Bristol/IP Driver

Table of Contents

Bristol/IP Driver

Help version 1.020

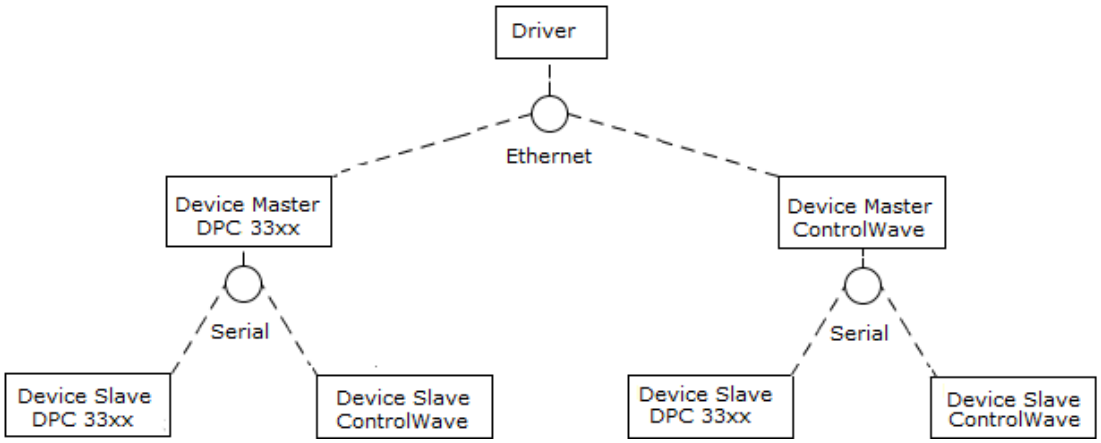
CONTENTS

Overview

The Bristol IP Driver provides a reliable way to communicate with DPC and ControlWave devices. It supports communications to both Ethernet-enabled Bristol devices and to serial Bristol devices (through an Ethernet-enabled Bristol device acting as a gateway). It supports Network 3000 Series (DPC 33xx) and ControlWave devices. For more information, refer to **Device Setup**.

BSAP Networks

The image below displays a typical Bristol IP Driver configuration. The devices are physical devices. The server channels and devices are not displayed for simplicity.



The Bristol IP Driver communicates to Master or Slave devices using Ethernet (UDP). Communications to Master devices are direct: the driver sends a request to the device and it responds. Communications to Slave devices are indirect: the driver sends a request through the Master device. In most configurations, Slave devices communicate over serial to the Master.

Note: BSAP allows six levels of hierarchy including the Master device, or five levels of hierarchy below the Master device. Each level below the Master device can have a maximum of 128 devices.

Operations By Symbolic Name vs. By Address

Bristol IP devices support reads and writes by address or by symbolic name. Reads and writes by symbolic name are usually less efficient because the request contains the signal name in ASCII format. Reads and writes by address are usually more efficient because the address is only two bytes. The driver will request by address whenever possible.

Supported BSAP Functions

The Bristol IP Driver supports a set of Bristol BSAP functions that are a subset of those available in the Remote Database Access (RDB) specification. All request types have the ability to request more than one signal at a time. For efficiency, the driver will include multiple signals in a request whenever possible.

Note: Requests can be made using either the signal's symbolic name or the signal's address (which is generally more efficient).

Operation	Description
Read Signal By Address	Read one or more signals using the two byte address.
Read Signal By Symbolic Name	Read one or more signals by signal name. Request the signal's two byte address in addition to the value.
Write Signal By Address	Write one or more signals using the two byte address.
Write Signal By Symbolic Name	Write one or more signals by signal name.

Channel Setup

Network Interface

The Network Interface property specifies the Network Interface Controller (NIC) over which the channel will communicate. Each channel must have a unique NIC and port. For example, two Bristol channels cannot be bound to the default NIC on Port 1234: only one channel can bind to the port and communicate.

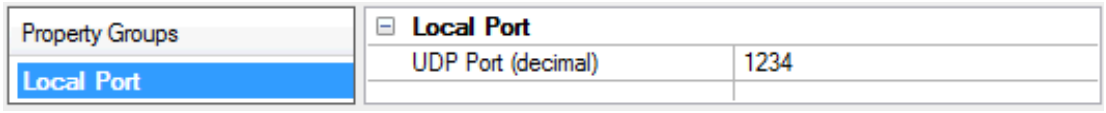
Note: The default adapter is the adapter bound to the machine hostname.

Communication Serialization

The Bristol IP Driver supports Communication Serialization, which specifies whether data transmissions should be limited to one channel at a time. For more information, refer to "Channel Properties - Advanced" in the server help file.

Local Port

The Local Port property group specifies the port that all devices under the channel will use for communications.



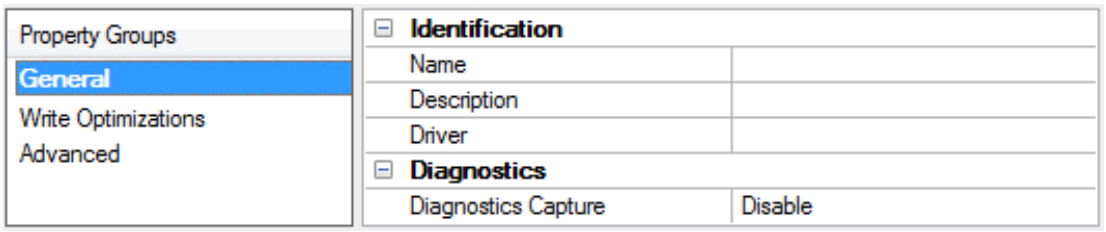
Description of the property is as follows:

- **UDP Port**: This property specifies the local port to which the channel will be bound. The default setting for most Bristol devices is 1234. Users that change the UDP Port may also need to change the device configuration to communicate with the port.

Note: This driver supports additional channel properties. For more information, refer to "What is a Channel?" in the server help file.

Channel Properties - General

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



Identification

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

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

Description: User-defined information about this channel.
Many of these properties, including Description, have an associated system tag.

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

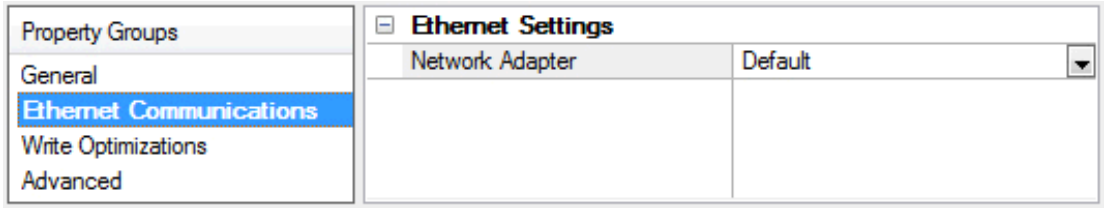
Note: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are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

Diagnostics

Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.
Note: This property is disabled if the driver does not support diagnostics.
For more information, refer to "Communication Diagnostics" in the server help.

Channel Properties - Ethernet Communications

Ethernet Communication can be used to communicate with devices.

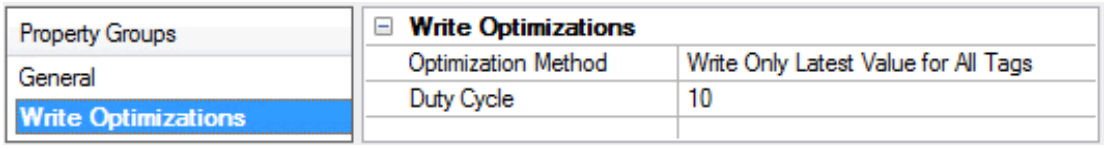


Ethernet Settings

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

Channel Properties - Write Optimizations

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



Write Optimizations

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

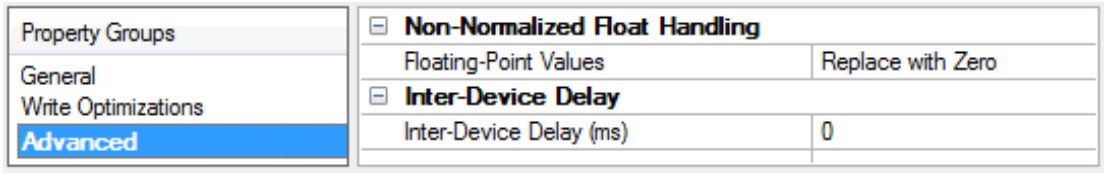
- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags**: Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are needed to reach the same final output value. In this way, no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.
  Note: This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

Duty Cycle: is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

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

Channel Properties - Advanced

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



Non-Normalized Float Handling: Non-normalized float handling allows users to specify how a driver handles non-normalized IEEE-754 floating point data. 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. 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.

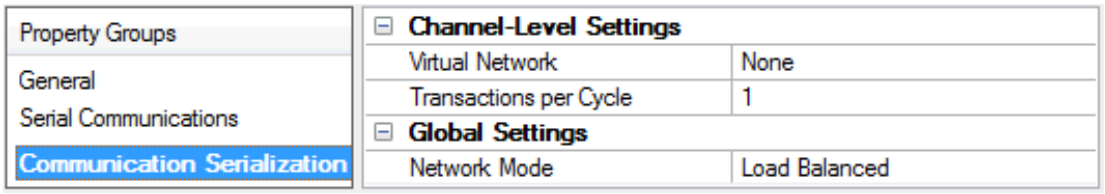
Inter-Device Delay: Specify the amount of time the communications channel waits to send new requests to the next device after data is received from the current device on the same channel. Zero (0) disables the delay.

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

Channel Properties - Communication Serialization

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

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



Channel-Level Settings

Virtual Network This property specifies the channel's mode of communication serialization. Options include None and Network 1 - Network 50. The default is None. Descriptions of the options are as follows:

- **None**: This option disables communication serialization for the channel.
- **Network 1 - Network 50**: This option specifies the virtual network to which the channel is assigned.

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

Global Settings

- **Network Mode**: This property is used to control how channel communication is delegated. In **Load Balanced** mode, each channel is given the opportunity to communicate in turn, one at a time. In **Priority** mode, channels are given the opportunity to communicate according to the following rules (highest to lowest priority):
  - Channels with pending writes have the highest priority.
  - Channels with pending explicit reads (through internal plug-ins or external client interfaces) are prioritized based on the read's priority.
  - Scanned reads and other periodic events (driver specific).

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

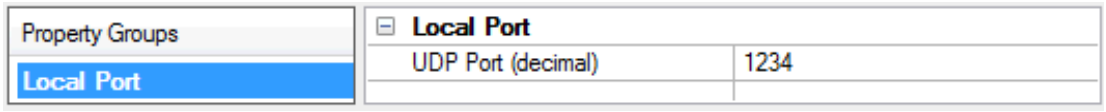
Devices that rely on unsolicited responses should not be placed in a virtual network. In situations where communications must be serialized, it is recommended that Auto-Demotion be enabled.

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

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

Channel Properties - Local Port

The Local Port property group specifies the port that all devices under the channel use for communications.



Local Port

- **UDP Port**: Specify the local port to which the channel is bound. The default for most Bristol devices is 1234. Changing the UDP Port may also require a change to the device configuration to communicate with the port.

This driver supports additional channel properties. For more information, refer to "What is a Channel?" in the server help file.

Device Setup

Supported Devices

- Network 3000 Series (DPC 33xx) devices with BSAP IP (Ethernet) support. 386EX Protected Mode DPC3330 and DPC 3335 with PES03/PEX03 or newer Firmware is required.
- ControlWave devices with BSAP IP (Ethernet) support. ControlWave Micro CWP/LPS/CWR 02.00 or newer Firmware is required.

Communication Protocol

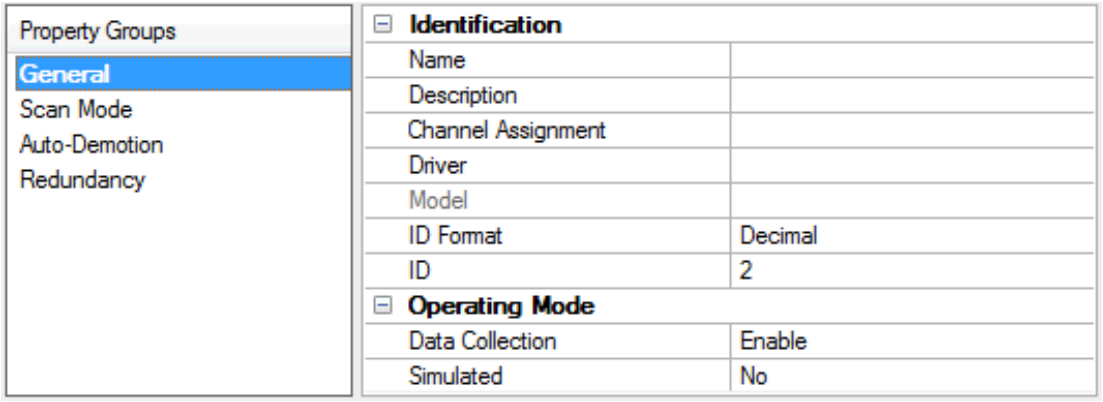
Bristol BSAP IP Protocol (UDP)

Maximum Number of Channels and Devices

The maximum number of channels supported by this driver is 256. The maximum number of devices supported per channel is 8192.

Device Properties - General

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



Identification

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

Note: Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".
For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.

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

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

Driver: Selected protocol driver for this device. This property specifies the driver selected during channel creation. It is disabled in the channel properties.

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

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

ID: This property specifies the device's 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 ID format can be Decimal, Octal, and Hexadecimal.

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

Operating Mode

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

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

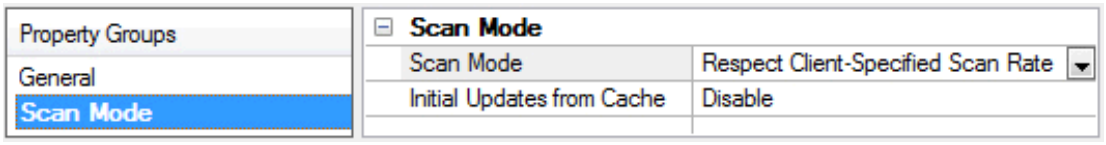
Notes:

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

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

Device Properties - Scan Mode

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



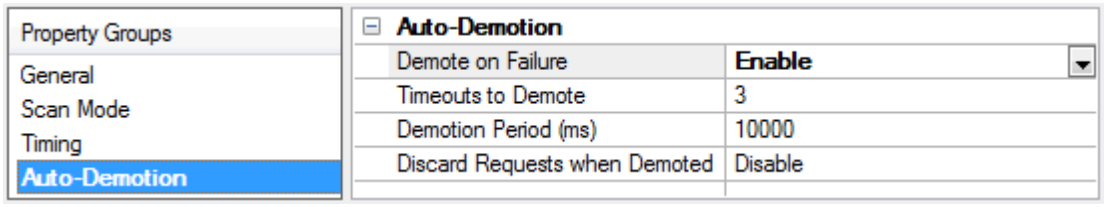
Scan Mode: specifies how tags in the device are scanned for updates sent to subscribed clients. Descriptions of the options are:

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

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

Device Properties - 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.

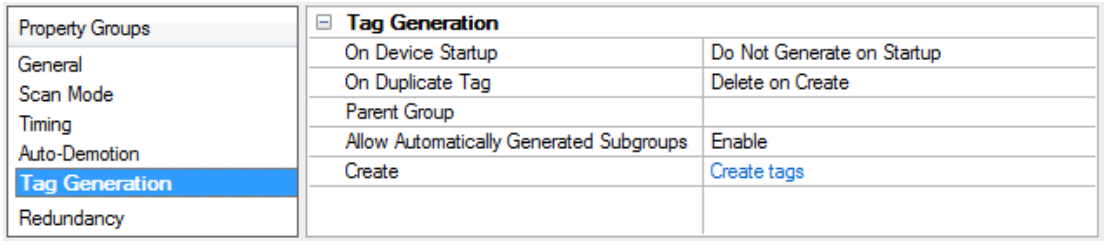
Device Properties - Tag Generation

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.

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 Device Startup

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

- **Do Not Generate on Startup**: This option prevents the driver from adding any OPC tags to the tag space of the server. This is the default setting.

- **Always Generate on Startup**: This option causes the driver to evaluate the device for tag information. It also adds tags to the tag space of the server every time the server is launched.
- **Generate on First Startup**: This option causes the driver to evaluate the target device for tag information the first time the project is run. It also adds any OPC tags to the server tag space as needed.

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

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

For example, if a user changes the I/O modules in the rack with the server configured to **Always Generate on Startup**, new tags would be added to the server every time the communications driver detected a new I/O module. If the old tags were not removed, many unused tags could accumulate in the server's tag space. The options are:

- **Delete on Create**: This option deletes any tags that were previously added to the tag space before any new tags are added. This is the default setting.
- **Overwrite as Necessary**: This option instructs the server to only remove the tags that the communications driver is replacing with new tags. Any tags that are not being overwritten remain in the server's tag space.
- **Do not Overwrite**: This option prevents the server from removing any tags that were previously generated or already existed in the server. The communications driver can only add tags that are completely new.
- **Do not Overwrite, Log Error**: This option has the same effect as the prior option, and also posts an error message to the server's Event Log when a tag overwrite would have occurred.

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

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

Allow Automatically Generated Subgroups: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting tags are named with the address value. For example, the tag names are not retained during the generation process.

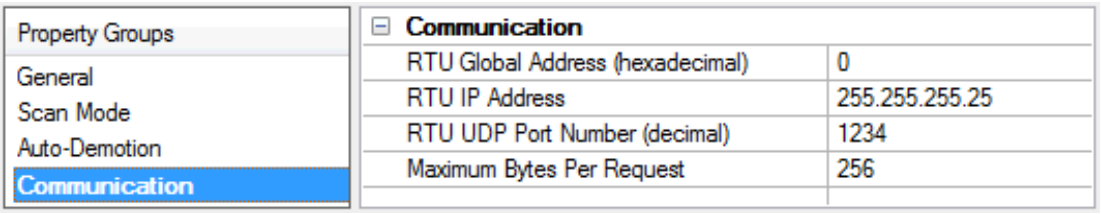
Note: If, as the server is generating tags, a tag is assigned the same name as an existing tag, the system automatically increments to the next highest number so that the tag name is not duplicated. For example, if the generation process creates a tag named "AI22" that already exists, it creates the tag as "AI23" instead.

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

Note: **Create tags** is disabled if the Configuration edits a project offline.

Device Properties - Communication

The Communication property group specifies the device's communication properties.



Communication

- **RTU Global Address**: This property specifies the hex address of the device. For Master devices, the default setting is 0. For Slave devices, the default setting is non-zero. To find the address in OpenBSI NetView, right-click on the device and select **Properties** | **BSAP**.
  Note: Although Master devices can also have a non-zero global address, users should follow convention and use the zero global address for communications.

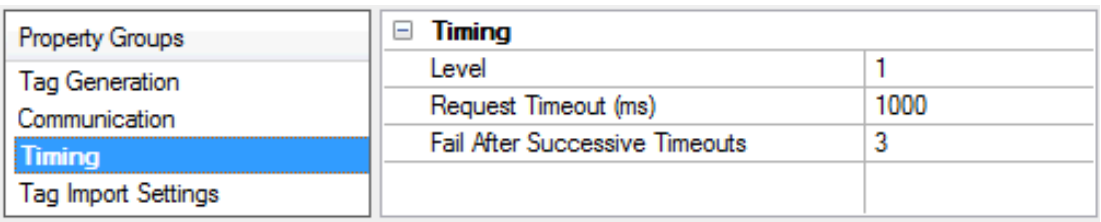
- **RTU IP Address**: This property specifies the IP address of the RTU. For Master devices, this is the IP of the Master. For Slave devices, this is the IP of the Master.
  Note: In the BSAP IP protocol, the Master device passes requests to the Slaves using the Global Address.

- **RTU UDP Port Number**: This property specifies the port number of the Master device. For Slave devices, this is the port number of the Master.

- **Maximum Bytes Per Request**: This property specifies the maximum number of bytes that will be sent to the device per request. The amount specified should depend on the device's bandwidth and performance requirements.
  Note: If communications with a device are fast and bandwidth is available, users should specify a larger number of bytes per request. If communications are slow or bandwidth is limited, users should specify a smaller amount of bytes per request.

Device Properties - Timing

The Timing group controls the device's timeout properties.



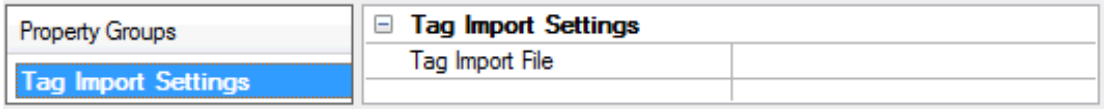
Level: Specify the device's network level. The valid range is 1 to 6. The default is 1.
Note: Level One devices are Masters and have a Global Address of 0. Level Two devices are Slaves. Level Three devices are Slaves of Slaves. The higher the level, the more latency when communicating with the device. This setting updates the "Request timeout" property.

Request Timeout: Specify the number of milliseconds that the driver will wait for a response from the device. The default is 1000 milliseconds.
Note: If a response is not received within this period, the driver will resend the request depending on the "Fail after successive timeouts" property. If the driver receives a malformed response, it will resend the request right away. Failure recovery improves when the device sends a valid response that is corrupted by network noise.

Fail After Successive Timeouts: Specify how many request timeouts can occur before the request is failed. The valid range is 1 to 10. The default is 3.

Device Properties - Tag Import Properties

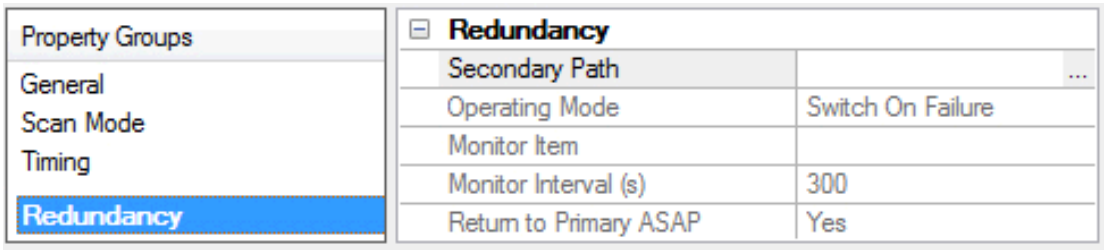
The Tag Import Properties property group specifies the tag import file that will be used in automatic tag generation.



Tag import file: Specify the device's tag import file. It must be a valid signal (*.sig) file.

*For more information, refer to **Automatic Tag Generation**.*

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

This driver supports communications with underlying BSAP child devices through a Master Ethernet device. Due to the nature of the BSAP protocol, communications to lower-level BSAP devices may be slower. As such, users may need to make special design considerations in the server configuration to attain optimal communication performance. For more information on improving performance when communicating with slower devices, refer to the subtopics below.

Channel and Device Setup

Communication to multiple devices on a channel is serialized. When communications with one device is slow, the performance of the other devices on the channel will decrease. To avoid degradation in performance, users can place slower devices on separate channels.

Note: Devices located on a common BSAP level should be grouped under the same channel in the server.

Multi-Home NIC

Separating devices onto different channels effectively improves performance, but is limited in that each channel requires a unique port and NIC combination. To eliminate the limitation, users can multi-home an NIC card and bind multiple channels to the same port.

Request Size

The device's Request Size directly impacts performance. For maximum performance, 256 bytes is generally recommended; however, a lower request size may increase performance in situations where bandwidth is limited. Lower request sizes are also generally recommended for non-Master devices. The valid range is 64 to 256 bytes.

Request Frequency

Frequently asking slower devices for data will slow down a channel's communications. To better control this, users can limit the rate at which data is requested from the device with the Scan Mode property.

Cached Data

The Bristol IP Driver does not maintain cached tag data after the client removes the tag from the server. Every time the server adds or re-adds a tag on behalf of the requesting client, the server will request the data from the device and send it to the client. This behavior ensures data freshness. Applications that require the reading of cached values should do so at the client level. Techniques for caching are as follows:

1. Have the client add the tag as inactive. Then, perform reads when fresh data is needed.
2. Have the client add the tag as active and configure the device's Scan Mode property to poll the device. Then perform cached reads when data is needed.

Data Types Description

Data Type	Description
Boolean	1 byte
Float	4-byte floating point
String	Zero-terminated ASCII character array

Address Descriptions

Bristol addresses depend on the model. For more information, refer to the subtopics below.

Network 3000 Series Devices

The syntax for Network 3000 Series devices is as follows: <*Base*>.<*Ext*>.<*Att*>.

- **Base**: This element must be 1 to 8 characters in length and begin with a letter or pound sign (#). The remaining characters will consist of letters and numbers.
- **Ext**: This element must be 0 to 6 characters in length, and consist of letters or numbers.
- **Att**: This element must be 0 to 4 characters in length, and consist of letters or numbers.

Note: An example of the syntax is "#DIAL.001.Val".

ControlWave Devices

The syntax for ControlWave devices is as follows: <*Access*>.<*Signal Name*>.

- **Access**: This element is the Bristol program name for local variables (or "%GV" for global and system variables).
- **Signal Name**: This element must contain letters, numbers, or a period and start with a letter or underscore. Signal names that start with an underscore are system variables.

Note: An example of the syntax is "%GV.MYSIGNAL".

Error Descriptions

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

<Operation> on device <device name> signal <signal> caused an overflow. Some data has been lost.

Error Type:
Error

Possible Cause:
A read or write operation on a signal caused a data overflow. The operation completed, but only part of the data was read or written to the device.

Solution:
1. If writing data to the device, verify that the data is in the correct format for the data type.
2. Verify that the String being written to the device is within the 64 character limit and is null terminated.

Note: Boolean Tags take up one byte, Float Tags take up four bytes, and String Tags can be up to 64 characters (and must be null terminated).

<Operation> on device <device name> signal <signal> failed. The device does not contain this signal.

Error Type:
Error

Possible Cause:
1. An operation was performed on a signal that is not in the device.
2. An attempt was made to write an out-of-range value to a tag in a ControlWave device.

Solution:
1. Check the tag address, and ensure that it matches the signal name in the device.
2. Check the data type of the tag in the ControlWave device. Then, ensure the value that will be written is within the range of values for that data type.

<Operation> on device <device name> signal <signal> failed. The device requires higher level permissions to perform this operation.

Error Type:
Error

Possible Cause:
The device does not have sufficient permissions to perform the operation.

Solution:
Check the permission settings for the signal in the device configuration. Then, ensure that the signal allows the operation to be performed with the highest level of permissions.

Note: The permission level is sent by the driver as part of the request, and is set to the highest level by default. This error is uncommon.

<Operation> on device <device name> signal <signal> failed. The request was malformed or incomplete.

Error Type:
Error

Possible Cause:

1. The driver is attempting to communicate with a device that is not supported.
2. Communications between the driver and device are noisy and causing malformed packets.

Solution:

1. Ensure that the device with which the driver is communicating is supported.
2. Ensure that the communication line between the driver and device is reliable.

Note: This error may indicate a problem in the driver. If the solutions listed above fail, contact Technical Support.

A read by address failed because the MSD version for device <device name> changed. The driver is sending read by name requests to get the updated MSD version from the device.

Error Type:
Warning

Possible Cause:
A configuration change occurred in the device that caused the MSD version to update. This changed the MSD address for all signals and caused the read to fail.

Solution:
The driver will reissue the read as a read-by-name and will automatically update the MSD information. No action is necessary.

Note: It is not recommended that configuration changes be made to the physical device while the driver is connected and receiving data. Users should disconnect the driver, make the configuration changes, and reconnect.

A write by address failed because the MSD version for device <device name> changed. The driver is sending a write by name request to the device.

Error Type:
Warning

Possible Cause:

A configuration change occurred in the device that caused the MSD version to update. This changed the MSD address for all signals and caused the write to fail.

Solution:

The driver will reissue the write as a write-by-name. This will complete the write, but will not update the device's MSD information. Future writes will be write-by-name until a read occurs.

Note: It is not recommended that configuration changes be made to the physical device while the driver is connected and receiving data. Users should disconnect the driver, make the configuration changes, then reconnect.

Failed to send a request to device <device>. The device is not communicating.

Error Type:

Error

The communication timed out.

Possible Cause:

1. The device is offline or the driver is not configured to communicate with the device properly.
2. The device is not supported.

Solution:

1. Ensure that the device can be reached by the driver by attempting to ping the device's IP address.
2. Ensure that the device is supported.

Failed to send a request to device <device>. The packet is malformed or too large.

Error Type:

Error

Possible Cause:

1. The driver is attempting to communicate with a device that is not supported.
2. Communications between the driver and device are noisy and are causing malformed packets.
3. The driver is attempting to write to a tag whose address length plus data length exceeds the maximum message size that is supported by the Bristol BSAP IP protocol.

Solution:

1. Ensure that the device with which the driver is communicating is supported.
2. Ensure that the communication line between the driver and device is reliable.

3. Perform a device read on the tag, then attempt a write. If the error persists, shorten the address in the device and then try again.

Note: This error may indicate a problem in the driver. If the solutions listed above fail, contact Technical Support.

Failed to stop Winsock communications.

Error Type:

Error

Winsock was placed in an invalid state. This may cause the next startup to fail.

Possible Cause:

1. Failed to cleanup Winsock while the driver was shutting down.
2. Winsock was never initialized.
3. A blocking socket call was in progress while the driver was shutting down.

Solution:

1. Restart the server and connect to a Bristol device.
2. Ensure that the operating system has the latest patches.

Failure to initiate 'winsock.dll' [Error:<error>].

Error Type:

Error

The driver cannot communicate over Ethernet without Winsock.

Possible Cause:

1. The underlying system is not ready to initialize network communications.
2. The version of Winsock requested is not supported.
3. A Winsock operation is already in progress.

Solution:

1. Restart the server.
2. Ensure that the operating system has the latest patches.

Invalid BSAP function.

Error Type:

Error

Possible Cause:

1. The BSAP function code sent to the device is invalid. The operation failed.
2. The device is not officially supported. The operation failed.
3. Communications between the driver and device are unreliable. The operation failed.

Solution:

1. Ensure that the device is supported.
2. Ensure that the communication line between the driver and device is reliable.

Read failed on device <device> address <address>. The server expects this address to be of type <type>, but it is of type <type> in the device.

Error Type:
Error

Possible Cause:
The client requested a Float Tag, but the actual type is a String. The tag read failed and the tag was set to Bad quality.

Solution:
For Static Tags, match the client-requested type to the device type by changing the tag type in the configuration. For Dynamic Tags, the client must change the request type.

Note: This error can occur any time that the client type does not match the device type. The device read succeeds, but the server cannot convert the device type to the client type.

The response received from device <device> contains an invalid sequence number. Review timing properties.

Error Type:
Warning

Possible Cause:
A response was received from the device, but the sequence number in the response packet does not match the sequence number issued in the request.
The response will be ignored. The driver will continue waiting for a valid response (depending on the Timing properties specified in Device Properties).

Solution:

1. Increase the value specified in the device's Request Timeout property.
2. Increase the value specified in the device's Fail After Successive Timeouts property.

The response received from device <device> is malformed.

Error Type:

Error

Possible Cause:

1. The device is not supported. The response will be ignored. A new request will be sent, depending on the device's timeout configuration.
2. The communications line between the driver and the device is unreliable.

Solution:

1. Ensure that the device is supported.
2. Verify that the communication line between the driver and device is reliable.

The response received from device <device> is too large (%<count> bytes). The data is lost.

Error Type:
Error

Possible Cause:
The size of the device's response is too large for the driver. The driver ignored the response.

Solution:

1. Ensure that the device with which the driver is communicating is supported.
2. Limit the size of the requests by changing the Maximum Bytes Per Request property. This will also limit the size of the response.

Note: This error is uncommon.

The signal was not found.

Error Type:
Error

Possible Cause:
The signal was not in the device. The operation on the signal failed.

Solution:
Check the device configuration and ensure that the signal is in the device and matches the tag address.

The string written to device <device>, address <address> is too long. The maximum supported string is <count> characters. The write will continue as is, but the device will truncate the value.

Error Type:
Warning

Possible Cause:
The String being written to the device is too large. The device ignored the excess characters in the String or ignored the write entirely.

Solution:
Check the maximum String size allowed for the address in the device. Then, ensure that the write does not exceed this size.

Note: The default String size is 64 characters (including the null terminator).

Unable to bind to port <port> on Channel <channel>.

Error Type:
Error

Possible Cause:
The port is already in use or is unavailable. There are no communications on the channel.

Solution:
Verify that no other processes are using the port and Network Interface Controller. To do so, run the "netstat" command in Windows. Then, change the port or network interface if possible.

Note: This error is most common in cases where channels have duplicate Network Interface Controllers and ports. This can occur between any channels; it is not limited to channels for this driver. Other programs on the machine can also be using the port.

Unable to create a socket connection for Channel <channel>.

Error Type:
Error

Possible Cause:
While initializing a channel, the driver failed to get a valid socket from the system.
There are no communications on the channel.

Solution:
Restart the server or reconnect the clients.

Note: This failure usually indicates that there is something wrong with the system (which could be out of sockets or in a Bad state).

Unable to generate a tag database for device <device name>. Reason: Import file does not contain any valid signals.

Error Type:
Error

Possible Cause:
A corrupted .sig file was imported. The tag database was not generated.

Solution:

Verify that the .sig file is valid. Then, re-import the file.

Write to device <device name> signal <signal> failed. The signal's manual inhibit attribute is set to false, not allowing writes.

Error Type:
Error

Possible Cause:
The signal is configured not to allow manual changes during Runtime; as such, write operations are also not allowed.

Solution:
Check the device configuration's manual inhibit setting for the signal, then change it accordingly. Users may want to change the Server Tag's access to read only.

Write to device <device name> signal <signal> failed. The signal is read only.

Error Type:
Error

Possible Cause:
The driver attempted to write to a device signal that is read only.

Solution:
Check the device configuration to see if the signal is read only. If it is, then change the server tag to read only. This will cause the server to stop writes before they make it to the device.

Write to device <device name> signal <signal> failed. The write value is invalid. Make sure the value is within the approved BSAP range.

Error Type:
Error

Possible Cause:

1. The String, Float, or Boolean written to the device is malformed or is too large.
2. The type being written is different than the device data type (such as when writing a String to a Float signal).

Solution:

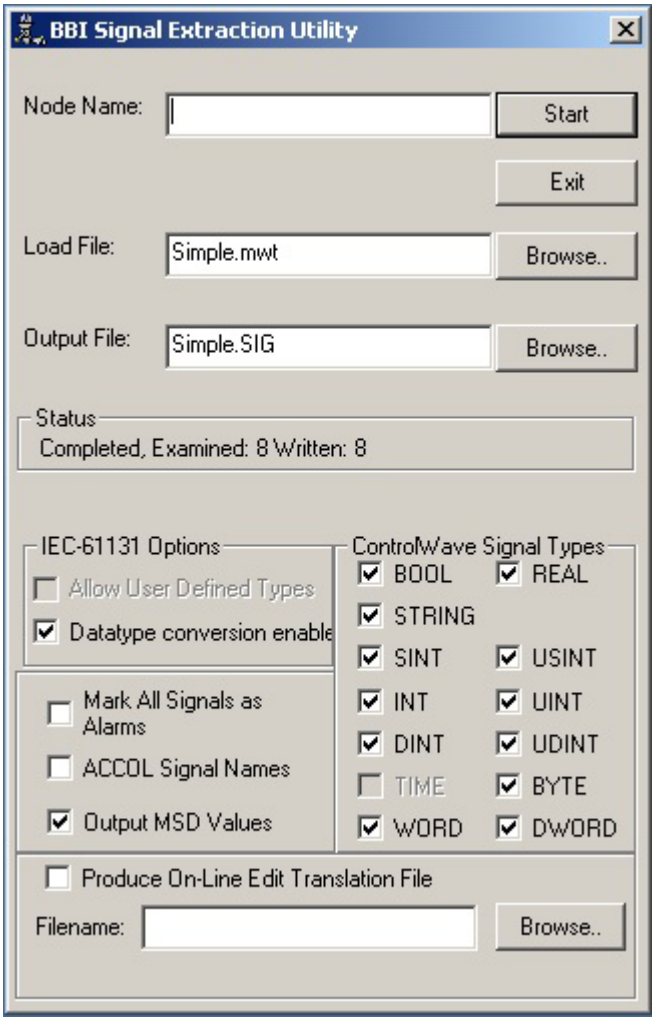
1. Ensure that the signal data type in the device matches the write.
2. If the signal is a String, ensure that the length does not exceed the device's 64-character limitation and is null terminated.

Appendix: Tag Generation

The Bristol IP Driver can automatically generate tags using a signal (*.sig) file, which is created by Bristol OpenBSI software. The procedure for creating the signal file is different for DPC 33xx devices and ControlWave devices.

DPC33xx Devices

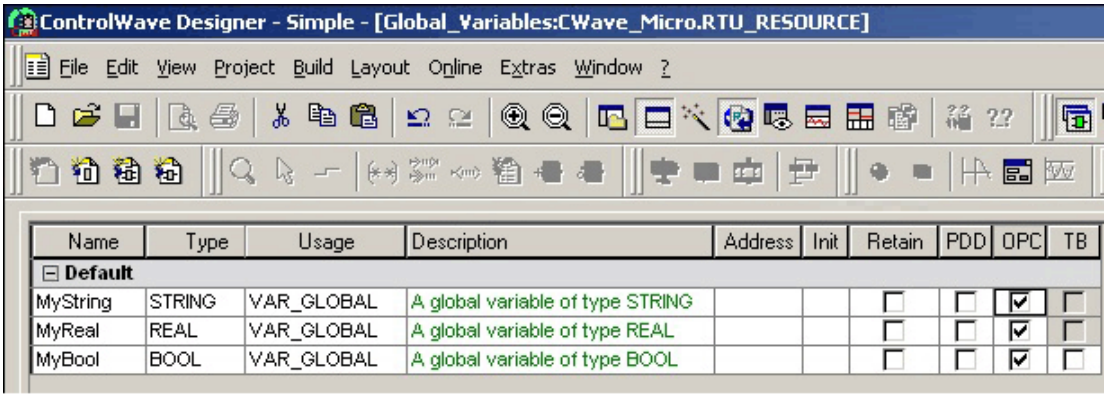
1. To start, open the **ACCOL Workbench**. Then, define a signal.
2. In **Signal Properties**, ensure that **Mark as Global** is enabled.
3. Next, launch the **BBI Signal Extraction Utility**. Then, locate **Mark All Signals as Alarms** and ensure that it is disabled.
4. In **Load File**, browse for and specify the load file.
5. In **Output File**, browse for and specify the output file.



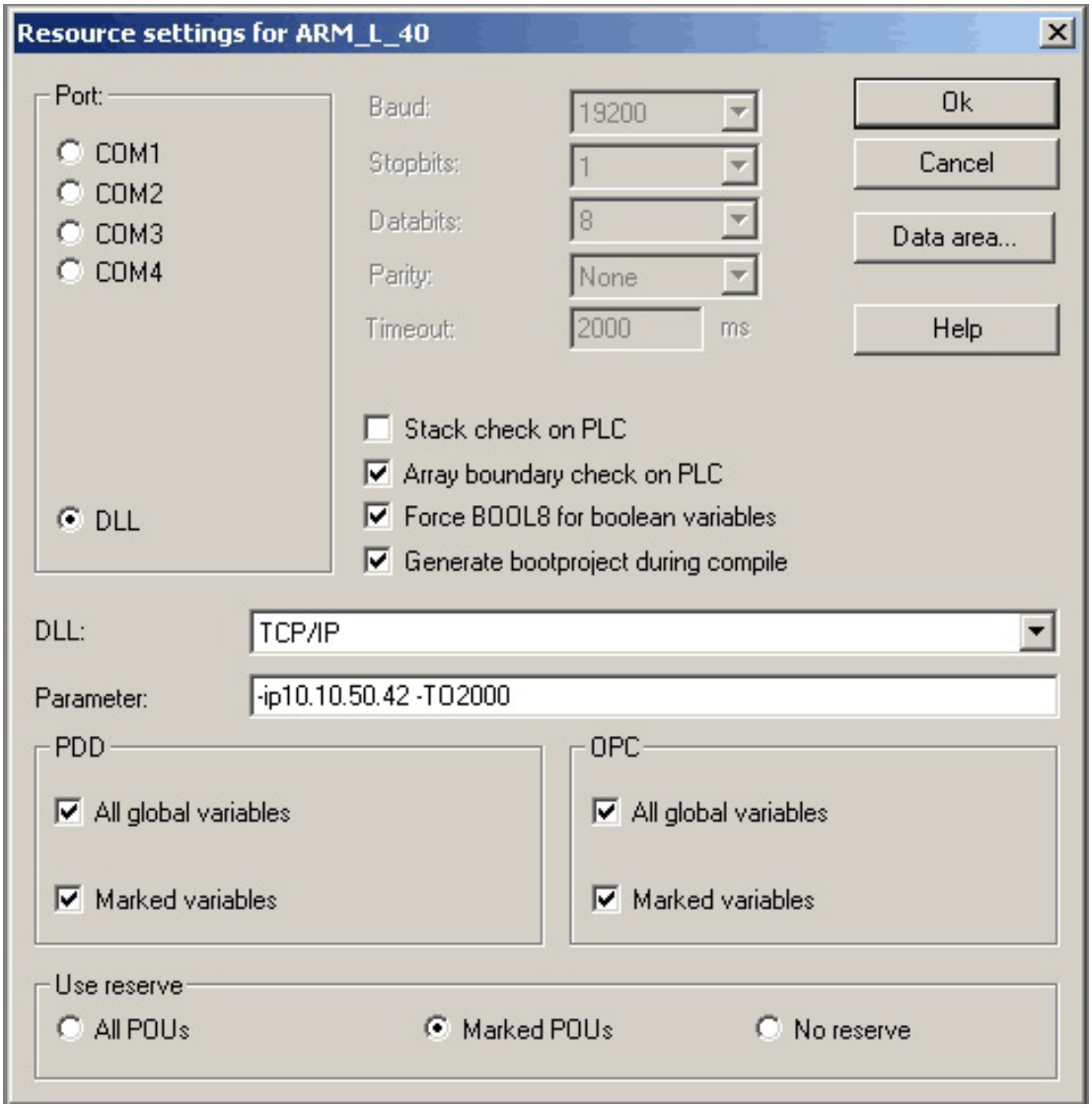
6. Once finished, click **Start** to extract the signals.

ControlWave Devices

1. To start, open **ControlWave Designer**. Then, ensure that the **OPC** column property box is enabled for all signals that will be exported.



2. Next, open the controller's **Resource Properties**.



3. Locate **PDD**, then ensure that the **Marked Variables** property is enabled.

   Note: **All Global Variables** is optional.

4. Once finished, click **OK**.
5. Choose **Start|All Programs|OpenBSI Tools|Common Tools|Signal Extractor**.
6. Select the **Browse** (...) button next to the "Load File:" and locate and select the .mwt project with the tags
7. If desired, select a different output directory by using the **Browse** (...) button next to the "Output File:" and locate the desired location
8. Select **Start** and the tag file is exported to the selected folder.
9. Place the exported signal file where the server can access it.
10. Enter the signal file into the "Tag import file" field in the **Tag Import Properties** property group of the device properties.
11. The signals can now be created with the database creation functionality.

Index

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W