Allen-Bradley ControlLogix Unsolicited Driver

Table of Contents

Allen-Bradley ControlLogix Unsolicited Driver

Help version 1.019

CONTENTS

Overview

The Allen-Bradley ControlLogix Unsolicited Driver provides a reliable way to connect Allen-Bradley ControlLogix PLCs to client applications. It simulates a ControlLogix 5000 series rack containing a single EtherNet/IP module and up to 16 ControlLogix CPUs. ControlLogix 5000 series PLCs can be configured to perform CIP Data Table Read/Writes to the driver with the MSG Ladder Instruction.

For more information on configuring the ControlLogix 5000 series PLC to communicate with the driver, refer to Rockwell/Allen Bradley documentation.

Setup

The Allen-Bradley ControlLogix Unsolicited Driver acts as a simulated ControlLogix 5000 series rack that contains a single Ethernet/IP module. The rack can contain up to sixteen ControlLogix CPUs, with one built into the EtherNet/IP module (considered local) and up to fifteen individual CPU modules (considered remote to the EtherNet/IP module). Up to 256 devices may connect to the simulated EtherNet/IP module at any time.

Supported Devices

All ControlLogix 5000 Series PLCs that support CIP Data Table Read/Write MSG Instructions and run Firmware revision 16 or higher. Both Connected and Unconnected CIP Data Table Reads or Writes are supported.

Communication Protocol

EtherNet/IP

PLC Configuration

Devices on the network must be programmed to use the CIP Data Table Read/Write MSG Instruction to the driver, as well as to handle returned data.

For more information on configuring the MSG Instruction, refer to Allen-Bradley's Programming Messages In a ControlLogix System.

Sockets

Up to 256 incoming connections are serviced simultaneously. The connections remain open until closed by the originator.

See Also:

Channel Setup
Device Setup

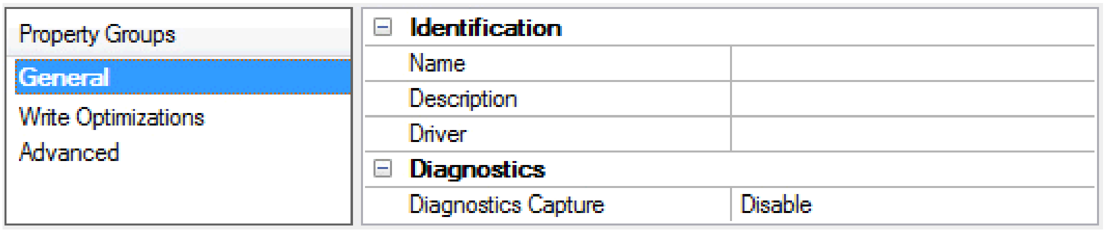
Channel Setup

Channel setup includes configuration of the following property groups:

General
Ethernet Communications
Write Optimizations
Advanced
EtherNet/IP Module

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.



Property Groups	Identification	
General	Name	
Write Optimizations	Description	
Advanced	Driver	
	Diagnostics	
	Diagnostics Capture	Disable

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.

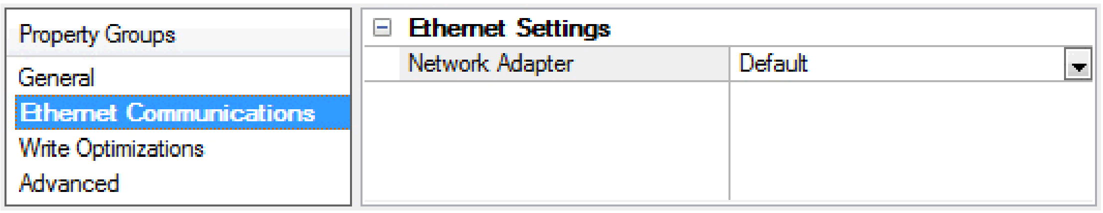
● **Note**: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items 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.



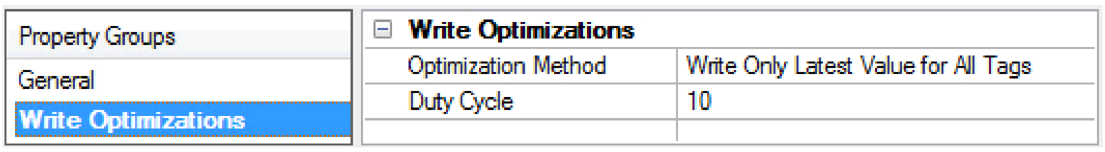
Property Groups	Ethernet Settings	
General	Network Adapter	Default
Ethernet Communications		
Write Optimizations		
Advanced		

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.



Property Groups: General, Write Optimizations

Write Optimizations	
Optimization Method	Write Only Latest Value for All Tags
Duty Cycle	10

Write Optimizations

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

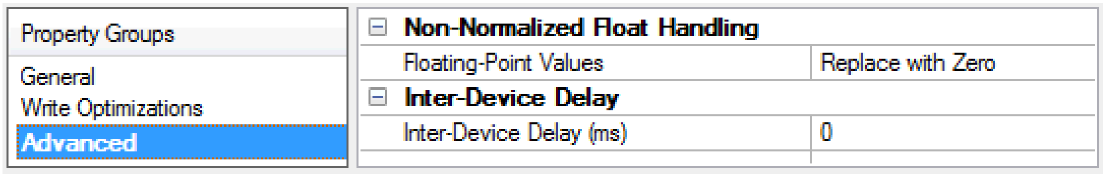
- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing 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	
Floating-Point Values	Replace with Zero
Inter-Device Delay	
Inter-Device Delay (ms)	0

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.

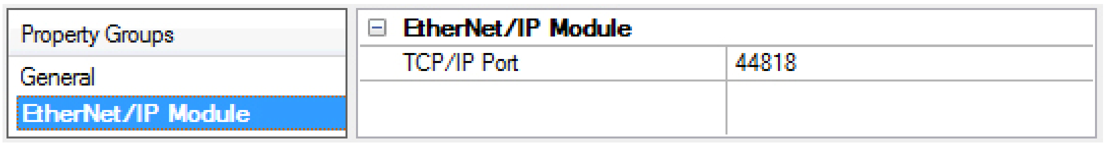
Note: This property is disabled if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM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



EtherNet/IP Module	
TCP/IP Port	44818

TCP/IP Port: Specify the TCP/IP and UDP port that provides a unique communication channel to the EtherNet/IP module. The valid range is 1 to 65535. The default is 44818.

Note: The Allen-Bradley ControlLogix Unsolicited Driver currently limits the number of channels to one. If the network adapter and port conflicts with another application on the host machine, the driver cannot accept inbound EtherNet/IP connections.
*For more information, refer to **Event Log Messages**.*

Master Device Configuration

Allen-Bradley ControlLogix PLCs must be programmed to issue CIP Data Table Read/Write messages to this driver using the MSG Ladder Instruction. A routing path representing the driver's configuration should be used that includes the IP Address, slot number, and optional port. *For more information on the MSG Ladder Instruction, refer to the Rockwell/Allen-Bradley PLC programming documentation.* The routing path that is associated with a particular slave device is provided in the Controller Module under Device Properties. *For more information, refer to **Device Properties - Controller Module**.*

Supported Services

Unfragmented Read
Fragmented Read
Unfragmented Write
Fragmented Write
Read/Modify/Write

Note: The ControlLogix MSG Ladder Instruction automatically decides whether to use Fragmented or Unfragmented services based on the size of the request. This is not a user configurable option.

Supported Logix Types

BOOL
DWORD (BOOL array)
SINT
INT
DINT
LINT
REAL

Error Codes

The Allen-Bradley ControlLogix Unsolicited Driver responds to all properly formatted messages that it receives. If it cannot complete the request, a response message is returned with a non-zero error status and an optional extended error status in the ERR and EXERR tags of the MESSAGE structure. Ladder programs should be written to handle these errors.

*For more information on the error codes that may be returned to master devices, refer to **Error Codes**.*

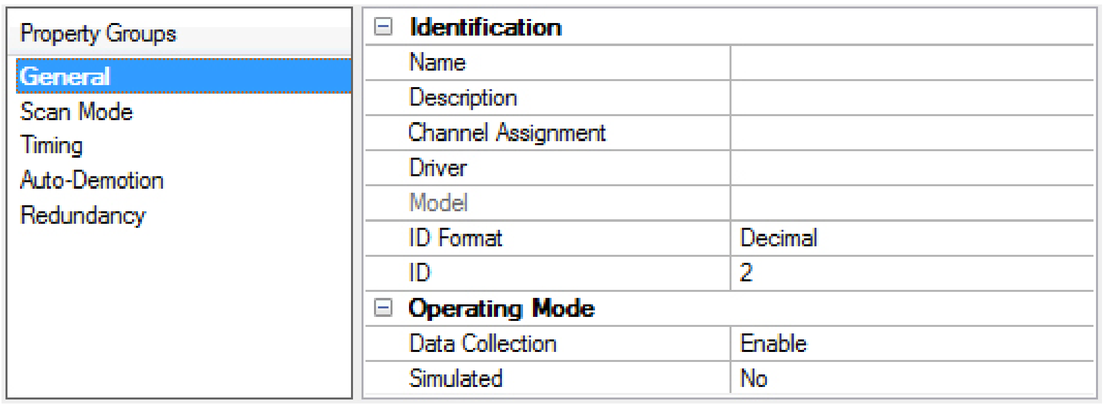
Note: This driver supports CIP Data Table Read/Writes for the Logix Atomic Types list above. Although structured types are not supported, the MSG Ladder Instruction can be used to Read/Write to individual Logix Atomic Types within a structured type. For example, if the tag "MyString @ STRING" must be written to the driver, a CIP Data Table Read must be performed for "MyString.DATA" and "MyString.LEN" separately.

Device Setup

Device setup includes configuration of the following property groups:

General
Scan Mode
Auto Demotion
Controller Module
Native Tag Database
Options

Device Properties — General



Identification	
Name	
Description	
Channel Assignment	
Driver	
Model	
ID Format	Decimal
ID	2
Operating Mode	
Data Collection	Enable
Simulated	No

Identification

Name: User-defined identity of this device.

Description: User-defined information about this device.

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

Driver: Selected protocol driver for this device.

Model: The specific version of the device.

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.

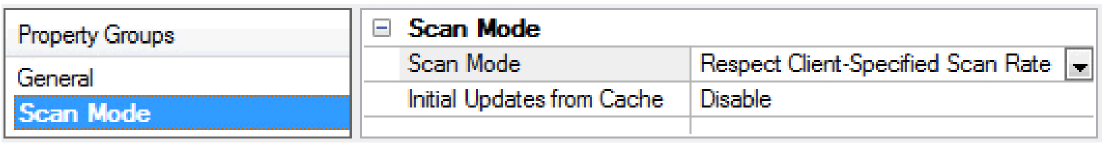
Notes:

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

Scan Mode	Respect Client-Specified Scan Rate
Initial Updates from Cache	Disable

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.

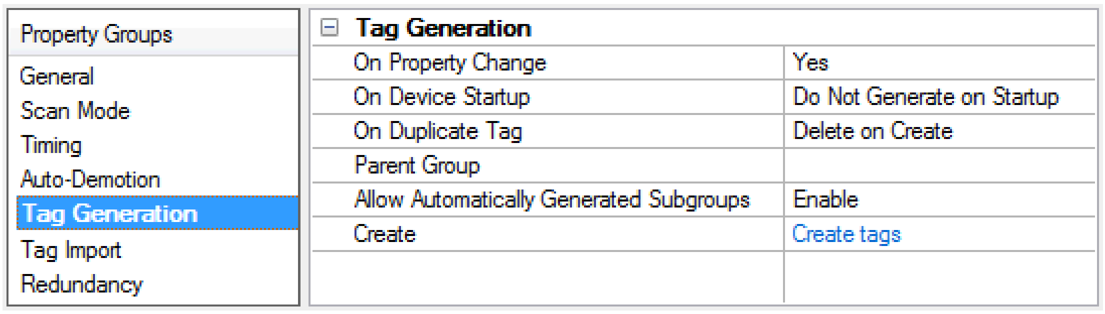
Device Properties — Tag Generation

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.



Tag Generation	
On Property Change	Yes
On Device Startup	Do Not Generate on Startup
On Duplicate Tag	Delete on Create
Parent Group	
Allow Automatically Generated Subgroups	Enable
Create	Create tags

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: 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.

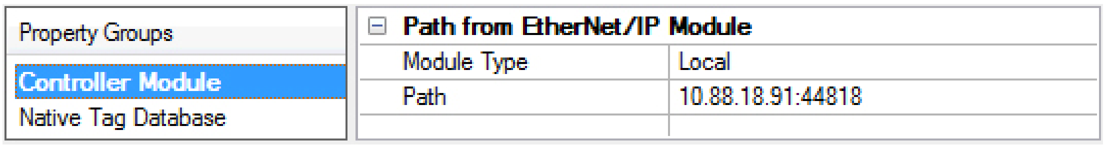
Allow Automatically Generated Subgroups: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting 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 — Controller Module



Path from EtherNet/IP Module	
Module Type	Local
Path	10.88.18.91:44818

Module Type: Verify if the device is Local (part of the simulated EtherNet/IP Module) or Remote (which require a slot number for EtherNet/IP routing). There can be one Local CPU and up to fifteen Remote CPUs.

- **Local**: When Local, the Controller Module is treated as a CPU local to the simulated EtherNet/IP Module. There can only be one Local Controller Module per channel. The default is enabled.

- **Remote**: When Remote, the Controller Module is treated as a CPU separate from the simulated EtherNet/IP Module. There can be up to fifteen remote controller modules per channel. When this option is enabled, a slot must also be specified. The default is disabled.

  Notes:

  1. Each slave device must be configured to represent a ControlLogix 5000 Series controller.
  2. This is disabled and forced to Remote (1) if another device is already configured as Local.

Slot: This property is part of the routing path to the Controller Module. It only contains slots that are currently available for the channel/device being configured. When a new slot is specified, the previous slot is made available for use in another device.

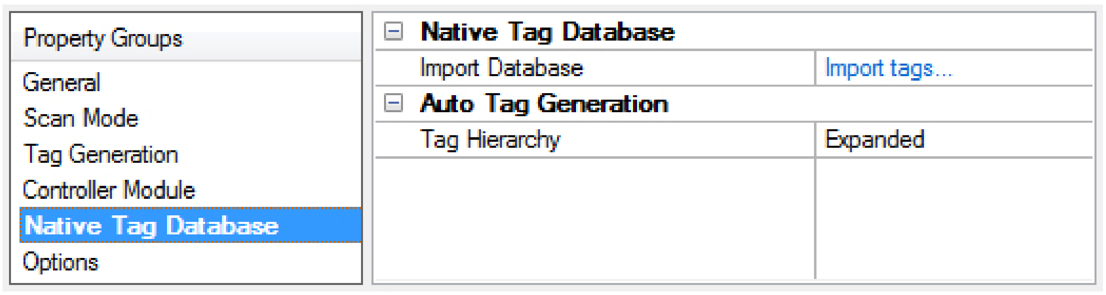
Path: This property represents the routing path to the Controller Module from the master device. It should be used during Master Device Configuration in the PLC.

*For more information on configuring master devices, refer to **Master Device Configuration**.*

Device Properties — Native Tag Database

The Allen-Bradley ControlLogix Unsolicited Driver can be configured to automatically generate a list of server tags that correspond to the Native Tag Database. Native Tag Database tags must be pre-defined Logix Atomic Types, but may also be a part of a structured type.

The driver generates a server tag for each Atomic Tag defined in the Native Tag Database. For array types, a server tag is defined for each element of the array. Array Tags can quickly increase the number of tags imported, as well as the number of tags available in the server. Automatically generated tags are always configured with a client access of Read/Write.
*For more information, refer to **Native Tag Database CSV Import**.*



Native Tag Database	
Import Database	Import tags...
Auto Tag Generation	
Tag Hierarchy	Expanded

Native Tag Database

The Native Tag Database dialog is used to collect data that the simulated Controller Module represents, in addition to automatically generating server tags. The database is imported as a comma-separated-value (CSV) file.
*For more information, refer to **Native Tag Database CSV Import**.*

Import Database: Click Import tags... to locate and select a CSV file for Native Tag Database import. Native tags cannot be imported when there are active client connections to the server.
Note: Once the Native Tag Database is imported, the server project file is used to maintain it. The CSV file is not needed for remote deployment. *For more information, refer to* Automatic Tag Database Generation.

Auto Tag Generation

Tag Hierarchy: Select if the tag display "tree" organization should be Expanded or Condensed. The default is Expanded.

- **Expanded**: generated client tags are grouped similarly to RSLogix, with groups created for each segment (following a period in the tag address), structure, substructure, and array.
- **Condensed**: generated client tags are grouped similarly to the tag addressing, with groups created for each segment preceding a period.

For more information on the Native Tag Database CSV format required for import, refer to ***Native Tag Database CSV Import***.

Tag Hierarchy

The automatically generated server tags can follow one of two hierarchies: Expanded or Condensed. The default is Expanded Mode.

Expanded Mode

In Expanded Mode, the automatically generated server tags follow a Group/Tag hierarchy consistent with the tag hierarchy in RSLogix5000. Groups are created for each segment that precedes a period, and are also created in logical groupings. Groups created include the following:

- Global (Controller) Scope
- Structures and Substructures
- Arrays

Note: Groups are not created for .bit addresses.

Basic Global Tags

Basic Global Tags (or non-structure, non-array tags) are placed under the Global group. Each Structure and Array Tag is provided with its own subgroup of the parent group. By organizing the data in this fashion, the server's Tag View mimics RSLogix5000.

Note: The name of the Structure/Array subgroup also provides a description. For example, an array "tag1 [1,6]" defined in the controller has a subgroup name "tag1[x,y]". In this example, *x* signifies that dimension 1 exists, and *y* signifies that dimension 2 exists. Furthermore, the tags within an array subgroup are the elements of that array unless explicitly limited. The tags within a structure subgroup are the structure members themselves. A structure that contains an array has an array subgroup of the structure group created as well.

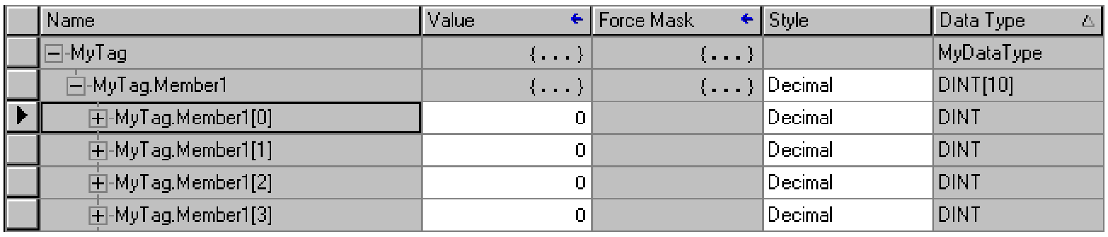
Array Tags

A group is created for each array that contains the array's elements. Group names have the notation *<array name>[x,y,z]*, where:

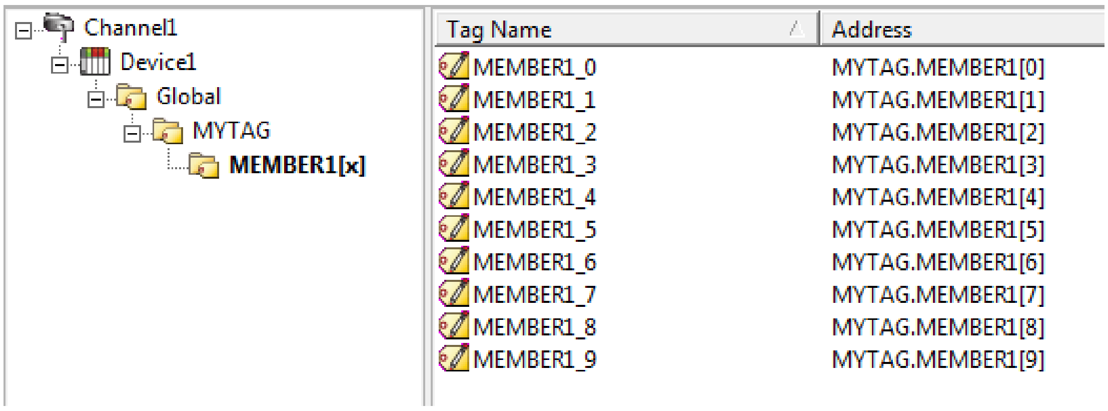
- **[x,y,z]:** 3 dimensional array.
- **[x,y]:** 2 dimensional array
- **[x]:** 1 dimensional array

Note: Array Tags have the notation *<tag element>_XXXXX_YYYYY_ZZZZZ*. For example, element "tag1 [12,2,987]" has the tag name "tag1_12_2_987".

Simple Example



Name	Value	Force Mask	Style	Data Type
⊟-MyTag	{...}	{...}		MyDataType
⊟-MyTag.Member1	{...}	{...}	Decimal	DINT[10]
⊞-MyTag.Member1[0]	0		Decimal	DINT
⊞-MyTag.Member1[1]	0		Decimal	DINT
⊞-MyTag.Member1[2]	0		Decimal	DINT
⊞-MyTag.Member1[3]	0		Decimal	DINT



- Channel1
  - Device1
    - Global
      - MYTAG
        - **MEMBER1[x]**

Tag Name	Address
MEMBER1_0	MYTAG.MEMBER1[0]
MEMBER1_1	MYTAG.MEMBER1[1]
MEMBER1_2	MYTAG.MEMBER1[2]
MEMBER1_3	MYTAG.MEMBER1[3]
MEMBER1_4	MYTAG.MEMBER1[4]
MEMBER1_5	MYTAG.MEMBER1[5]
MEMBER1_6	MYTAG.MEMBER1[6]
MEMBER1_7	MYTAG.MEMBER1[7]
MEMBER1_8	MYTAG.MEMBER1[8]
MEMBER1_9	MYTAG.MEMBER1[9]

Complex Example
A Logix Tag defined with the address "MyStructArray[0].MySubStruct.Data" would be represented in the following groups: "Global," "MYSTRUCTARRAY[x]," "MYSTRUCTARRAY[0]," and "MYSUBSTRUCT". The tag "DATA" would be in the last group. The static reference to "DATA" would be "Channel1.Device1.Global.MYSTRUCTARRAY[X].MYSTRUCTARRAY[0].MYSUBSTRUCT.DATA". The dynamic reference would be "Channel1.Device1. MyStructArray[0].MySubStruct.Data".
For more information, refer to "Static Tags (User-Defined)" and "Dynamic Tags" in server help file.

Condensed Mode

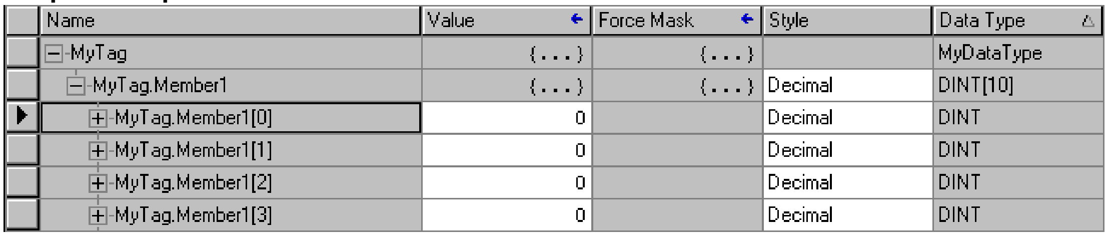
In Condensed Mode, the automatically generated server tags follow a group/tag hierarchy consistent with the tag's address. Groups are created for each segment that precedes the period. Groups created include the following:

- Program Scope
- Structures and Substructures

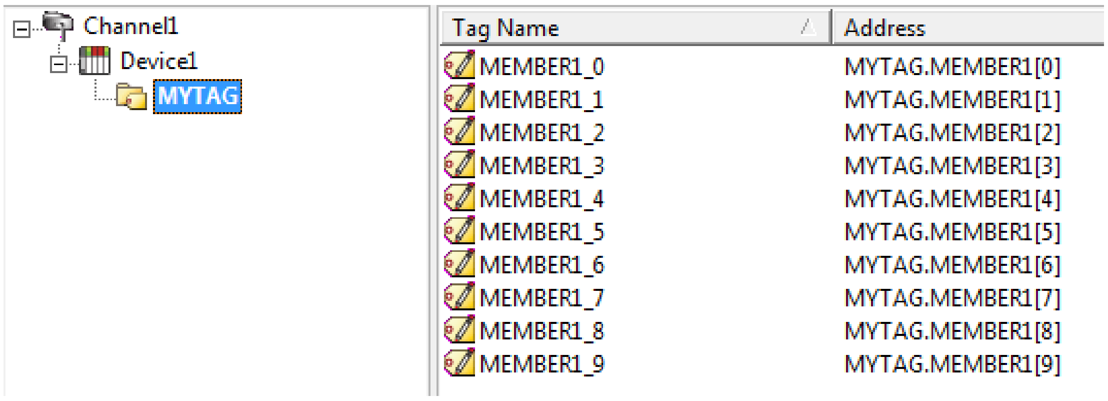
Notes:

1. Groups are not created for arrays or .bit addresses.
2. Tag or structure member names that start with an underscore are converted to "U". This is required because the server does not support leading underscores in tag name fields.

Simple Example



Name	Value	Force Mask	Style	Data Type
⊟-MyTag	{...}	{...}		MyDataType
⊟-MyTag.Member1	{...}	{...}	Decimal	DINT[10]
⊞-MyTag.Member1[0]	0		Decimal	DINT
⊞-MyTag.Member1[1]	0		Decimal	DINT
⊞-MyTag.Member1[2]	0		Decimal	DINT
⊞-MyTag.Member1[3]	0		Decimal	DINT



Tag Name	Address
MEMBER1_0	MYTAG.MEMBER1[0]
MEMBER1_1	MYTAG.MEMBER1[1]
MEMBER1_2	MYTAG.MEMBER1[2]
MEMBER1_3	MYTAG.MEMBER1[3]
MEMBER1_4	MYTAG.MEMBER1[4]
MEMBER1_5	MYTAG.MEMBER1[5]
MEMBER1_6	MYTAG.MEMBER1[6]
MEMBER1_7	MYTAG.MEMBER1[7]
MEMBER1_8	MYTAG.MEMBER1[8]
MEMBER1_9	MYTAG.MEMBER1[9]

Complex Example
A Logix Tag defined with address "MyStructArray[0].MySubStruct.Data" would be represented in the following groups: "MYSTRUCTARRAY[0]"and "MYSUBSTRUCT". The tag "DATA" would be in the last group. The static reference to "DATA" would be "Channel1.Device1.MYSTRUCTARRAY[0].MYSUBSTRUCT.DATA" and the dynamic reference would be "Channel1.Device1.MyStructArray[0].MySubStruct.Data".

Native Tag Database CSV Import

A CSV file specifies the Native Tags that each device can represent. It is used once for tag import, and is not required for automatic tag database generation or remote deployment. The following CSV header must be used for Native Tag import:

Logix Address	Logix DataType	External Access	Description

Tip: A template Native Tag Database CSV file is included for reference in <*Server Installation Directory*>*/Drivers/controllogix_unsolicited_ethernet/import_template.csv*.

Importing Native Tags as SINT, INT, and DINT Arrays

Native Tags that are imported as SINT, INT, and DINT arrays also have a string tag defined that uses the number of elements of the corresponding array in the tag address. Examples are as follows:

- If a Native Tag called "MySINTarray @ SINT[100]" is imported, a Static Tag with the address "MYSINTARRAY / 100" and String data type is generated.
- If a Native Tag called "MyINTarray @ INT[100]" is imported, a Static Tag with the address "MYINTARRAY / 100" and String data type is generated.
- If a Native Tag called "MyDINTarray @ DINT[100]" is imported, a Static Tag with the address "MYDINTARRAY / 100" and String data type is generated.

Tip: To import RSLogix5000 pre-defined Strings, the two elements contained within the String types ("STRING.DATA" and "STRING.LEN") should be defined in the Native Tag Database CSV file before the import is performed.

Logix Address

Restrictions on the Logix Address are consistent with RSLogix5000 requirements, which correspond to the following IEC 1131-3 identifier rules:

- Must begin with an alphabetic character (A-Z, a-z) or an underscore.
- Can only contain alphanumeric characters and underscores.
- Can have as many as 40 characters in each segment.

- Cannot have consecutive underscores.
- Are not case sensitive.

Tags that do not have a unique Logix Address or meet the identifier requirements above fail to import, causing a message with the specified Logix Address to be posted to the server's Event Log.

Logix DataType

The following pre-defined Logix Atomic Types are supported:

Logix DataType	Supported Data Types
BOOL	Boolean
SINT	Char, Byte
INT	Short, Word
DINT	Long, DWord
LINT	Double, Date
REAL	Float

Note: Other pre-defined or user-defined complex (structured) data types are not supported. Structured data can be imported by qualifying the Logix Address down to the atomic type. For example, there exists a structured type called TIME, which is described as the following:

TIME
{
HOUR @ SINT
MIN @ SINT
SEC @ SINT
}

The structure can be broken down and the atomic member imported as "TIME.HOUR," "TIME.MIN," and "TIME.SEC" with the associated Logix DataType, External Access, and Description following the CSV format outlined above. All unsupported Logix DataType values specified in the CSV import are defaulted to DINT so that the import succeeds.

External Access

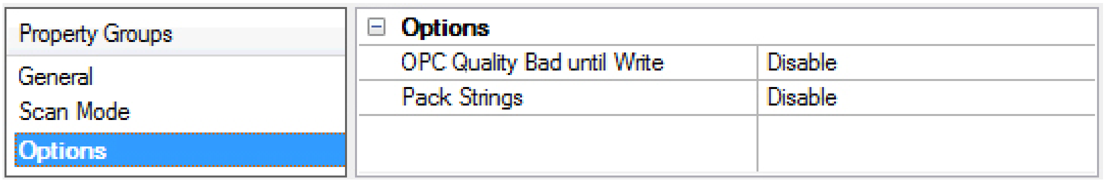
The External Access specifies the master devices' Read/Write privileges. This access does not apply to client tags, which always have a client access of Read/Write by default. The following external access types are supported: all other values specified are set to Read/Write.

- **R/W:** Master Devices have Read/Write permissions to the Native Tag.
- **RO:** Master Devices have Read Only permissions. All write attempts fail with the appropriate error (CIP error 0x0F).

Description

Descriptions are used during automatic tag database generation, truncated to 64 characters. The Description field must be present, but may be left blank.

Device Properties — Options



Options	
OPC Quality Bad until Write	Disable
Pack Strings	Disable

OPC Quality Bad until Write: Select Enable to force the driver to return Bad quality until a write occurs to the Native Tag. The write may occur from a client interface (such as OPC) or from a Master Device (such as a ControlLogix 5000 Series PLC). When a write occurs to a single item of an array, the entire array is initialized and Good quality is returned. The default is disabled. At startup, the Allen-Bradley ControlLogix Unsolicited Driver initializes integer/numeric data type values to zero (0) and strings to empty. Clients receive initial updates with Good quality by default; however, this behavior can be modified for each device.

Pack Strings: Enabled for strings displayed in the String Tag to include all bytes of the array elements in a packed format. When disabled, the string displayed in the String Tag is in an unpacked format, where only the low byte of each element is displayed.

Data Types Description

Data Types	Description
Boolean	Single bit
Char	Signed 8-bit value bit 0 is the low bit bit 6 is the high bit bit 7 is the sign bit
Byte	Unsigned 8-bit value bit 0 is the low bit bit 7 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
Word	Unsigned 16-bit value bit 0 is the low bit bit 15 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
DWord	Unsigned 32-bit value bit 0 is the low bit bit 31 is the high bit
Float	32-bit floating point value bit 0 is the low bit bit 31 is the high bit
Double	64-bit floating point value bit 0 is the low bit bit 63 is the high bit
String	Typically null terminated, null padded, or blank padded ASCII string.
Date	64-bit floating point value.

For a description of Logix platform-specific data types, refer to ***Address Descriptions***.

Address Descriptions

The Allen-Bradley ControlLogix Unsolicited Driver supports symbolic tag-based addressing.

Logix Tag-Based Addressing

This driver uses a tag or symbol-based addressing structure that is commonly referred to as Logix or Native Tags (which is consistent with Rockwell Automation's Integrated Architecture). These tags differ from conventional PLC data items in that the tag name is the address, not a physical or logical address.

The driver allows users to access the controller's atomic data types BOOL, SINT, INT, DINT, LINT, and REAL. Although some of the pre-defined types are structures, they are ultimately based on these atomic data types. As such, all non-structure (atomic) members of a structure are accessible. For example, a TIMER cannot be assigned to a server tag but an atomic member of the TIMER can be (such as TIMER.EN, TIMER.ACC, and so forth). If a structure member is a structure itself, both structures must be expanded to access an atomic member of the substructure. This is more common with user and module-defined types, and is not found in any of the pre-defined types.

Atomic Data Type	Description		Range
BOOL	Single bit value	VT_BOOL	0, 1
SINT	Signed 8-bit value	VT_I1	-128 to 127
INT	Signed 16-bit value	VT_I2	-32,768 to 32,767
DINT	Signed 32-bit value	VT_I4	-2,147,483,648 to 2,147,483,647
LINT	Signed 64-bit value	VT_R8	-9.22337E18 to 9.22336E18
REAL	32-bit IEEE floating point	VT_R4	1.1755 E-38 to 3.403E38 0 -3.403E-38 to -1.1755

Client/Server Tag Address Rules

Logix Tag names correspond to Client/Server Tag addresses. Logix Tag names, which are entered via RSLogix5000, follow the IEC 1131-3 identifier rules. Client/Server Tag addresses follow these same rules. They are as follows:

- Must begin with an alphabetic (A-Z, a-z) character or an underscore.
- Can only contain alphanumeric characters and underscores.
- Can have as many as 40 characters in each segment.
- Cannot have consecutive underscores.
- Are not case sensitive.

Notes

1. Tag name assignment in the server differs from address assignment in that names cannot begin with an underscore.
2. For tags to be properly validated, a Native Tag that represents the Static Client Tag must exist in the Native Tag Database.

Address Formats

There are several ways to address a Logix Tag statically in the server or dynamically from a client. The selected format depends both on the type of tag and how it is used. For example, when accessing a bit within a SINT-type tag, the bit format would be used.

For more information on address format and syntax, refer to the table below.

Note: All formats are native to RSLogix5000 except for Array and String. When referencing an atomic data type, users can copy an RSLogix5000 tag name and then paste it into the server's Tag Address parameter: it is valid as long as the corresponding Native Tag exists in the Native Tag Database.

Format	Syntax	Example	Notes
Standard	<Logix Tag Name>	tag_1	The tag cannot be an array.
Array Element	<Logix Array Tag Name> [dim 1, dim2, dim 3]	tag_1 [2, 58, 547] tag_1 [0, 3]	Dimension Range = 1 to 3. Element Range = 0 to 65535.

Array w/o Offset*	<Logix Array Tag Name> {# columns}
	<Logix Array Tag Name> {# rows}{# columns}
		tag_1 {8}
		tag_1 {2}{4}
			Dimension Range = 1 to 2.
			Element Range = 1 to 65535.
			
			The number of elements to Read/Write equals the # of rows times the # of columns. If no rows are specified, the number of rows default to 1.
			
			The array begins at a zero offset (array index equals 0 for all dimensions).

Array w/ Offset*	<Logix Array Element Tag> {# columns}
	<Logix Array Element Tag> {# rows}{# columns}
		tag_1 [2, 3] {10}
		tag_1 [2, 3] 2}{5}
			The array begins at an offset specified by the dimensions in the Array Element Tag. The array always covers the highest dimension. Thus, "tag_1[2,3]{10}" would produce an array of elements tag_1[2,3] -> tag_1 [2,13].

Bit	<Logix Tag Name>.bit <Logix Tag Name>.[bit]	tag_1.0 tag_1.[0]	Bit Range = 0 to 31. If the tag is an array, it must be a BOOL array; otherwise, tag cannot be an array.
String	<Logix Tag Name>.Data/<Maximum string length>	tag_1.Data/4	Length Range = 1 to 65535. The maximum number of characters that can Read/Write to the string.

*Because this format may request more than one element, the order in which array data is passed depends on the dimension of the Logix Array Tag. For example, if the rows multiplied by the columns is 4 and the Controller Tag is a 3X3 element array, then the elements that are being referenced are "array_tag [0,0]," "array_tag [0,1]," "array_tag [0,2]," and "array_tag [1,0]" in that order. The results would be different if the Controller Tag were a 2X10 element array.

*For more information on how elements are referenced for 1, 2, and 3 dimensional arrays, refer to **Ordering of Logix Array Data**.*

Tag Scope

Global Tags

Global Tags are Logix Tags that have global scope in the controller. Any program or task can access Global Tags; however, the number of ways a Global Tag can be referenced depends on its Logix data type and the address format being used.

Program Tags

Program Tags are identical to Global Tags except that their scope is local to the program in which it is defined. The driver does not currently support importing Native Tags with a program designation.

Structure Tag Addressing

Logix Structure Tags are tags with one or more member tags (which can be atomic or structured).
<structure name> . <atomic-type tag>

This implies that a substructure would be addressed as the following:
<structure name> . <substructure name> .<atomic-type tag>

Arrays of structures would be addressed as the following:
<structure array name> [dim1, dim2, dim3] . <atomic-type tag>

This implies that an array of substructures would be addressed as the following:
<structure name> . <substructure array name> [dim1, dim2, dim3] . <atomic-type tag>

Note: The examples above display a few of the addressing possibilities that involve structures, and are only provided as an introduction to structure addressing. *For more information, refer to Rockwell/Allen-Bradley documentation.*

Advanced Addressing

Users have several options for symbolic addressing that can be included in the Symbolic Tag address. The following restrictions have been placed on the data type for the bit and array addressing syntaxes:

- For bit syntaxes, the index cannot exceed the bit size for the data type. For example, "MyDint @ Dint" is imported as a Native Tag. The bit index cannot exceed 31 (because DINTs are 32-bit signed values).
- For array syntaxes, the offset and number of elements in the array cannot exceed the number of elements in the associated Native Tag. For example, "MyDintArray @ DINT[10]" is imported as a Native Tag. A Static Tag with addresses "MYDINTARRAY[0] {5}" and "MYDINTARRAY[4] {5}" are valid because the arrays include the first and last 5 elements of the Native Tag respectively. A Static Tag with address "MYDINTARRAY[5]{10}" is invalid because the tag is asking for 10 DINTs beginning at offset 5 and the Native Tag array is not that large.

For more information on advanced topics, refer to the table below.

Element	Syntax	Example	Notes
Standard	<tag name>	tag_1	N/A.
Array w/o Offset	<array tag name> {# of columns}	tag_1 {8}	The number of elements to Read/Write equals # of rows times # of columns. If no rows are specified, # of rows defaults to 1. At least 1 element of the array must be addressed. The array begins at a zero offset (array index equals 0 for all dimensions).

Element	Syntax	Example	Notes
	<array tag name> {# of rows} {# of columns}	tag_1 {2} {4}	

Array w/ Offset	<array element tag> [offset] {# of columns}
	
	<array element tag> [offset]{# of rows}{# of columns}
		tag_1 [5] {8}
		
		tag_1[5] {2} {4}
			The number of elements to Read/Write equals # of rows * # of columns. If no rows are specified, # of rows defaults to 1. At least 1 element of the array must be addressed.
			
			The array begins at a zero offset (array index equals 0 for all dimensions).

Bit	<tag name> . bit <tag name> . [bit]	tag_1 . 0 tag_1 . [0]	N/A. N/A.
String	<tag name> / <element count>	tag_1 / 4	The element count must be at least 1. The number of characters to Read/Write depends on the Pack Strings property.*

*When enabled, the number of characters equals the element count times the element size (4 elements of an INT array indicates 8 characters). When disabled, the number of characters equals the element count (4 elements of an INT array indicates 4 characters).

*For more information, refer to **Options**.*

Ordering of Logix Array Data

Since Native Tags support up to three dimensional arrays, the ordering of Logix Array Data is mapped to a 2-dimensional OPC array.

1. Dimensional Arrays - array [dim1]

1 dimensional array data is passed to and from the controller in ascending order.
for (dim1 = 0; dim1 < dim1_max; dim1++)

Example: 3 element array
array [0]
array [1]
array [2]

2. Dimensional Arrays - array [dim1, dim2]

2 dimensional array data is passed to and from the controller in ascending order.

for (dim1 = 0; dim1 < dim1_max; dim1++)
for (dim2 = 0; dim2 < dim2_max; dim2++)

Example: 3X3 element array
array [0, 0]
array [0, 1]
array [0, 2]
array [1, 0]
array [1, 1]
array [1, 2]
array [2, 0]
array [2, 1]
array [2, 2]

3. Dimensional Arrays - array [dim1, dim2, dim3]
3 dimensional array data is passed to and from the controller in ascending order.
for (dim1 = 0; dim1 < dim1_max; dim1++)
for (dim2 = 0; dim2 < dim2_max; dim2++)
for (dim3 = 0; dim3 < dim3_max; dim3++)

Example: 3X3x3 element array
array [0, 0, 0]
array [0, 0, 1]
array [0, 0, 2]
array [0, 1, 0]
array [0, 1, 1]
array [0, 1, 2]
array [0, 2, 0]
array [0, 2, 1]
array [0, 2, 2]
array [1, 0, 0]
array [1, 0, 1]
array [1, 0, 2]
array [1, 1, 0]
array [1, 1, 1]
array [1, 1, 2]
array [1, 2, 0]
array [1, 2, 1]
array [1, 2, 2]
array [2, 0, 0]
array [2, 0, 1]
array [2, 0, 2]
array [2, 1, 0]
array [2, 1, 1]
array [2, 1, 2]
array [2, 2, 0]
array [2, 2, 1]
array [2, 2, 2]

Error Codes

The Allen-Bradley ControlLogix Unsolicited Driver may return the following error codes.

For more information on a specific type of error code, select a link from the list below.

EtherNet/IP Encapsulation Error Codes
CIP Error Codes
0x01 Extended Error Codes
0xFF Extended Error Codes

EtherNet/IP Encapsulation Error Codes

The Allen-Bradley ControlLogix Unsolicited Driver may return the following error codes.

Note: The error codes are in hexadecimal.

Error	Description
0001	Command not handled.
0002	Memory not available for command.
0003	Poorly formed or incomplete data.
0064	Invalid Session ID.
0065	Invalid length in header.
0069	Requested protocol version not supported.

CIP Error Codes

The error codes are in hexadecimal.

Error	Description
01	Connection failure*
02	Insufficient resources
03	Parameter value invalid
04	IOI could not be deciphered or tag does not exist
05	Unknown destination
06	Data requested would not fit in response packet
08	Unsupported service
0F	Permission denied
13	Insufficient command data / parameter specified to execute service
26	The number of IOI words specified does not match IOI word count
FF	General Error**

****See Also:*** *0x01 Extended Error Codes*
*****See Also:*** *0xFF Extended Error Codes*

0x01 Extended Error Codes

The Allen-Bradley ControlLogix Unsolicited Driver may return the following extended errors for CIP error 0x01.

Note: The error codes are in hexadecimal.

Error	Description
0x0205	Unconnected Send parameter error.
0x0312	Link address is not available.
0x0318	Link address to self is invalid.

0xFF Extended Error Codes

The Allen-Bradley ControlLogix Unsolicited Driver may return the following extended errors for CIP error 0xFF.

Note: The error codes are in hexadecimal.

Error	Description
2104	Address is out of range.
2105	Attempt to access beyond the end of the data object.
2107	The data type is invalid or not supported.

Event Log Messages

The following information concerns messages posted to the Event Log pane in the main user interface. Consult the server help on filtering and sorting the Event Log detail view. Server help contains many common messages, so should also be searched. Generally, the type of message (informational, warning) and troubleshooting information is provided whenever possible.

Error importing Native Tag database. Unable to open file. | OS error = '<error>'.

Error Type:
Error

Error importing Native Tag database. Unable to open file, general read failure.

Error Type:
Error

Possible Cause:
The file is missing, corrupt, or formatted incorrectly.

Possible Solution:
Locate the tag database file and verify header row, format, and location before trying again.

Error importing Native Tag database. File encoding not supported.

Error Type:
Error

Possible Cause:
The CSV file being imported uses unsupported file encoding.

Possible Solution:
Update the CSV file to use ANSI or UTF-8 encoding method.

Error importing Native Tag database.

Error Type:
Error

Possible Cause:
An unexpected error has been encountered.

Possible Solution:
Verify that the CSV file being imported is properly formatted.

Error importing Native Tag database, unrecognized field name. | Unrecognized field name = '<field name>'.

Error Type:
Error

Possible Cause:
The CSV file defines a field that is not supported by the native tag database import.

Possible Solution:
Verify that there are no unintended fields in the CSV file.

Note:
Supported field names include "Logix Address," "Logix DataType," "External Access," and "Description".

Error importing Native Tag database, duplicate field name. | Duplicate field name = '<field>'.

Error Type:
Error

Possible Cause:
The CSV file being imported contains multiple definitions of the same field name.

Possible Solution:
Remove or correct duplicated fields in the CSV file. Supported field names include Logix Address, Logix Data Type, External Access, and Description.

Error importing Native Tag database. Missing tag field identification record.

Error Type:
Error

Possible Cause:
The CSV file being imported does not contain a header.

Possible Solution:
Verify that there are no missing fields in the CSV file and confirm or correct the header row.

Note:
Supported field names include Logix Address, Logix Data Type, External Access, and Description.

Error importing Native Tag database. Incomplete tag field identification record.

Error Type:
Error

Possible Cause:

The CSV file being imported does not contain a complete and valid header.

Possible Solution:
Verify that there are no missing fields in the CSV file and confirm or correct the header row.

● Note:
Supported field names include Logix Address, Logix Data Type, External Access, and Description.

Failed to start unsolicited Logix server.

Error Type:
Error

Possible Cause:
The driver was unable to bind and listen on the specified IP/port.

Possible Solution:
Verify that the specified port (TCP or UDP) is not being used by another application and release any conflicts.

Invalid tag address. Native Tag not imported. | Invalid address = '<address>'.

Error Type:
Warning

Two channels are configured to use the same network adapter IP and TCP port. Each channel must be bound to a unique local IP and port. | First channel = '<channel>', second channel = '<channel>'.

Error Type:
Warning

Two devices are configured to use the same path from the EtherNet/IP module. Each channel must have a unique path from the EtherNet/IP module. | First device = '<address>', Second device = '<address>'.

Error Type:
Warning

Possible Cause:
The device is configured for a CPU type and slot number already being used.

Possible Solution:

1. Select a different CPU type (Local or Remote).
2. Select a different slot number if the device is configured as a remote controller module.

Invalid tag address. Duplicate tag addresses are not allowed. | Invalid address = '<address>'.

Error Type:
Warning

Possible Cause:
The CSV file being imported contains one or more tags with the same Logix address.

Possible Solution:
In the Native Tag Database CSV file being imported, eliminate Native Tags that contain a duplicate Logix Address.

Memory could not be allocated for tag. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The resources needed to build a tag could not be allocated. The tag was not added to the project.

Possible Solution:
Close any unused applications and/or increase the amount of virtual memory and try again.

Invalid native tag. Individual tag size limited to 128 kB. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The CSV file being imported contains a single tag definition that requires more than 128 kB of memory to represent, such as in a large array. Native tags cannot be defined that require more than 128 kB of memory.

Possible Solution:
Reduce the array size or remove the tag from the CSV file to import.

Error importing native tag. Total database tag data size is limited to 128 kB. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The CSV file being imported contains tag definitions that require more than 128 kB of memory, such as in a large array.

Possible Solution:
If more than 128 kB of memory is required, create a new device and split the tag database across multiple devices.

The TCP/IP port specified is out of range. Using the default port. | Valid range = <number> to <number>, default port = <number>.

Error Type:
Warning

Possible Cause:
A project was loaded that specifies a TCP/IP port of 0, but this option is not supported.

Possible Solution:
Use the default port (44818) or select a port within the valid range (1 to 65535).

Another device within the channel is already registered as the local CPU. | Device = <device>.

Error Type:
Warning

Native Tags imported. | Tag count = <count>, tag database path = '<path>'.

Error Type:
Informational

Tags generated. | Tag count = <count>, Tag hierarchy mode = '<mode>'.

Error Type:
Informational

Failed to perform automatic tag generation due to low resources.

Error Type:
Informational

Index