual device model. Verify its IP address

Error Type:
Warning

Possible Cause:
Terminator I/O EBC's specify its model within the base definition. If a device specified in the server project is given a model that does not match the model specification in the base definition (ie. non-Terminator I/O), then this error will occur. Write failed.

Solution:
Verify the model and/or IP address selected for device <device name>.

Unable to write to address <address>. Device '<device name>' responded with exception <code>

Error Type:
Warning

Possible Cause:
Device <device name> returned an error code during a GS Drive write request to address <address>. Addresses are listed in Modbus address form (4xxxx). Consult the drive's user manual for the corresponding properties for the given start and end Modbus addresses. For example,

4-03 Analog Input Gain
Mem Addr 0403H (41208)

The address 41208 is the Modbus address for parameter P4.3.

Solution:
Such an error state can not be cleared unless the problem is remedied. Contact Technical Support.

EBC Errors/Warnings/Information Return Values & Description

Value	Description
111	Invalid type.
112	RAM already locked.
113	Invalid request.
114	Timeout error.
115	Flash program error.
116	Invalid OS.
117	Invalid location (ie. Write attempted to an invalid analog channel).
118	Invalid slot number.
119	Invalid data.
120	Module busy.
121	Analog Input Channel failure; nn contains channel number that failed.
122	Unused analog input channels exist.
123	Invalid UDP port.
124	Shutdown OS.
125	Invalid IP address.
126	Protection error.
127	Unknown type.
128	Backplane initialization error.
129	Unknown response.
130	Unknown Read/Write format.
131	Unknown ACK.
132	Unknown NAK.
133	Range error.
134	Length warning.
135	Invalid base number.
136	Invalid module type.
137	Invalid offset.
138	Invalid boot version for OS.
139	Broken transmitter; nn contains channel number that failed.
140	Invalid address.
142	Channel fail multiple; nn contains channel BITS from module. Example: If bit 0 is set then channel 0 has failed If bit 1 and 3 are set then channels 1 and 3 have failed.
153	I/O module missing (I/O module removed in hot swap).
154	I/O Base has changed (I/O module replaced in hot swap).
155	Module in error. Possible errors:

Value	Description
	- missing 24V on discrete modules - blown fuses on discrete modules - missing 24V on analog modules - missing CJC block on the T1F-14THM
200-216	XX unused analog input channels exist where: XX = Value - 200.
> 32 (0x20) and < 64 (0x40) for 405 Family	BIT Type of Error 0 Terminal block off 1 External P/S voltage low 2 Fuse blown 3 Bus Error 4 Module init error (intelligent module) 5 Faults exist in module (this bit is set if any of the above bits are set) Example: 0x22: External P/S Voltage low
0x8000	Function not implemented.
0x8001	Version passed to function not correct for library.
0x8002	Supplied transport not supported.
0x8003	Supplied device is not valid.
0x8004	Supplied buffer is too small.
0x8005	Zero bytes were returned in the packet.
0x8006	Timeout error.
0x8007	Supplied protocol not supported.
0x8008	The device's IP address has not been set.
0x8009	No transport specified.
0x800A	IPX transport not installed.
0x800B	Error opening IPX Socket.
0x800C	No packet driver found.
0x800D	CRC did not match.
0x800E	Memory allocation error failed.
0x800F	No cache has been allocated for IPX.
0x8010	Invalid request.
0x8011	No response was available.
0x8012	Invalid format response was received.
0x8013	Given data is too large.
0x8014	Error loading procedures.
0x8015	Attempted command before successful OpenTransport.
0x8016	Data not aligned on proper boundary.

Application Notes

Applications notes exist for the following modules. Select a link from the following list to obtain specific information for the module of interest.

T1F-14THM
DF-HSC

Application Notes: T1F-14THM

Enabling/Disabling of Thermocouple Channels

The 14THM Module provides 14 channels for thermocouple input. The module also provides the capability to enable/disable thermocouple channels based on the application's needs. It is preferred that users only enable the channels that will be used. An enabled channel without a thermocouple (or short to ground) will return a broken transmitter error to the AutomationDirect EBC Device Driver.

Broken Transmitter = Enabled, open thermocouple channel.

All tags referencing channels with broken transmitters will have a bad quality. All tags referencing disabled channels will have a good quality because disabled channels are not technically in error. The driver cannot distinguish good data originating from enabled channels from meaningless data originating from disabled channels. For this reason, it is important that users only reference enabled channels from the client.

Viewing a Thermocouple Channel's Status

Users can view a thermocouple channel's status through the Event Log and Tags.

Event Log

The list of broken transmitters for a thermocouple module is given in the error **Device '<device name>' : Base '<base>' : Slot '<slot>' returned extended error: Type = '<ext err type>', Data = '<error bytes>'** where the <error bytes> lists the status of each channel in the form of an array. The meaning of each array byte is as follows:

00 = analog channel good
01 = broken transmitter

Example

"...Data = '00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00'" means Channel 3 contains a broken transmitter.

Tags

Broken transmitter information can be referenced from a tag through **Terminator I/O Status Addressing**. The syntax for referencing broken transmitters is S1:STS.EXT<nn> where nn is the 0-based thermocouple channel. The values of the tag are as follows:

0 = analog channel good
1 = broken transmitter

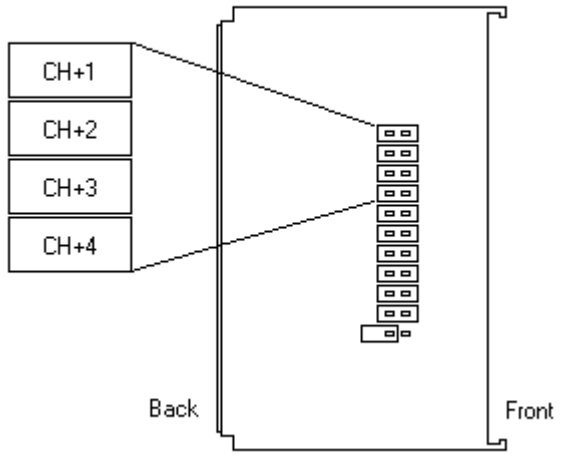
Examples

1. S1:STS.EXT0 = Extended error for thermocouple channel 1 in Slot 1.
2. S1:STS.EXT0 = 1 means a broken transmitter has been detected on Channel 1.

Note: If a channel is referenced that exceeds the number of channels for the module, STS.EXT<nn> will hold the value 0.

The diagram below displays the jumper layout for the 14THM module. The table that follows is for configuring the number of channels desired.

Top View of T1F-14THM Module Jumpers



Number of Channels Enabled	Jumper CH+1	Jumper CH+2	Jumper CH+3	Jumper CH+4
1				
2	X			
3		X		
4	X	X		
5			X	
6	X		X	
7		X	X	
8	X	X	X	
9				X
10	X			X
11		X		X
12	X	X		X
13			X	X
14	X	X	X	X

Power-Up Notes

Upon powering-up, it takes time for the 14THM module to detect broken transmitters. Between the time a broken transmitter is powered-up and detected, the 14THM module returns data based on radiant noise picked up by that open channel. For this reason, it is possible to receive valid data from a channel with a broken transmitter. Once a broken transmitter is detected, the EBC will return updated error information for that channel. This allows the driver to properly invalidate tags referencing that open channel.

Application Notes: D4-HSC

4 DWI (DWord In) locations share memory with the 4 DWO (DWord Out) locations as mapped below.

Mapping	Data
DWI1 <--> DWO0	Offset Value (4 bytes)
DWI2 <--> DWO1	Preset Value (4 bytes)
DWI3 <--> DWO2	Deceleration (4 bytes)
DWI5 <--> DWO3	Timebase (2 bytes but represented in 4 bytes)

These DWI and DWO locations have been designed to be Read Only and Write Only respectively. DWI is truly Read Only from the OPC Server standpoint, but DWO has been enhanced to be Read/Write for maximum flexibility. In summary:

For	Read From	Write To
Offset Value	DWI1 or DWO0	DWO0
Preset Value	DWI2 or DWO1	DWO1
Deceleration	DWI3 or DWO2	DWO2
Timebase	DWI5 or DWO3	DWO3

For further information on D4-HSC Data locations and usage, refer to the AutomationDirect D4-HSC Manual.

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

Write to address '<address>' failed. Device '<device name>': Base '<base>' : Slot '<slot>' returned extended error: Type... 55

Write to address '<address>' failed. Device '<device name>': Base '<base>' : Slot '<slot>' returned information with value... 54-55

Write to address '<address>' failed. Format version of frame received from device '<device name>' is not supported 56

Write to address '<address>' failed. Model selected for device '<device name>' does not match the actual device model 57

Write to address '<address>' failed. Device '<device name>' returned an undefined status code with value = '<value>' 53