Allen-Bradley Micro800 Serial Driver Help

Table of Contents

Allen-Bradley Micro800 Serial Driver Help

Help version 1.019

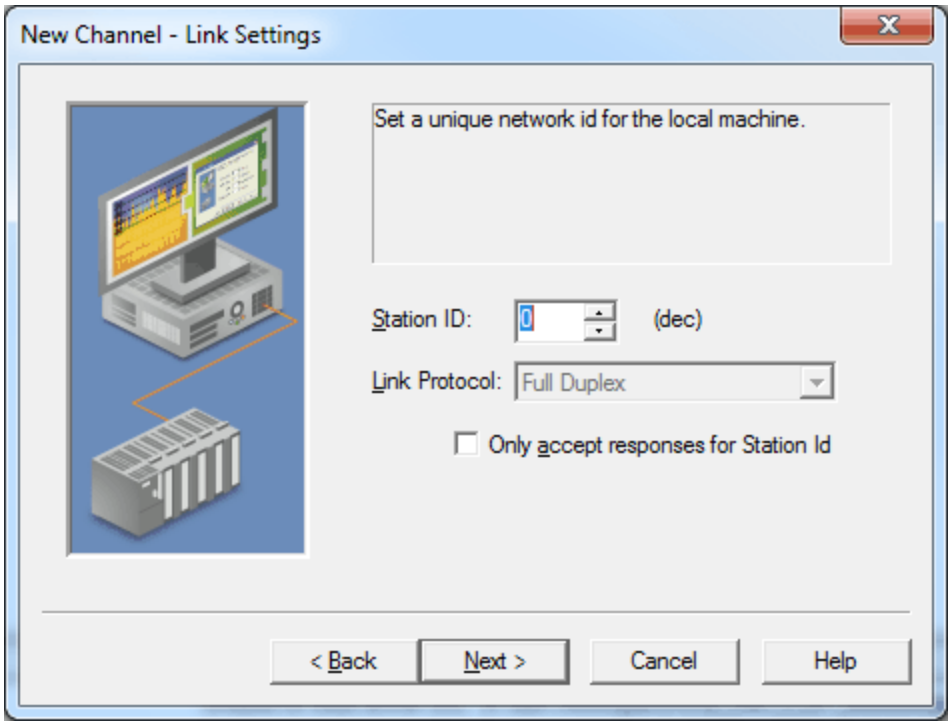
CONTENTS

Overview

The Allen-Bradley Micro800 Serial Driver provides an easy and reliable way to connect Allen-Bradley Micro800 controllers over Serial to OPC client applications, including HMI, SCADA, Historian, MES, ERP, and countless custom applications.

Channel Setup

Link Settings



Descriptions of the parameters are as follows:

- **Station ID:** This parameter specifies a unique Network ID for the local machine. It should be set based on the device with which it is communicating (excluding radio modems). The format is in decimal. The default setting is 0.
- **Link Protocol:** This parameter specifies the link protocol that will be used for communications. The default setting is Full Duplex.
- **Only Accept Responses for Station ID:** When checked, this parameter limits the acceptance of responses to those that are destined for the station as indicated by the Station ID. This parameter only applies to Full Duplex. The default setting is unchecked.

  Note: If the destination device is on a DH+ or DH-485 network, communication must go through a Serial-to-DH+/DH-485 converter (that is, a KF2/KF3 module). In this case, the device being communicated with is the converter and not the destination device itself. For this configuration, the Station ID should be set to the converter's node address. The range for DH-4850 is 1 to 63. If the destination device is not on a DH+ or DH-485 network, the device being communicated with is a Micro800. The Station ID for this configuration can be set to an arbitrary unique address. The range is 0 to 255.

Device Setup

Supported Devices

Micro830
Micro850

Note: The connection is made over an embedded Serial port or plug-in Serial module.

Communication Protocol

Rockwell Automation Fragmentation Protocol (CIP over DF1).

Maximum Number of Channels and Devices

The maximum number of channels supported is 256. The maximum number of devices supported is 1024.

Device ID

The Device ID is the PLC's network address. For PLCs on a DH-485 or DH+ network, the range is 1 to 63. Otherwise, the range is 0 to 255.

For Full Duplex, use the default address (1).
For Half-Duplex, match the slave address.
For Radio Modem, match the slave/peer address.

Ethernet Encapsulation

This driver supports Ethernet Encapsulation, which allows the driver to communicate with serial devices attached to an Ethernet network using a terminal server. Ethernet Encapsulation mode may be invoked through the COM ID dialog in Channel Properties. For more information, refer to the server's help documentation.

DH-485 and DH+ Support

An Allen Bradley KF3 or compatible device is needed to connect the driver to the DH-485 network. There are four options for communicating to a device on DH+ using the Allen-Bradley Micro800 Serial Driver.

- Allen Bradley KF2 or compatible device.
- 1784-U2DHP USB converter. This converter appears as a new serial port to the system.
- DataLink DL Interface Cards (PCI/ISA/PC104). These cards add virtual serial ports for seamless configuration.
- DataLink DL4500 Ethernet-to-DH+ Converter. Configure the device for Ethernet Encapsulation. NIC is required.

Cable Diagram

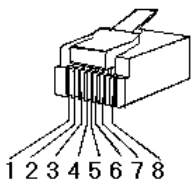
Patch Cable (Straight Through)

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

Crossover Cable

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

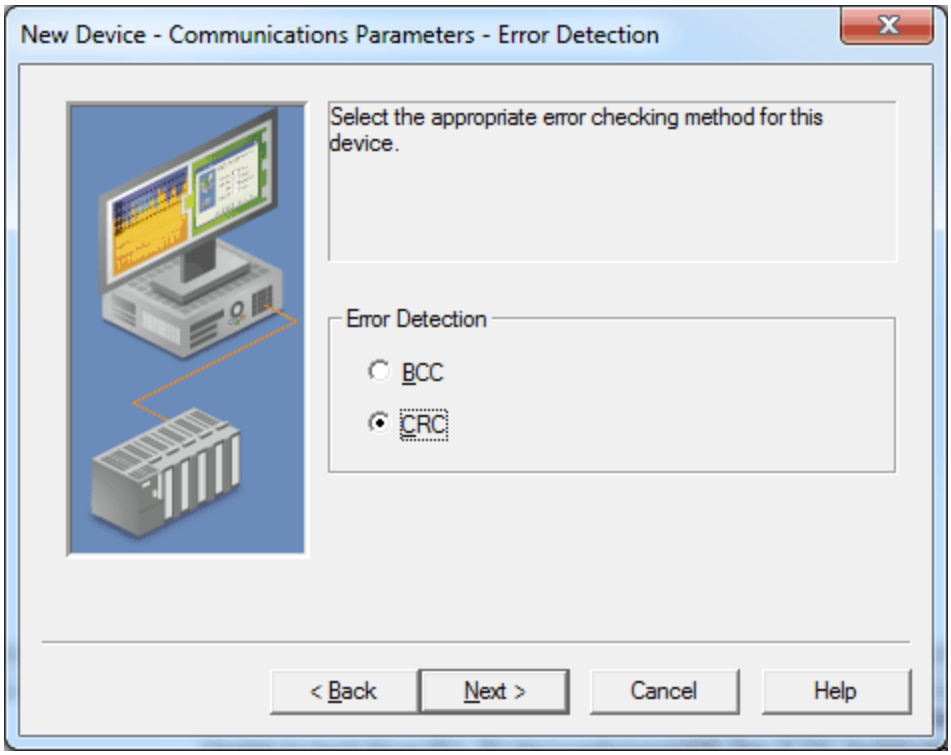
10 BaseT



8-pin RJ45

Communications Parameters

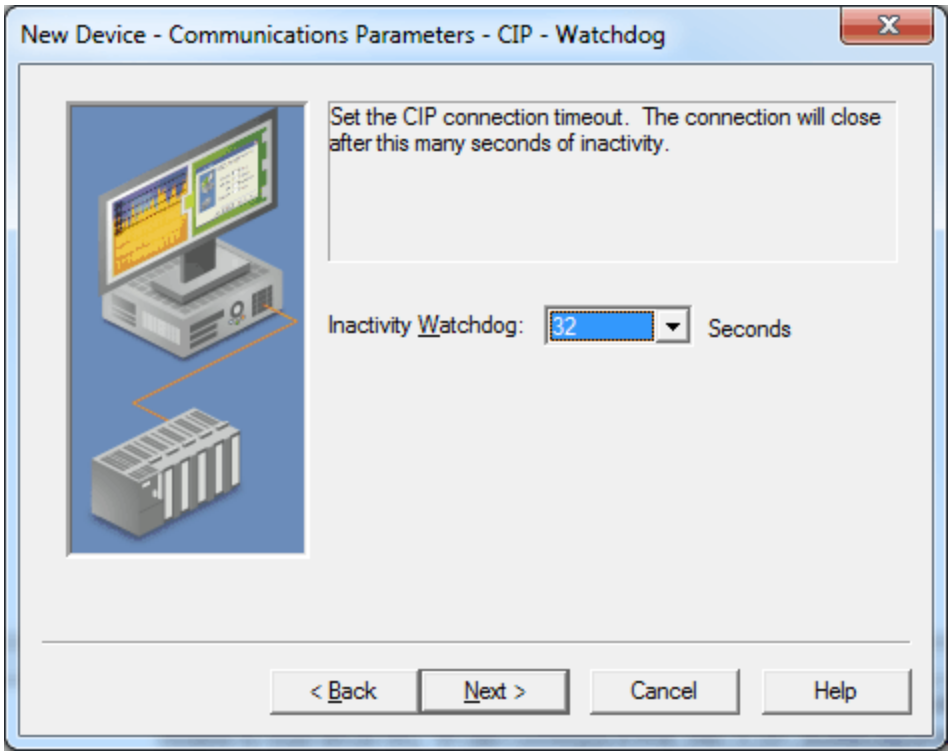
Error Detection



Description of the parameter is as follows:

- **Error Detection:** This parameter specifies the method of error detection. Options include BCC (Block Check Character) and CRC (Cyclic Redundancy Check). Users must choose the checksum method expected by the device; otherwise, the device will fail requests. The default setting is CRC.

CIP - Watchdog

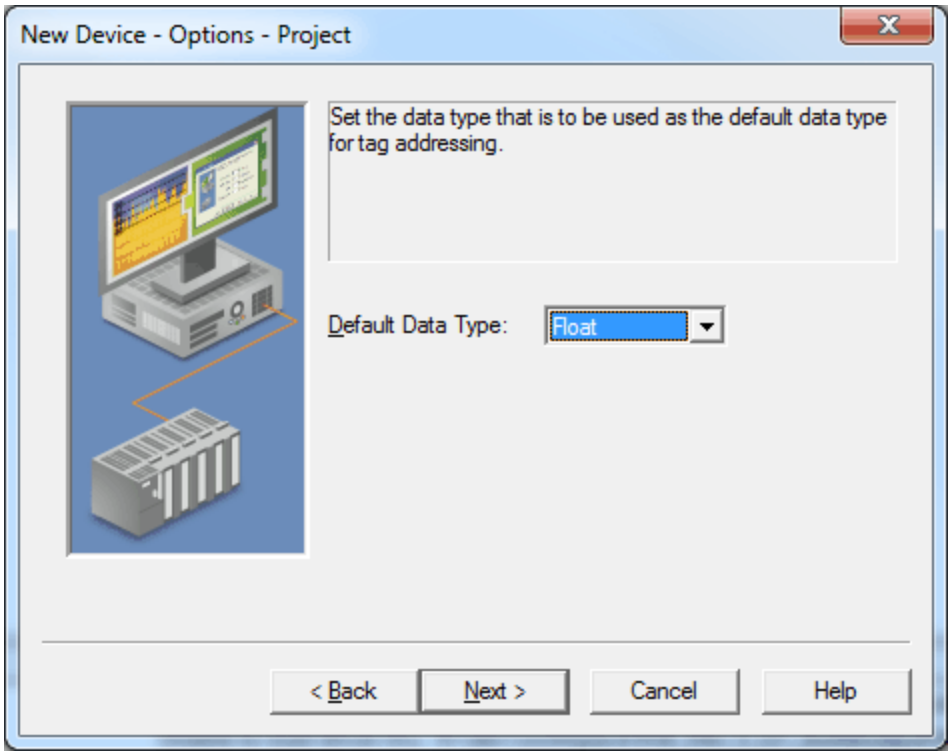


Description of the parameter is as follows:

- **Inactivity Watchdog:** This parameter specifies the amount of time a connection can remain idle (without Read/Write transactions) before being closed by the controller. In general, the larger the watchdog value, the more time it will take for connection resources to be released by the controller (and vice versa). The default setting is 32 seconds.

Options

Project



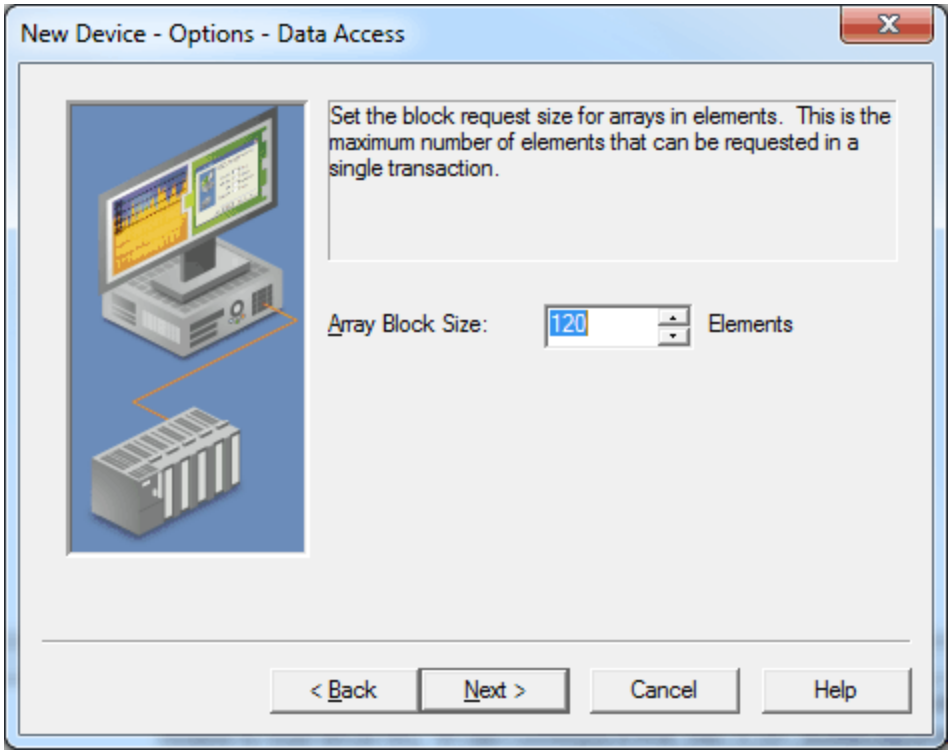
Description of the parameter is as follows:

- **Default Data Type:** This parameter specifies the data type that will be assigned to a Client/Server Tag when the default type is selected during tag addition/modification/import. The default setting is Float.

Default Data Type Conditions
Client/Server Tags are assigned the default data type when any of the following conditions occur:

1. A Dynamic Tag is created in the client with Native as its assigned data type.
2. A Static Tag is created in the server with Default as its assigned data type.

Data Access



Description of the parameter is as follows:

- **Array Block Size:** This parameter specifies the maximum number of atomic array elements that will be read in a single transaction. The value is adjustable and ranges from 30 to 3840 elements. The default setting is 120 elements.
  Note: For Boolean arrays, a single element is considered a 32 element bit array. Thus, setting the block size to 30 elements translates to 960 bit elements, whereas 3840 elements translate to 122880 bit elements.

Performance Optimizations

Several guidelines may be applied to the Allen-Bradley Micro800 Serial Driver to gain maximum performance. For more information on optimization at the communication and application levels, select a link from the list below.

Optimizing Communications
Optimizing Application

Optimizing Communications

As with any programmable controller, there are a variety of ways to enhance the overall performance and system communications.

Keep Native Tag Names Short

Native Tags read from and write to the device by specifying its symbolic name in the communications request. As such, the longer the tag name is, the larger the request will be.

Array Elements Blocked

To optimize the reading of atomic array elements, read a block of the array in a single request instead of individually. The more elements read in a block, the greater the performance. Since transaction overhead and processing consumes the most time, do as few transactions as possible while scanning as many desired tags as possible. This is the essence of array element blocking.

Block sizes are specified as an element count. A block size of 120 elements means that a maximum of 120 array elements will be read in one request. The maximum block size is 3840 elements. Boolean arrays are treated differently: in protocol, a Boolean array is a 32 bit array. Thus, requesting element 0 is requesting bits 0 through 31. To maintain consistency in discussion, a Boolean array element will be considered a single bit. In summary, the maximum number of array elements (based on block size of 3840) that can be requested is as follows: 122880 BOOL, 3840 SINT, 3840 INT, 3840 DINT, 3840 LINT, and 3840 REAL.

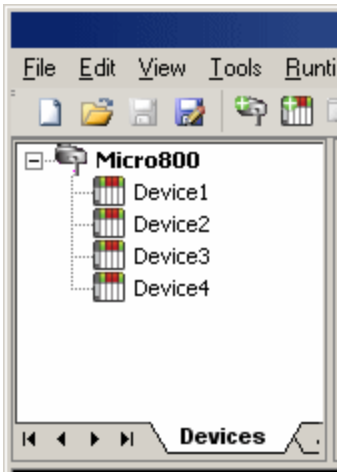
The block size is adjustable, and should be chosen based on the project at hand. For example, if array elements 0-26 and element 3839 are tags to be read, then using a block size of 3840 is not only overkill, but detrimental to the driver's performance. This is because all elements between 0 and 3839 will be read on each request, even though only 28 of those elements are of importance. In this case, a block size of 30 is more appropriate. Elements 0-26 would be serviced in one request and element 3839 would be serviced on the next.

See Also: **Options**

Optimizing Applications

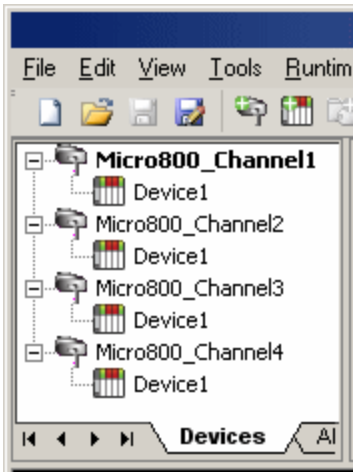
The Allen-Bradley Micro800 Serial Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the Allen-Bradley Micro800 Serial Driver is fast, there are a couple of guidelines that can be used in order to gain maximum performance.

The server refers to communications protocols like Allen-Bradley Micro800 Serial as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must then be defined under that channel. Each of these devices represents a single Micro800 CPU from which data will be collected. While this approach to defining the application will provide a high level of performance, it won't take full advantage of the Allen-Bradley Micro800 Serial Driver or the network. An example of how the application may appear when configured using a single channel is shown below.



Each device appears under a single channel, called "Micro800". In this configuration, the driver must move from one device to the next as quickly as possible in order to gather information at an effective rate. As more devices are added or more information is requested from a single device, the overall update rate begins to suffer.

If the Allen-Bradley Micro800 Serial Driver could only define one single channel, then the example shown above would be the only option available; however, the Allen-Bradley Micro800 Serial Driver can define up to 256 channels. Using multiple channels distributes the data collection workload by simultaneously issuing multiple requests to the network. An example of how the same application may appear when configured using multiple channels to improve performance is shown below.



Each device has now been defined under its own channel. In this new configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has 256 or fewer devices, it can be optimized exactly how it is shown here.

The performance will improve even if the application has more than 256 devices. While 256 or fewer devices may be ideal, the application will still benefit from additional channels. Although spreading the device load across all channels will cause the server to move from device to device again, it can now do so with far less devices to process on a single channel.

Data Types Description

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8-bit value
Char	Signed 8-bit value
Word	Unsigned 16-bit value
Short	Signed 16-bit value
DWord	Unsigned 32-bit value
Long	Signed 32-bit value
BCD	Two byte packed BCD, four decimal digits
LBCD	Four byte packed BCD, eight decimal digits
Float	32-bit IEEE Floating point
Double	64-bit IEEE Floating point
Date	64-bit Date/Time
String	Null terminated character array

Address Descriptions

Micro800 uses a tag or symbol-based addressing structure referred to as Native Tags. These tags differ from conventional PLC data items in that the tag name itself is the address, not a file or register number.

The Allen-Bradley Micro800 Serial Driver allows users to access the controller's atomic data types: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, and SHORT_STRING. Although some of the pre-defined types are structures, they are ultimately based on these atomic data types. Thus, 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 assigned to the tag (for example, 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-defined and module-defined types, and is not found in any of the pre-defined types.

Atomic Data Type	Description	Client Type	Range
BOOL	Single bit value	VT_BOOL	0, 1
SINT	Signed 8 bit value	VT_U1	-128 to 127
USINT	Unsigned 8 bit value	VT_UI1	0 to 255
BYTE	Bit string (8 bits)	VT_UI1	0 to 255
INT	Signed 16 bit value	VT_I2	-32,768 to 32,767
UINT	Unsigned 16 bit value	VT_UI2	0 to 65535
WORD	Bit string (16 bits)	VT_UI2	0 to 65535
DINT	Signed 32 bit value	VT_I4	-2,147,483,648 to 2,147,483,647
UDINT	Unsigned 32 bit value	VR_UI4	0 to 4294967296
DWORD	Bit string (32 bits)	VR_UI4	0 to 4294967296
LINT	Signed 64 bit value	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
ULINT	Unsigned 64 bit value	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
LWORD	Bit string (64 bits)	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
REAL	32 bit IEEE Floating point	VT_R4	1.1755 E-38 to 3.403E38, 0, -3.403E-38 to -1.1755
LREAL	64 bit IEEE Floating point	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
SHORT_STRING	Character string. The maximum number of characters allowed is 80.	VT_BSTR	

See Also: Advanced Use Cases

Client/Server Tag Address Rules

Native Tag names correspond to Client/Server Tag addresses. Both Native Tag names (entered via the Connected Components Workbench) and Client/Server Tag addresses follow the IEC 1131-3 identifier rules. Descriptions of the rules are as follows:

- Must begin with an alphabetic character or an underscore
- Can only contain alphanumeric characters and underscores
- Can have as many as 40 characters
- Cannot have consecutive underscores
- Characters are not case sensitive

Note: For optimum performance, keep Native Tag names to a minimum in size. The smaller the name, the more requests that can fit in a single transaction.

Client/Server Tag Name Rules
Tag name assignment in the server differs from address assignment in that names cannot begin with an underscore.

See Also: Performance Optimizations

Address Formats

A Native Tag may be addressed statically in the server or dynamically from a client in several ways. The tag's format will depend on its type and intended usage. For example, the bit format would be used when accessing a bit within a SINT-type tag. For information on address format and syntax, refer to the table below.

Note: Every format is native to Connected Components Workbench (CCW) except for the Array formats. Therefore, when referencing an atomic data type, a CCW tag name could be copied and pasted into the server's tag address field and be valid.

See Also: Advanced Use Cases

Format	Syntax
Array Element	<Native Tag name> [dim 1, dim2, dim 3]
Array w/ Offset*	<Native Tag name> {# columns} <Native Tag name> {# rows}{# columns}
Array w/o Offset*	<Native Tag name> {# columns} <Native Tag name> {# rows}{# columns}
Bit	<Native Tag name>.bit <Native Tag name>.[bit]
Standard	<Native Tag name>
String	<Native Tag name>

*Since these formats may request more than one element, the order in which array data is passed depends on the dimension of the array tag. For example, if rows times cols = 4 and the Native 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 exact order. The results would be different if the Native Tag were a 2X10 element array. For more information, refer to Ordering of Array Data.

Expanded Address Formats

Array Element
At least 1 dimension (but no more then 3) must be specified.

Syntax	Example	Notes
<Native Tag Name> [dim1]	tag_1 [5]	N/A
<Native Tag name> [dim 1, dim2]	tag_1 [2, 3]	N/A
<Native Tag name> [dim 1, dim2, dim 3]	tag_1 [2, 58, 547]	N/A

Array With Offset
Since this class may request more than one element, the order in which array data is passed depends on the dimension of the Array Tag.

Syntax	Example	Notes
<Native	tag_1 [5]	The number of elements to Read/Write equals the number of rows multiplied by the

Tag name> [offset] {# of columns}	{8}	number of columns. If no rows are specified, the number of rows will default 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).
<Native Tag name> [offset] {# of rows}{# of columns}	tag_1 [5] {2}{4}	

Note: If rows*cols = 4 and the Native 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 exact order. The results would be different if the Native Tag were a 2X10 element array.

Array Without Offset
Since this class may request more than one element, the order in which array data is passed depends on the dimension of the Array Tag.

Syntax	Example	Notes
<Native Tag name> {# of columns}	tag_1 {8}	The number of elements to Read/Write equals the number of rows multiplied by the number of columns. If no rows are specified, the number of rows will default 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).
<Native Tag name> {# of rows}{# of columns}	tag_1 {2}{4}	

Note: For example, if rows*cols = 4 and the Native 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 exact order. The results would be different if the Native Tag were a 2X10 element array.

Bit

Syntax	Example	Notes
<Native Tag name> . bit	tag_1 . 0	N/A
<Native Tag name> . [bit]	tag_1 . [0]	N/A

Standard

Syntax	Example	Notes
<Native Tag name>	tag_1	N/A

String

Syntax	Example	Notes
<Native Tag name>	tag_1	The number of characters to Read/Write equals the string length and must be at least 1.

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

Tag Scope

The scope of variables can be local to a program or global to a controller.

- Local variables are assigned to a specific program in the project; they are available only to that program.
- Global variables belong to the controller in the project; they are available to any program in the project.

Local Variables

Local variables (program-scoped tags) cannot be accessed directly through the communications port of the controller, so are not directly supported within the driver. If access is required, cut and paste the tags from the Local variable table to the Global variable table.

Global Variables

Global Variables (controller-scoped tags) are Native 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 both its Native Data Type and the address format being used.

User-Defined Data Types

Users may create unique data types, e.g. STRING with 12 characters rather than 80. These user-defined data types may be used as local or global variables.

Structured Variables

There are no structured variables in Micro800 controllers. Users may build unique Data Types, but each member must have a unique name.

Addressing Atomic Data Types

The table below contains suggested usage and addressing possibilities for each Native Data Type given the available address formats. For each data type's advanced addressing possibilities, click **Advanced**.

Note: Empty cells do not necessarily indicate a lack of support.

BOOL

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Boolean	Boolean (BOOL 1 dimensional array)	Boolean Array (BOOL 1 dimensional array)		
Example	BOOLTAG	BOOLARR[0]	BOOLARR[0]{32}		

SINT, USINT, and BYTE

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Byte, Char	Byte, Char	Byte Array, Char Array (SINT 1/2/3 dimensional array)	Boolean (Bit w/i SINT)	
Example	SINTTAG	SINTARR[0]	SINTARR[0]{4}	SINTTAG.0	j

INT, UINT, and WORD

Tag	Standard	Array Element	Array w/o Offset	Bit	String
Data Type Advanced	Word, Short	Word, Short	Word Array, Short Array (INT 1/2/3 dimensional array)	Boolean (Bit w/i INT)	
Example	INTTAG	INTARR[0]	INTARR[0]{4}	INTTAG.0	

DINT, UDINT, and DWORD

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	DWord, Long	DWord, Long	DWord Array, Long Array	Boolean (Bit w/i DINT)	Advanced
Example	DINTTAG	DINTARR[0]	DINTARR[0]{4}	DINTTAG.0	

LINT, ULINT, and LWORD

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Double, Date	Double, Date	Double Array		
Example	LINTTAG	LINTARR[0]	LINTARR[0]{4}		

REAL

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Float	Float	Float Array		
Example	REALTAG	REALARR[0]	REALARR[0]{4}		

LREAL

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Double	Double	Double Array		
Example	LREALTAG	LREALARR[0]	LREALARR[0]{4}		

SHORT_STRING

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	String	String			
Example	STRINGTAG	STRINGARR[0]			

See Also: Address Formats

Addressing Structured Data Types

Structures cannot be referenced at the structure level: only the atomic structure members can be addressed. For more information, refer to the examples below.

Native Tag
MyTimer @ TIMER

Valid Client/Server Tag
Address = MyTimer.ACC
Data type = DWord

Invalid Client/Server Tag
Address = MyTimer
Data type = ??

Addressing STRING Data Type

STRING is a pre-defined Native Data Type whose structure contains two members: DATA and LEN. DATA is an array of SINTs and stores the characters of the string. LEN is a DINT and represents the number of characters in DATA to display to a client.

Note: Because LEN and Data are atomic members, they must be referenced independently from the client/server.

Description	Syntax	Example
STRING Value	DATA/<Maximum STRING Length>	MYSTRING.DATA/82
Actual STRING Length	LEN	MYSTRING.LEN

Reads

The string read from DATA will be terminated by the following:

a. The first null terminator encountered.

b. The value in LEN if a) doesn't occur first.

c. The <Maximum STRING Length> if either a) or b) doesn't occur first.

Example
MYSTRING.DATA contains "Hello World" in the PLC, but LEN is manually set to 5. A read of MYSTRING.DATA/82 will display "Hello". If LEN is set to 20, MYSTRING.DATA/82 will display "Hello World".

Writes

When a STRING value is written to DATA, the driver will also write to LEN with the length of DATA written. If the write to LEN fails for any reason, the write operation to DATA will be considered failed as well (despite the fact that the DATA write to the controller succeeded).

Note: This behavior was designed specifically for Native Tags of type STRING (or a custom derivative of STRING). The following precautions apply to users who wish to implement their own string in UDTs.

- If a UDT exists that has a DATA member referenced as a string and a LEN member referenced as a DINT, the write to LEN will succeed regardless of the intentions of LEN for the given UDT. Care must be taken when designing UDTs to avoid this possibility if LEN is not intended to be the length of DATA.
- If a UDT exists that has a DATA member referenced as a string but does not have a LEN member, the write to LEN will fail silently without consequence to DATA.

Example

MYSTRING.DATA/82 holds the value "Hello World." MYSTRING.LEN holds 11. If the value "Alarm Triggered" is written to MYSTRING.DATA/82, 15 will be written to MYSTRING.LEN. If the write to MYSTRING.LEN fails, MYSTRING.LEN will hold its previous value of 11 while MYSTRING.DATA/82 displays the first 11 characters ("Alarm Trigg"). If the write to MYSTRING.DATA/82 fails, neither tag is affected.

Ordering of Array Data

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]

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]

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]

Advanced Use Cases

For more information on the advanced use cases for a specific atomic data type, select a link from the list below.

BOOL
SINT, USINT, and BYTE
INT, UINT, and WORD
DINT, UDINT, and DWORD
LINT, ULINT, and LWORD
REAL
LREAL
SHORT_STRING

BOOL

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Boolean	The Native Tag must be a 1 dimensional array.
Array w/ Offset	Boolean Array	1. The Native Tag must be a 1 dimensional array. 2. The offset must lay on a 32-bit boundary. 3. The number of elements must be a factor of 32.
Array w/o Offset	Boolean Array	1. The Native Tag must be a 1 dimensional array. 2. The number of elements must be a factor of 32.
Bit	Boolean	1. The Native Tag must be a 1 dimensional array. 2. The range is limited from 0 to 31.
Standard	Boolean Byte, Char Word, Short, BCD DWord, Long, LBCD Float*	None.
String	Not supported.	

*The Float value will equal the face value of the Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted in yellow signify common use cases.

BOOL Atomic Tag - booltag = True

Server Tag Address	Format	Data Type	Notes
booltag	Standard	Boolean	Value = True

booltag	Standard	Byte	Value = 1
booltag	Standard	Word	Value = 1
booltag	Standard	DWord	Value = 1
booltag	Standard	Float	Value = 1.0
booltag [3]	Array Element	Boolean	Invalid: Tag is not an array.
booltag [3]	Array Element	Word	Invalid: Tag is not an array.
booltag {1}	Array w/o Offset	Word	Invalid: Not supported.
booltag {1}	Array w/o Offset	Boolean	Invalid: Not supported.
booltag [3] {32}	Array w/ Offset	Boolean	Invalid: Tag is not an array.
booltag . 3	Bit	Boolean	Invalid: Tag is not an array.
booltag / 1	String	String	Invalid: Not supported.
booltag / 4	String	String	Invalid: Not supported.

BOOL Array Tag - bitarraytag = [0,1,0,1]

Server Tag Address	Format	Data Type	Notes
bitarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
bitarraytag	Standard	Byte	Invalid: Tag cannot be an array.
bitarraytag	Standard	Word	Invalid: Tag cannot be an array.
bitarraytag	Standard	DWord	Invalid: Tag cannot be an array.
bitarraytag	Standard	Float	Invalid: Tag cannot be an array.
bitarraytag [3]	Array Element	Boolean	Value = True
bitarraytag [3]	Array Element	Word	Invalid: Bad data type.
bitarraytag {3}	Array w/o Offset	Word	Invalid: Tag cannot be an array.
bitarraytag {1}	Array w/o Offset	Word	Invalid: Tag cannot be an array.
bitarraytag {1}	Array w/o Offset	Boolean	Invalid: Array size must be a factor of 32.
bitarraytag {32}	Array w/o Offset	Boolean	Value = [0,1,0,1,...]
bitarraytag [3] {32}	Array w/ Offset	Boolean	Offset must begin on 32-bit boundary.
bitarraytag[0]{32}	Array w/ Offset	Boolean	Value = [0,1,0,1,...]
bitarraytag[32]{64}	Array w/ Offset	Boolean	Syntax valid. Element is out of range.
bitarraytag . 3	Bit	Boolean	Value = True
bitarraytag / 1	String	String	Invalid: Not supported.
bitarraytag / 4	String	String	Invalid: Not supported.

SINT, USINT, and BYTE

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Byte, Char Word, Short, BCD DWord, Long, LBCD Float***	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array Word Array, Short Array, BCD Array** DWord Array, Long Array, LBCD Array** Float Array**,***	The Native Tag must be an array.
Array w/o Offset	Boolean Array Byte Array, Char Array Word Array, Short Array, BCD Array**	1. Use this case to have the bits within an SINT in array form. This is not an array of SINTs in Boolean notation. 2. Applies to bit-within-SINT only. Example: tag_1.0{8}. 3. The .bit plus the array size cannot exceed 8 bits. Example: tag_1.1{8} exceeds an SINT, tag_1.0{8} does not. If accessing more than a single element, the Native Tag must be an array.

	DWord Array, Long Array, LBCD Array** Float Array**,***	
Bit	Boolean	1. The range is limited from 0 to 7. 2. If the Native Tag is an array, the bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0.
Standard	Boolean* Byte, Char Word, Short, BCD DWord, Long, LBCD Float***	None.
String	String	1. If accessing a single element, the Native Tag does not need to be an array. **Note:** The value of the string will be the ASCII equivalent of the SINT value. Example: SINT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string will be the null-terminated ASCII equivalent of all the SINTs in the string. 1 character in string = 1 SINT.

*Non-zero values will be clamped to True.
**Each element of the array corresponds to an element in the SINT array. Arrays are not packed.
***Float value will equal face value of Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted in yellow signify common use cases for SINT, USINT, and BYTE.

SINT, USINT, and BYTE Atomic Tag - sinttag = 122 (decimal)

Server Tag Address	Format	Data Type	Notes
sinttag	Standard	Boolean	Value = True
sinttag	Standard	Byte	Value = 122
sinttag	Standard	Word	Value = 122
sinttag	Standard	DWord	Value = 122
sinttag	Standard	Float	Value = 122.0
sinttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
sinttag [3]	Array Element	Byte	Invalid: Tag is not an array.
sinttag {3}	Array w/o Offset	Byte	Invalid: Tag is not an array.
sinttag {1}	Array w/o Offset	Byte	Value = [122]
sinttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
sinttag [3] {1}	Array w/ Offset	Byte	Invalid: Tag is not an array.
sinttag . 3	Bit	Boolean	Value = True
sinttag . 0 {8}	Array w/o Offset	Boolean	Value = [0,1,0,1,1,1,1,0] Bit value of 122
sinttag / 1	String	String	Invalid: Syntax / data type not supported.
sinttag / 4	String	String	Invalid: Syntax / data type not supported.

SINT, USINT, and BYTE Array Tag - sintarraytag [4,4] = [[83,73,78,84],[5,6,7,8],[9,10,11,12],[13,14,15,16]]

Server Tag Address	Format	Data Type	Notes
sintarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
sintarraytag	Standard	Byte	Invalid: Tag cannot be an array.
sintarraytag	Standard	Word	Invalid: Tag cannot be an array.

sintarraytag	Standard	DWord	Invalid: Tag cannot be an array.
sintarraytag	Standard	Float	Invalid: Tag cannot be an array.
sintarraytag [3]	Array Element	Byte	Invalid: Server tag missing dimension 2 address.
sintarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
sintarraytag [1,3]	Array Element	Byte	Value = 8
sintarraytag {10}	Array w/o Offset	Byte	Value = [83,73,78,84,5,6,7,8,9,10]
sintarraytag {2} {5}	Array w/o Offset	Word	Value = [83,73,78,84,5] [6,7,8,9,10]
sintarraytag {1}	Array w/o Offset	Byte	Value = 83
sintarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
sintarraytag [1,3] {4}	Array w/ Offset	Byte	Value = [8,9,10,11]
sintarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
sintarraytag [1,3] . 3	Bit	Boolean	Value = 1
sintarraytag [1,3] . 0 {8}	Array w/o Offset	Boolean	Value = [0,0,0,1,0,0,0,0]
sintarraytag / 1	String	String	Invalid: Syntax / data type not supported.
sintarraytag / 4	String	String	Invalid: Syntax / data type not supported.

INT, UINT, and WORD

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Byte, Char** Word, Short, BCD DWord, Long, LBCD Float****	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array** Word Array, Short Array, BCD Array DWord Array, Long Array, LBCD Array*** Float Array ***,****	The Native Tag must be an array.
Array w/o Offset	Boolean Array Byte Array, Char Array** Word Array, Short Array, BCD Array DWord Array, Long Array, LBCD Array *** Float Array ***,****	1. Use this case to have the bits within an INT in array form. This is not an array of INTs in Boolean notation. 2. Applies to bit-within-INT only. Example: tag_1.0{16}. 3. The .bit plus the array size cannot exceed 16 bits. Example: tag_1.1{16} exceeds an INT, tag_1.0{16} does not. If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	1. The range is limited from 0 to 15. 2. If the Native Tag is an array, the bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0.
Standard	Boolean* Byte, Char** Word, Short, BCD DWord, Long, LBCD Float****	None.
String	String	1. If accessing a single element, the controller tag does not need to be an array.

		Note: The value of the string will be the ASCII equivalent of the INT value (clamped to 255). Example: INT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string will be the null-terminated ASCII equivalent of all the INTs (clamped to 255) in the string. 1 character in string = 1 INT, clamped to 255. **Note:** INT strings are not packed. For greater efficiency, use SINT strings or the STRING structure instead.

*Non-zero values will be clamped to True.
**Values exceeding 255 will be clamped to 255.
*** Each element of the array corresponds to an element in the INT array. Arrays are not packed.
****Float value will equal face value of Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted in yellow signify common use cases for INT, UINT, and WORD.

INT, UINT, and WORD Atomic Tag - inttag = 65534 (decimal)

Server Tag Address	Class	Data Type	Notes
inttag	Standard	Boolean	Value = True
inttag	Standard	Byte	Value = 255
inttag	Standard	Word	Value = 65534
inttag	Standard	DWord	Value = 65534
inttag	Standard	Float	Value = 65534.0
inttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
inttag [3]	Array Element	Word	Invalid: Tag is not an array.
inttag {3}	Array w/o Offset	Word	Invalid: Tag is not an array.
inttag {1}	Array w/o Offset	Word	Value = [65534]
inttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
inttag [3] {1}	Array w/ Offset	Word	Invalid: Tag is not an array.
inttag . 3	Bit	Boolean	Value = True
inttag . 0 {16}	Array w/o Offset	Boolean	Value = [0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] Bit value of 65534
inttag / 1	String	String	Invalid: Syntax / data type not supported.
inttag / 4	String	String	Invalid: Syntax / data type not supported.

INT, UINT, and WORD Array Tag - intarraytag [4,4] = [[73,78,84,255],[256,257,258,259], [9,10,11,12],[13,14,15,16]]

Server Tag Address	Class	Data Type	Notes
intarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
intarraytag	Standard	Byte	Invalid: Tag cannot be an array.
intarraytag	Standard	Word	Invalid: Tag cannot be an array.
intarraytag	Standard	DWord	Invalid: Tag cannot be an array.
intarraytag	Standard	Float	Invalid: Tag cannot be an array.
intarraytag [3]	Array Element	Word	Invalid: Server tag is missing dimension 2 address.
intarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
intarraytag [1,3]	Array Element	Word	Value = 259
intarraytag {10}	Array w/o Offset	Byte	Value = [73,78,84,255,255,255,255,255,9,10]
intarraytag {2} {5}	Array w/o	Word	Value = [73,78,84,255,256] [257,258,259,9,10]

	Offset		
intarraytag {1}	Array w/o Offset	Word	Value = 73
intarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
intarraytag [1,3] {4}	Array w/ Offset	Word	Value = [259,9,10,11]
intarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
intarraytag [1,3] . 3	Bit	Boolean	Value = 0
intarraytag [1,3] . 0 {16}	Array w/o Offset	Boolean	Value = [1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0] Bit value for 259
intarraytag / 1	String	String	Invalid: Syntax / data type not supported.
intarraytag / 3	String	String	Invalid: Syntax / data type not supported.

DINT, UDINT, and DWORD

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Byte, Char** Word, Short, BCD*** DWord, Long, LBCD Float****	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array** Word Array, Short Array, BCD Array*** DWord Array, Long Array, LBCD Array Float Array****	The Native Tag must be an array.
Array w/o Offset	Boolean Array Byte Array, Char Array** Word Array, Short Array, BCD Array*** DWord Array, Long Array, LBCD Array Float Array****	1. Use this case to have the bits within an DINT in array form. This is not an array of DINTs in Boolean notation. 2. Applies to bit-within-DINT only. Example: tag_1.0{32}. 3. The .bit plus the array size cannot exceed 32 bits . Example: tag_1.1{32} exceeds an DINT, tag_1.0{32} does not. If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	1. The range is limited from 0 to 31. 2. If Native Tag is an array, bit class reference must be prefixed by an array element class reference . Example: tag_1 [2,2,3].0.
Standard	Boolean* Byte, Char** Word, Short, BCD*** DWord, Long, LBCD Float****	None.
String	String	1. If accessing a single element, the controller tag does not need to be an array. **Note:** The value of the string will be the ASCII equivalent of the DINT value (clamped to 255). Example: SINT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string will be the null-terminated ASCII equivalent of all the DINTs (clamped

		to 255) in the string. 1 character in string = 1 DINT, clamped to 255. **Note:** DINT strings are not packed. For greater efficiency, use SINT strings or the STRING structure instead.

*Non-zero values will be clamped to True.
**Values exceeding 255 will be clamped to 255.
***Values exceeding 65535 will be clamped to 65535.
****Float value will equal face value of Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted in yellow signify common use cases for DINT, UDINT, and DWORD.

DINT, UDINT, and DWORD Atomic Tag - dinttag = 70000 (decimal)

Server Tag Address	Format	Data Type	Notes
dinttag	Standard	Boolean	Value = True
dinttag	Standard	Byte	Value = 255
dinttag	Standard	Word	Value = 65535
dinttag	Standard	DWord	Value = 70000
dinttag	Standard	Float	Value = 70000.0
dinttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
dinttag [3]	Array Element	DWord	Invalid: Tag is not an array.
dinttag {3}	Array w/o Offset	DWord	Invalid: Tag is not an array.
dinttag {1}	Array w/o Offset	DWord	Value = [70000]
dinttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type
dintag [3] {1}	Array w/ Offset	DWord	Invalid: Tag is not an array.
dinttag . 3	Bit	Boolean	Value = False
dinttag . 0 {32}	Array w/o Offset	Boolean	Value = [0,0,0,0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,...0] Bit value for 70000
dinttag	String	String	Invalid: Syntax / data type not supported.
dinttag	String	String	Invalid: Syntax / data type not supported.

DINT, UDINT, and DWORD Array Tag - dintarraytag [4,4] = [[68,73,78,84],[256,257,258,259], [9,10,11,12],[13,14,15,16]]

Server Tag Address	Format	Data Type	Notes
dintarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
dintarraytag	Standard	Byte	Invalid: Tag cannot be an array.
dintarraytag	Standard	Word	Invalid: Tag cannot be an array.
dintarraytag	Standard	DWord	Invalid: Tag cannot be an array.
dintarraytag	Standard	Float	Invalid: Tag cannot be an array.
dintarraytag [3]	Array Element	DWord	Invalid: Server tag missing dimension 2 address.
dintarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
dintarraytag [1,3]	Array Element	DWord	Value = 259
dintarraytag {10}	Array w/o Offset	Byte	Value = [68,73,78,84,255,255,255,255,9,10]
dintarraytag {2}{5}	Array w/o Offset	DWord	Value = [68,73,78,84,256] [257,258,259,9,10]
dintarraytag {1}	Array w/o Offset	DWord	Value = 68
dintarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.

dintarraytag [1,3]{4}	Array w/ Offset	DWord	Value = [259,9,10,11]
dintarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
dintarraytag [1,3] . 3	Bit	Boolean	Value = 0
dintarraytag [1,3] .0 {32}	Array w/o Offset	Boolean	Value = [1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0] Bit value for 259
dintarraytag	String	String	Invalid: Syntax / data type not supported.
dintarraytag	String	String	Invalid: Syntax / data type not supported.

LINT, ULINT, and LWORD

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Double* Date**	The Native Tag must be an array.
Array w/ Offset	Double Array*	The Native Tag must be an array.
Array w/o Offset	Double Array*	If accessing more than a single element, the controller tag must be an array.
Bit	Not supported.	Not supported.
Standard	Double* Date**	None.
String	Not supported.	Not supported.

*Double value will equal face value of controller tag in Float form (non-IEEE Floating point number).
**Date values are in universal time (UTC), not localized time.

Examples

Examples highlighted in yellow signify common use cases for LINT, ULINT, and LWORD.

LINT, ULINT, and LWORD Atomic Tag - linttag = 2007-01-01T16:46:40.000 (date) == 1.16767E+15 (decimal)

Server Tag Address	Format	Data Type	Notes
linttag	Standard	Boolean	Invalid: Boolean is not supported.
linttag	Standard	Byte	Invalid: Byte is not supported.
linttag	Standard	Word	Invalid: Word is not supported.
linttag	Standard	Double	Value = 1.16767E+15
linttag	Standard	Date	Value = 2007-01-01T16:46:40.000*
linttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
linttag [3]	Array Element	Double	Invalid: Tag is not an array.
linttag {3}	Array w/o Offset	Double	Invalid: Tag is not an array.
linttag {1}	Array w/o Offset	Double	Value = [1.16767E+15]
linttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lintag [3] {1}	Array w/ Offset	Double	Invalid: Tag is not an array.
linttag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
linttag / 1	String	String	Invalid: Syntax/data type not supported.

*Date values are in universal time (UTC), not localized time.

LINT, ULINT, and LWORD Array Tag -
dintarraytag [2,2] = [0, 1.16767E+15],[9.4666E+14, 9.46746E+14] where:
1.16767E+15 == 2007-01-01T16:46:40.000 (date)
9.4666E+14 == 1999-12-31T17:06:40.000
9.46746E+14 == 2000-01-1T17:00:00.000
0 == 1970-01-01T00:00:00.000

Server Tag Address	Format	Data Type	Notes
lintarraytag	Standard	Boolean	Invalid: Boolean not supported.
lintarraytag	Standard	Byte	Invalid: Byte not supported.
lintarraytag	Standard	Word	Invalid: Word not supported.

lintarraytag	Standard	Double	Invalid: Tag cannot be an array.
lintarraytag	Standard	Date	Invalid: Tag cannot be an array.
lintarraytag [1]	Array Element	Double	Invalid: Server tag missing dimension 2 address.
lintarraytag [1,1]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
lintarraytag [1,1]	Array Element	Double	Value = 9.46746E+14
lintarraytag [1,1]	Array Element	Date	Value = 2000-01-01T17:00:00.000*
lintarraytag {4}	Array w/o Offset	Double	Value = [0, 1.16767E+15, 9.4666E+14, 9.46746E+14]
lintarraytag {2} {2}	Array w/o Offset	Double	Value = [0, 1.16767E+15][9.4666E+14, 9.46746E+14]
lintarraytag {4}	Array w/o Offset	Date	Invalid: Date array not supported.
lintarraytag {1}	Array w/o Offset	Double	Value = 0
lintarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lintarraytag [0,1] {2}	Array w/ Offset	Double	Value = [1.16767E+15, 9.4666E+14]
lintarraytag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
lintarraytag / 1	String	String	Invalid: Syntax/data type not supported.

*Date values are in universal time (UTC), not localized time.

REAL

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Byte, Char** Word, Short, BCD*** DWord, Long, LBCD Float****	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array** Word Array, Short Array, BCD Array*** DWord Array, Long Array, LBCD Array Float Array****	The Native Tag must be an array.
Array w/o Offset	Boolean Array Byte Array, Char Array** Word Array, Short Array, BCD Array*** DWord Array, Long Array, LBCD Array Float Array****	1. Use this case to have the bits within an REAL in array form. This is not an array of REALs in Boolean notation. 2. Applies to bit-within-REAL only. Example: tag_1.0{32}. 3. The .bit plus the array size cannot exceed 32 bits. Example: tag_1.1{32} exceeds an REAL, tag_1.0{32} does not. If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	1. The range is limited from 0 to 31. 2. If the Native Tag is an array, the bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0. **Note:** Float is cast to a DWord to allow referencing of bits.
Standard	Boolean* Byte, Char** Word, Short, BCD***	None

	DWord, Long, LBCD Float****	
String	String	1. If accessing a single element, the controller tag does not need to be an array. **Note:** The value of the string will be the ASCII equivalent of the REAL value (clamped to 255). Example: SINT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string will be the null-terminated ASCII equivalent of all the REALs (clamped to 255) in the string. 1 character in string = 1 REAL, clamped to 255. **Note:** REAL strings are not packed. For greater efficiency, use SINT strings or the STRING structure instead.

*Non-zero values will be clamped to True.
**Values exceeding 255 will be clamped to 255.
***Values exceeding 65535 will be clamped to 65535.
****Float value will be a valid IEEE single precision Floating point number.

Examples

Examples highlighted in yellow signify common use cases.

REAL Atomic Tag - realtag = 512.5 (decimal)

Server Tag Address	Format	Data Type	Notes
realtag	Standard	Boolean	Value = True
realtag	Standard	Byte	Value = 255
realtag	Standard	Word	Value = 512
realtag	Standard	DWord	Value = 512
realtag	Standard	Float	Value = 512.5
realtag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
realtag [3]	Array Element	DWord	Invalid: Tag is not an array.
realtag {3}	Array w/o Offset	DWord	Invalid: Tag is not an array.
realtag {1}	Array w/o Offset	Float	Value = [512.5]
realtag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
realtag [3] {1}	Array w/ Offset	Float	Invalid: Tag is not an array.
realtag . 3	Bit	Boolean	Value = True
realtag . 0 {32}	Array w/o Offset	Boolean	Value = [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,...0] Bit value for 512
realtag	String	String	Invalid: Syntax / data type not supported.
realtag	String	String	Invalid: Syntax / data type not supported.

REAL Array Tag - realarraytag [4,4] = [[82.1,69.2,65.3,76.4],[256.5,257.6,258.7,259.8], [9.0,10.0,11.0,12.0],[13.0,14.0,15.0,16.0]]

Server Tag Address	Format	Data Type	Notes
realarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
realarraytag	Standard	Byte	Invalid: Tag cannot be an array.
realarraytag	Standard	Word	Invalid: Tag cannot be an array.

realarraytag	Standard	DWord	Invalid: Tag cannot be an array.
realarraytag	Standard	Float	Invalid: Tag cannot be an array.
realarraytag [3]	Array Element	Float	Invalid: Server tag missing dimension 2 address.
realarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
realarraytag [1,3]	Array Element	Float	Value = 259.8
realarraytag {10}	Array w/o Offset	Byte	Value = [82,69,65,76,255,255,255,255,9,10]
realarraytag {2} {5}	Array w/o Offset	Float	Value = [82.1,69.2,65.3,76.4,256.5] [257.6,258.7,259.8,9,10]
realarraytag {1}	Array w/o Offset	Float	Value = 82.1
realarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
realarraytag [1,3] {4}	Array w/ Offset	Float	Value = [259.8,9.0,10.0,11.0]
realarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
realarraytag [1,3] . 3	Bit	Boolean	Value = 0
realarraytag [1,3] . 0 {32}	Array w/o Offset	Boolean	Value = [1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0] Bit value for 259
realarraytag	String	String	Invalid: Syntax / data type not supported.
realarraytag	String	String	Invalid: Syntax / data type not supported.

LREAL

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	Double*	The Native Tag must be an array.
Array w/ Offset	Double Array	The Native Tag must be an array.
Array w/o Offset	Double Array	If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	Invalid: Syntax/Data type not supported.
Standard	Double*	None.
String	String	Invalid: Syntax/Data type not supported.

*Double value will be a valid IEEE double precision Floating point number.

Examples

Examples highlighted in yellow signify common use cases.

LREAL Atomic Tag – lrealtag = 512.5 (decimal)

Server Tag Address	Format	Data Type	Notes
lrealtag	Standard	Boolean	Invalid: Data type not supported.
lrealtag	Standard	Byte	Invalid: Data type not supported.
lrealtag	Standard	Word	Invalid: Data type not supported.
lrealtag	Standard	DWord	Invalid: Data type not supported.
lrealtag	Standard	Double	Value = 512.5
lrealtag [3]	Array Element	Boolean	Invalid: Tag is not an array, and Boolean is invalid.
lrealtag [3]	Array Element	DWord	Invalid: Tag is not an array.
lrealtag {3}	Array w/o Offset	DWord	Invalid: Tag is not an array.
lrealtag {1}	Array w/o Offset	Double	Value = [512.5]
lrealtag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lrealtag [3] {1}	Array w/ Offset	Float	Invalid: Tag is not an array.
lrealtag . 3	Bit	Boolean	Invalid: Data type not supported.
lrealtag . 0 {32}	Array w/o Offset	Boolean	Invalid: Data type not supported.
lrealtag	String	String	Invalid: Syntax / data type not supported.
lrealtag	String	String	Invalid: Syntax / data type not supported.

LREAL Array Tag - realarraytag [4,4] = [[82.1,69.2,65.3,76.4],[256.5,257.6,258.7,259.8], [9.0,10.0,11.0,12.0],[13.0,14.0,15.0,16.0]]

Server Tag Address	Format	Data Type	Notes
lrealarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
lrealarraytag	Standard	Byte	Invalid: Tag cannot be an array.
lrealarraytag	Standard	Word	Invalid: Tag cannot be an array.
lrealarraytag	Standard	DWord	Invalid: Tag cannot be an array.
lrealarraytag	Standard	Double	Invalid: Tag cannot be an array.
lrealarraytag [3]	Array Element	Double	Invalid: Server tag missing dimension 2 address.
lrealarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
lrealarraytag [1,3]	Array Element	Double	Value = 259.8
lrealarraytag {10}	Array w/o Offset	Byte	Invalid: Data type not supported.
lrealarraytag {2} {5}	Array w/o Offset	Double	Value = [82.1,69.2,65.3,76.4,256.5] [257.6,258.7,259.8,9,10]
lrealarraytag {1}	Array w/o Offset	Double	Value = 82.1
lrealarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lrealarraytag [1,3] {4}	Array w/ Offset	Double	Value = [259.8,9.0,10.0,11.0]
lrealarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
lrealarraytag [1,3] . 3	Bit	Boolean	Value = 0
lrealarraytag [1,3] . 0 {32}	Array w/o Offset	Boolean	Invalid: Syntax/Data type not supported.
lrealarraytag	String	String	Invalid: Syntax / data type not supported.
lrealarraytag	String	String	Invalid: Syntax / data type not supported.

SHORT_STRING

For more information on the format, refer to **Address Formats**.

Format	Supported Data Types	Notes
Array Element	String	The Native Tag must be an array.
Array w/ Offset	N/A	N/A
Array w/o Offset	N/A	N/A
Bit	N/A	N/A
Standard	String	The length of the string is based on the length encoding contained within the Native Tag. If the string contains non-printable characters, these will be included in the string.
String	N/A	The length of the string needs to be specified in the tag address.

Examples

Examples highlighted in yellow signify common use cases.

SHORT_STRING Atomic Tag – stringtag = "mystring"

Server Tag Address	Format	Data Type	Notes
stringtag	Standard	String	Value = mystring.
stringtag	Standard	Byte	Invalid: Byte is not supported.
stringtag	Standard	Word	Invalid: Word is not supported.
stringtag [3]	Array Element	Boolean	Invalid: Tag is not an array, and Boolean is invalid.
stringtag [3]	Array Element	Double	Invalid: Tag is not an array.
stringtag {3}	Array w/o Offset	Double	Invalid: Tag is not an array.
stringtag {1}	Array w/o Offset	Double	Value = [1.16767E+15].

stringtag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lintag [3] {1}	Array w/ Offset	Double	Invalid: Tag is not an array.
stringtag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
stringtag / 1	String	String	Invalid: Syntax/data type not supported.

SHORT_STRING Array Tag – stringarraytag[2,2] = [one,two].[three,four]

Server Tag Address	Format	Data Type	Notes
stringarraytag	Standard	Boolean	Invalid: Boolean not supported.
stringarraytag	Standard	Byte	Invalid: Byte not supported.
stringarraytag	Standard	Word	Invalid: Word not supported.
stringarraytag	Standard	Double	Invalid: Tag cannot be an array.
stringarraytag	Standard	Date	Invalid: Tag cannot be an array.
stringarraytag [1]	Array Element	Double	Invalid: Server tag missing dimension 2 address.
stringarraytag [1,1]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
stringarraytag [1,1]	Array Element	String	Value: "four"
stringarraytag {4}	Array w/o Offset	String	Invalid: String array not supported.
stringarraytag {2} {2}	Array w/o Offset	String	Invalid: String array not supported.
stringarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
stringarraytag [0, 1] {2}	Array w/ Offset	String	Value: "three"
stringarraytag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
stringarraytag / 1	String	String	Invalid: Syntax not supported.

Error Codes

The following sections define error codes that may be encountered in the server's Event Log. For more information on a specific error code type, select a link from the list below.

Encapsulation Protocol Error Codes
CIP Error Codes

Encapsulation Protocol Error Codes

The following error codes are in hexadecimal.

Error Code	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.
0070	Invalid Target ID.

CIP Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0001	Connection Failure.*
0002	Insufficient resources.
0003	Value invalid.
0004	IOI could not be deciphered or tag does not exist.
0005	Unknown destination.
0006	Data requested would not fit in response packet.
0007	Loss of connection.
0008	Unsupported service.
0009	Error in data segment or invalid attribute value.
000A	Attribute list error.
000B	State already exists.
000C	Object Model conflict.
000D	Object already exists.
000E	Attribute not configurable.
000F	Permission denied.
0010	Device state conflict.
0011	Reply will not fit.
0012	Fragment primitive.
0013	Insufficient command data/parameters specified to execute service.
0014	Attribute not supported.
0015	Too much data specified.
001A	Bridge request too large.
001B	Bridge response too large.
001C	Attribute list shortage.
001D	Invalid attribute list.
001E	Embedded service error.
001F	Failure during connection.**
0022	Invalid reply received.
0025	Key segment error.
0026	Number of IOI words specified does not match IOI word count.
0027	Unexpected attribute in list.

***See Also:** 0x0001 Extended Error Codes

**See Also: 0x001F Extended Error Codes

Allen-Bradley Specific Error Codes

Error Code (hex)	Description
00FF	General Error.*

*See Also: 0x00FF Extended Error Codes

Note: For unlisted error codes, refer to the Rockwell Automation documentation.

0x0001 Extended Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0100	Connection in use.
0103	Transport not supported.
0106	Ownership conflict.
0107	Connection not found.
0108	Invalid connection type.
0109	Invalid connection size.
0110	Module not configured.
0111	EPR not supported.
0114	Wrong module.
0115	Wrong device type.
0116	Wrong revision.
0118	Invalid configuration format.
011A	Application out of connections.
0203	Connection timeout.
0204	Unconnected message timeout.
0205	Unconnected send parameter error.
0206	Message too large.
0301	No buffer memory.
0302	Bandwidth not available.
0303	No screeners available.
0305	Signature match.
0311	Port not available.
0312	Link address not available.
0315	Invalid segment type.
0317	Connection not scheduled.
0318	Link address to self is invalid.

Note: For unlisted error codes, refer to the Rockwell Automation documentation.

0x001F Extended Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0203	Connection timed out.

Note: For unlisted error codes, refer to the Rockwell Automation documentation.

0x00FF Extended Error Codes

The following error codes are in hexadecimal.

Error Code	Description
2104	Address out of range.

2105	Attempt to access beyond end of data object.
2106	Data in use.
2107	Data type is invalid or not supported.

Note: For unlisted error codes, refer to the Rockwell Automation documentation.

Error Descriptions

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

Address Validation

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

Address <address> is out of range for the specified device or register

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically references a location that is beyond the range of the device's supported locations.

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

Note:
For valid bit and array element ranges, refer to **Address Formats**.

Array size is out of range for address <address>

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically is requesting an array size that is too large.

Solution:
Specify a smaller value for the array or a different starting point by re-entering the address in the client application.

Note:
For valid array size ranges, refer to **Address Formats**.

Array support is not available for the specified address: <address>

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically contains an array reference for an address type that doesn't support arrays.

Solution:
Re-enter the address in the client application to remove the array reference or correct the address type.

Data Type <type> is not valid for device address <address>.

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically has been assigned an invalid data type.

Solution:
Modify the requested data type in the client application.

Device address <address> contains a syntax error.

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically contains one or more of the following errors:

1. Address doesn't conform to the tag address naming conventions.
2. Address is invalid according to the address format and the data type specified.
3. A Program Tag was specified incorrectly.
4. An invalid address format was used.

Solution:
Re-enter the address in the client application.

See Also:
Address Formats
Addressing Atomic Data Types

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

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically references a location that is valid for the communications protocol but not supported by the target device.

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

Device address <address> is read only.

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically has a requested access mode that is not compatible with what the device supports for that address.

Solution:
Change the access mode in the client application.

Memory could not be allocated for tag with address <address> on device <device name>.

Error Type:
Warning

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

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

Missing address

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically has no length.

Solution:
Re-enter the address in the client application.

Communication Messages

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

Device <device> responded with CIP error: Status Code=<status code>, Ext. Status Code=<extended status code>.
Device <device> responded with DF1 error: Data Link Layer NAK.
Device <device> responded with DF1 error: Frame received failed checksum validation.
Device <device> responded with DF1 error: No data received.
Device <device> responded with DF1 error: Slave sink/source full.
Device <device> responded with DF1 error: Status Code=<status code>.
Fragmentation Protocol NAK, Error Code=<error code>.
Frame received from device <device> contains errors.
Unexpected Fragmentation Command <command>.
Unexpected Fragmentation Function received.

Device <device> responded with CIP error: Status Code=<status code>, Ext. Status Code=<extended status code>.

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during a request. All reads and writes within the request failed.

Solution:
The solution depends on the error code(s) returned.

See Also:
CIP Error Codes

Device <device> responded with DF1 error: Data Link Layer NAK.

Error Type:
Warning

Possible Cause:
1. The server sent an invalid response.
2. The server sent a response in an unexpected size.
3. The server's Link Protocol settings may not match the device configuration.

Solution:
Ensure that the device's Link Protocol settings match those defined in the channel. If this error occurs frequently, contact Technical Support.

Device <device> responded with DF1 error: Frame received failed checksum validation.

Error Type:
Warning

Possible Cause:
1. The server sent an invalid response.
2. The server sent a response in an unexpected size.
3. The server's Link Protocol settings may not match the device configuration.

Solution:
Ensure that the device's Link Protocol settings match those defined in the channel. If this error occurs frequently, contact Technical Support.

Device <device> responded with DF1 error: No data received.

Error Type:
Warning

Possible Cause:
1. The server sent an invalid response.
2. The server sent a response in an unexpected size.
3. The server's Link Protocol settings may not match the device configuration.

Solution:
Ensure that the device's Link Protocol settings match those defined in the channel. If this error occurs frequently, contact Technical Support.

Device <device> responded with DF1 error: Slave sink/source full.

Error Type:
Warning

Possible Cause:
The slave device cannot accept any more requests from the master. The client may be requesting data too fast.

Solution:
The driver will automatically poll and re-poll the slave device in order to empty its source and make room for the responses from requests that were previously in the full sink. If this error occurs frequently, decrease the Update Rate on suspected tags (which are not necessarily the tags that are being written).

Device <device> responded with DF1 error: Status Code=<status code>.

Error Type:
Warning

Possible Cause:
1. The node cannot be found.
2. A duplicate node was detected.

Solution:
Check the status and extended status codes that are being returned by the PLC. The codes are displayed in hexadecimal.

Fragmentation Protocol NAK, Error Code=<error code>.

Error Type:
Warning

Possible Cause:
1. The server sent an invalid response.
2. The server sent a response in an unexpected size.
3. The server's Link Protocol settings may not match the device configuration.

Solution:
Ensure that the device's Link Protocol settings match those defined in the channel. If this error occurs frequently, contact Technical Support.

Frame received from device <device> contains errors.

Error Type:
Warning

Possible Cause:
1. The packets are misaligned (due to connection/disconnection between the PC and device).
2. There is bad cabling connecting the device that is causing noise.
3. An incorrect frame size was received.
4. There is a TNS mismatch.
5. An invalid response command was returned from the device.

Solution:
The driver will recover from this error without intervention. If this error occurs frequently, there may be an issues with the cabling or the device itself.

Unexpected Fragmentation Command <command>.

Error Type:
Warning

Possible Cause:
The device's response was unexpected.

Solution:
Ensure that the device's Link Protocol settings match those defined in the channel. If this error occurs frequently, contact Technical Support.

Unexpected Fragmentation Function received.

Error Type:
Warning

Possible Cause:
The device's response was unexpected.

Solution:
Ensure that the device's Link Protocol settings match those defined in the channel. If this error occurs frequently, contact Technical Support.

Device Specific Messages

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

Device <device name> is not responding.
Encapsulation error occurred during a request to device <device name>. [Encap. Error=<code>].
Error occurred during a request to device <device name>. [CIP Error=<code>, Ext. Error=<code>].

Device <device name> is not responding.

Error Type:
Warning

Possible Cause:
1. The Ethernet connection between the device and the Host PC is broken.
2. The communications parameters for the Ethernet connection are incorrect.
3. The named device may have been assigned an incorrect IP address.
4. When using the Serial Gateway device model, one or more devices has an incorrect serial port configuration.
5. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

Solution:
1. Verify the cabling between the PC and the device.
2. Verify that the correct port is specified for the named device.
3. Verify that the IP address given to the named device matches that of the actual device.
4. Verify that all devices have the correct serial port and system protocol configuration.
5. Increase the Request Timeout setting so that the entire response can be handled.

Encapsulation error occurred during a request to device <device name>. [Encap. Error=<code>].

Error Type:
Warning

Possible Cause:
The device returned an error within the Encapsulation portion of the Ethernet/IP packet during a request. All reads and writes within the request failed.

Solution:
The driver will attempt to recover from such an error. If the problem persists, contact Technical Support. This excludes error 0x02, which is device-related, not driver-related.

See Also:
Encapsulation Protocol Error Codes

Error occurred during a request to device <device name>. [CIP Error=<code>, Ext. Error=<code>].

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during a request. All reads and writes within the request failed.

Solution:
The solution depends on the error code(s) returned.

See Also:
CIP Error Codes

Read Errors (Blocking)

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

Read request for <count> element(s) starting at <tag address> on device <device name> failed due to a framing error. Block Deactivated.
Unable to read <count> element(s) starting at <tag address> on device <device name>. [CIP Error=<code>, Ext. Error=<code>].
Unable to read <count> element(s) starting at <tag address> on device <device name>. Block Deactivated.
Unable to read <count> element(s) starting at <tag address> on device <device name>. Block does not support multi-element arrays. Block Deactivated.
Unable to read <count> element(s) starting at <tag address> on device <device name>. Data type <type> is illegal for this block. Block Deactivated.
Unable to read <count> element(s) starting at <tag address> on device <device name>. Data type <type> not supported. Block Deactivated.
Unable to read <count> element(s) starting at <tag address> on device <device name>. Native Tag data type <type> unknown. Block Deactivated.

Read request for <count> element(s) starting at <tag address> on device <device name> failed due to a framing error. Block Deactivated.

Error Type:
Warning

Possible Cause:
A read request for tags <tag address> to <tag address>+<count> failed due to one of the following reasons:

1. Incorrect request service code.
2. Received more or less bytes than expected.

Solution:
If this error occurs frequently, there may be an issue with the cabling or the device itself. If the error occurs frequently for a specific tag, contact Technical Support. Increasing the request attempts will also give the driver more opportunities to recover from this error. In response to this error, <count> elements of the block will be deactivated; thus, it will not be processed again.

Unable to read <count> element(s) starting at <tag address> on device <device name>. [CIP Error=<code>, Ext. Error=<code>].

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during a read request for the tag.

Solution:
The solution depends on the error code(s) returned.

See Also:
CIP Error Codes

Unable to read <count> element(s) starting at <tag address> on device <device name>. Block Deactivated.

Error Type:
Warning

Possible Cause:
1. The Ethernet connection between the device and the Host PC is broken.
2. The communication parameters for the Ethernet connection are incorrect.
3. The named device may have been assigned an incorrect IP address.

Solution:
1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.
3. Verify that the IP address given to the named device matches that of the actual device.

Note:
In response to this error, <count> elements of the block will be deactivated; thus, it will not be processed again.

Unable to read <count> element(s) starting at <tag address> on device <device name>. Block does not support multi-element arrays. Block Deactivated.

Error Type:
Warning

Possible Cause:
A read request for tags <tag address> to <tag address>+<count>, failed because the driver does not support multi-element array access to the given Native Tag.

Solution:
Change the data type or address for tags within this block to one that is supported. In response to this error, <count> elements of the block will be deactivated; thus, it will not be processed again.

See Also:
Addressing Atomic Data Types

Unable to read <count> element(s) starting at <tag address> on device <device name>. Data type <type> is illegal for this block.

Error Type:
Warning

Possible Cause:
A read request for tags <tag address> to <tag address>+<count>, failed because the client's tag data type is illegal for the given Native Tag.

Solution:
Change the data type for tags within this block to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean would remedy this problem. In response to this error, <count> elements of the block will be deactivated; thus, it will not be processed again.

See Also:
Addressing Atomic Data Types

Unable to read <count> element(s) starting at <tag address> on device <device name>. Data type <type> not supported.

Error Type:
Warning

Possible Cause:
A read request for tags <tag address> to <tag address>+<count> failed because the client's tag data type is not supported.

Solution:
Change the data type for tags within this block to one that is supported. In response to this error, <count> elements of the block will be deactivated; thus, it will not be processed again.

See Also:
Addressing Atomic Data Types

Unable to read <count> element(s) starting at <tag address> on device <device name>. Native Tag data type <type> unknown. Block deactivated.

Error Type:
Warning

Possible Cause:
A read request for tags <tag address> to <tag address>+<count>, failed because the Native Tag's data type is not currently supported.

Solution:
Contact Technical Support so that support may be added for this type. In response to this error, <count> elements of the block will be deactivated; it will not be processed again.

Write Errors

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

Unable to write to <tag address> on device <device name>.
Unable to write to tag <tag address> on device <device name>. [CIP Error=<code>, Ext. Status=<code>].
Unable to write to tag <tag address> on device <device name>. Data type <type> is illegal for this tag.
Unable to write to tag <tag address> on device <device name>. Data type <type> not supported.
Unable to write to tag <tag address> on device <device name>. Native Tag data type <type> unknown.
Unable to write to tag <tag address> on device <device name>. Tag does not support multi-element arrays.

Write request for tag <tag address> on device <device name> failed due to a framing error.

Unable to write to <tag address> on device <device name>.

Error Type:
Warning

Possible Cause:
1. The Ethernet connection between the device and the Host PC is broken.
2. The communication parameters for the Ethernet connection are incorrect.
3. The named device may have been assigned an incorrect IP address.

Solution:
1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.
3. Verify that the IP address given to the named device matches that of the actual device.

Unable to write to tag <tag address> on device <device name>. [CIP Error=<code>, Ext. Status=<code>].

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during a write request for the tag.

Solution:
The solution depends on the error code(s) returned.

See Also:
CIP Error Codes

Unable to write to tag <tag address> on device <device name>. Data type <type> is illegal for this tag.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the client's tag data type is illegal for the given Native Tag.

Solution:
Change the tag's data type to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean would remedy this problem.

See Also:
Addressing Atomic Data Types

Unable to write to tag <tag address> on device <device name>. Data type <type> not supported.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the client's tag data type is not supported.

Solution:
Change the tag's data type to one that is supported.

See Also:
Addressing Atomic Data Types

Unable to write to tag <tag address> on device <device name>. Native Tag data type <type> unknown.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the Native Tag's data type is not currently supported.

Solution:
Contact Technical Support so that support may be added for this type.

Unable to write to tag <tag address> on device <device name>. Tag does not support multi-element arrays.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the driver does not support multi-element array access to the given Native Tag.

Solution:
Change the tag's data type or address to one that is supported.

See Also:
Addressing Atomic Data Types

Write request for tag <tag address> on device <device name> failed due to a framing error.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed after so many retries due to one of the following reasons:

1. Incorrect request service code.
2. Received more or less bytes than expected.

Solution:
If this error occurs frequently, there may be an issue with the cabling or the device itself. Increasing the Retry Attempts will also give the driver more opportunities to recover from this error.

Read Errors (Non-Blocking)

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

Read request for tag <tag address> on device <device name> failed due to a framing error. Tag deactivated.
Unable to read <tag address> on device <device name>. Tag deactivated.
Unable to read tag <tag address> on device <device name>. [CIP Error=<code>, Ext. Error=<code>].
Unable to read tag <tag address> on device <device name>. Data type <type> is illegal for this tag. Tag deactivated.
Unable to read tag <tag address> on device <device name>. Data type <type> not supported. Tag deactivated.
Unable to read tag <tag address> on device <device name>. Native Tag data type <type> unknown. Tag deactivated.
Unable to read tag <tag address> on device <device name>. Tag does not support multi-element arrays. Tag deactivated.

Read request for tag <tag address> on device <device name> failed due to a framing error. Tag deactivated.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed due to one of the following reasons:

1. Incorrect request service code.
2. Received more or less bytes than expected.

Solution:
If this error occurs frequently, there may be an issue with the cabling or the device itself. If the error occurs frequently for a specific tag, contact Technical Support. Increasing the request attempts will also give the driver more opportunities to recover from this error. In response to this error, the tag will be deactivated; thus, it will not be processed again.

Unable to read <tag address> on device <device name>. Tag deactivated.

Error Type:
Warning

Possible Cause:
1. The Ethernet connection between the device and the Host PC is broken.
2. The communication parameters for the Ethernet connection are incorrect.
3. The named device may have been assigned an incorrect IP address.

Solution:
1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.
3. Verify that the IP address given to the named device matches that of the actual device.

Note:
In response to this error, the tag will be deactivated and will not be processed again.

Unable to read tag <tag address> on device <device name>. [CIP Error=<code>, Ext. Error=<code>].

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during a read request for the tag.

Solution:
The solution depends on the error code(s) returned.

See Also:
CIP Error Codes

Unable to read tag <tag address> on device <device name>. Data type <type> is illegal for this tag. Tag deactivated.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client's tag data type is illegal for the given Native Tag.

Solution:
Change the tag's data type to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean would remedy this problem. In response to this error, the tag will be deactivated; thus, it will not be processed again.

See Also:
Addressing Atomic Data Types

Unable to read tag <tag address> on device <device name>. Data type <type> not supported. Tag deactivated.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client's tag data type is not supported.

Solution:
Change the tag's data type to one that is supported. In response to this error, the tag will be deactivated; thus, it will not be processed again.

See Also:
Addressing Atomic Data Types

Unable to read tag <tag address> on device <device name>. Native Tag data type <type> unknown. Tag deactivated.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the Native Tag's data type is not currently supported.

Solution:
Contact Technical Support so that support may be added for this type. In response to this error, the tag will be deactivated; thus, it will not be processed again.

Unable to read tag <tag address> on device <device name>. Tag does not support multi-element arrays. Tag deactivated.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the driver does not support multi-element array access to the given Native Tag.

Solution:
Change the tag's data type or address to one that is supported. In response to this error, the tag will be deactivated; thus, it will not be processed again.

See Also:
Addressing Atomic Data Types

Glossary

Native Tag-Based Addressing

Term	Definition
Array Element	Element within a native Array Tag. For client/server access, the element must be an atomic. For example, ARRAYTAG [0].
Array with Offset	Client/Server array tag whose address has a native Array Element specified. For example, ARRAYTAG [0] {5}.
Array w/o Offset	Client/Server array tag whose address has no native Array Element specified. For example, ARRAYTAG {5}.
Atomic Data Type	A pre-defined, non-structured Native data type. For example, SINT, DINT.
Atomic Tag	A Native Tag defined with an Atomic Data Type.
Client	An HMI/SCADA or data bridging software package utilizing OPC, DDE, or proprietary client/server protocol to interface with the server.
Client/Server Data Type	Data type for tags defined statically in the server or dynamically in a client. The data types supported in the client depends on the client in use.*
Client/Server Tag	Tag defined statically in the server or dynamically in a client. These tags are different entities than Native Tags. A Native Tag name becomes a Client/Server Tag address when referencing such Native Tag.
Client/Server Array	Row x column data presentation format supported by the server and by some clients. Not all clients support arrays.
CCW	Connected Components Workbench.
Native Data Type	A data type defined in CCW for Micro800 controllers.
Native Tag	A tag defined in CCW for Micro800 controllers.
Native Array Data Type	A multi-dimensional array (1, 2 or 3 dimensions possible) supported in CCW for Micro800 controllers. All atomic data types support Native Arrays. Not all structured data types support Native Arrays.
Array Tag	A Native Tag defined with a native Array Data Type.
Pre-Defined Data Type	A Native data type supported and pre-defined by CCW for Micro800 controllers.*
User-Defined Data Type	A Native data type supported by CCW and defined by the user for Micro800 controllers.*
Server	The OPC/DDE/proprietary server utilizing this Allen-Bradley Micro800 Serial Driver.
Structured Data Type	A pre-defined or user-defined data type, consisting of members whose data types are atomic or structure in nature.
Structure Tag	A Native Tag defined with a Structured Data Type.

*The data types supported in the server are listed in **Data Types Description**.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W