Application Technique

Allen-Bradley

EtherNet/IP Socket Interface

Allen-Bradley • Rockwell Software

Rockwell Automation

Important User Information

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

⚠	**WARNING:** Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
⚠	**ATTENTION:** Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.
⚠	**SHOCK HAZARD:** Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
⚠	**BURN HAZARD:** Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.

Preface

This publication describes the socket interface that you can use to program MSG instructions to communicate between a Logix5000™ controller via an EtherNet/IP module and Ethernet devices that do not support the EtherNet/IP application protocol, such as bar code scanners, RFID readers, or other standard Ethernet devices.

Additional Resources

These documents contain additional information concerning related products from Rockwell Automation.

Resource	Description
Logix5000 Controllers General Instructions Reference Manual, 1756-RM003	Details on programming MSG instructions.
Logix5000 Controllers Common Procedures Programming Manual, 1756-PM001	Information on programming Logix5000 controllers.
EtherNet/IP Modules in Logix5000 Control Systems User Manual, ENET-UM001	Information on configuring EtherNet/IP modules in Logix5000 systems.
Industrial Automation Wiring and Grounding Guidelines, publication 1770-4.1	Provides general guidelines for installing a Rockwell Automation industrial system.
Product Certifications website, http://www.ab.com	Provides declarations of conformity, certificates, and other certification details.

You can view or download publications at http:/www.rockwellautomation.com/literature/. To order paper copies of technical documentation, contact your local Allen-Bradley® distributor or Rockwell Automation sales representative.

Notes:

Socket Interface Architecture

The socket interface lets you use a Logix5000 controller to communicate via an EtherNet/IP module with Ethernet devices, such as bar code scanners, RFID readers, or other standard Ethernet devices, that do not support the EtherNet/IP application protocol.

Socket services are available with these modules:

- 1756-EN2*xx* ControlLogix® EtherNet/IP communication modules, firmware revision 5.007 or later
- 1756-EWEB ControlLogix Ethernet/IP web server module, firmware revision 4.006 or later
- 1768-EWEB CompactLogix™ Ethernet/IP web server module, firmware revision 1.002 or later
- 1769-L30ER, 1769-L30ERM, 1769-L30ER-NSE, 1769-L33ER, 1769-L33ERM, and 1769-L36ERM CompactLogix controllers, firmware revision 20.011 or later
- 1769-L24ER-QB1B, 1769-L24ER-QBFC1B, 1769-L27ERM-QBFC1B CompactLogix controllers, firmware revision 20.011 or later
- 1769-L16ER, 1769-L18ER, 1769-L18ERM CompactLogix controllers, firmware revision 20.011 or later

IMPORTANT	MicroLogix™ 1400 controllers also support socket capability, but the information in this document does not apply to those products. For details on those products, see the MicroLogix 1400 Programmable Controllers Reference Manual, publication 1766-RM001D.

Before you use the socket interface, you should be familiar with these concepts:

- Basic TCP/IP, UDP, and socket programming concepts
- How to write socket programs in a programming language, such as C or Visual Basic
- How to use diagnostic tools, such as a network sniffer
- The application protocols of the devices and applications with which the Logix5000 controller will communicate
- How to write ladder logic or structured text for a Logix5000 controller

Socket Interface Architecture

The socket interface is implemented via the socket object in the EtherNet/IP module. Logix5000 controller programs communicate with the socket object via MSG instructions. MSG requests to the socket object are similar to socket API calls in most computer operating systems. The socket object services let you open connections, accept incoming connections, send data, and receive data.

To communicate with another device, you must understand the other device's application protocol. The EtherNet/IP module has no application protocol knowledge. The module only makes the socket services available to programs in Logix5000 controllers.

Number and Type of Sockets

The 1756-EN2*xx* modules and CompactLogix 5370 controllers support 32 socket instances. The 1756-EWEB and 1768-EWEB modules support 20 socket instances. Each instance can be one of these types:

- UDP socket—Sends and receives UDP datagrams.
- TCP client socket—The Logix5000 program initiates the connection.
- TCP server socket—Another device initiates the connection to the Logix5000 program.
- TCP listen socket—Listens on a specified port number for incoming connections.

These options are available for UDP and TCP send and receive services.

Type	Communication	Send (Write)	Receive (Read)
UDP	Unicast	Yes	Yes
	Multicast	Yes	Yes
	Broadcast	Yes	Yes
TCP	Unicast	Yes	Yes
	Multicast	NA	NA
	Broadcast	NA	NA

You must have a listen socket for each TCP port number that accepts connections. Multiple TCP server sockets can share a listen socket if the connections are made to the same port number.

You can partition the available socket instances between UDP and TCP sockets in these ways:

- Use all instances for client TCP connections
- Use one instance to listen for incoming TCP connections and then accept the remaining connections from other device
- Perform both TCP client and server operations
- Perform both TCP and UDP operations

These socket services are available.

Socket Service	Socket Instance	Page
Socket Create	Server or client	24
OpenConnection	Client	26
AcceptConnection	• If you issue an AcceptConnection service, the instance is a listen type. • If the AcceptConnection service returns an instance as a result of an incoming connection request, the socket instance is a server type.	28
ReadSocket	Server or client	30
WriteSocket	Server or client	32
DeleteSocket	Server or client	34
DeleteAllSockets	Server or client	35
ClearLog	Server or client	36
JoinMulticastAddress	Server or client	37
DropMulticastAddress	Server or client	38

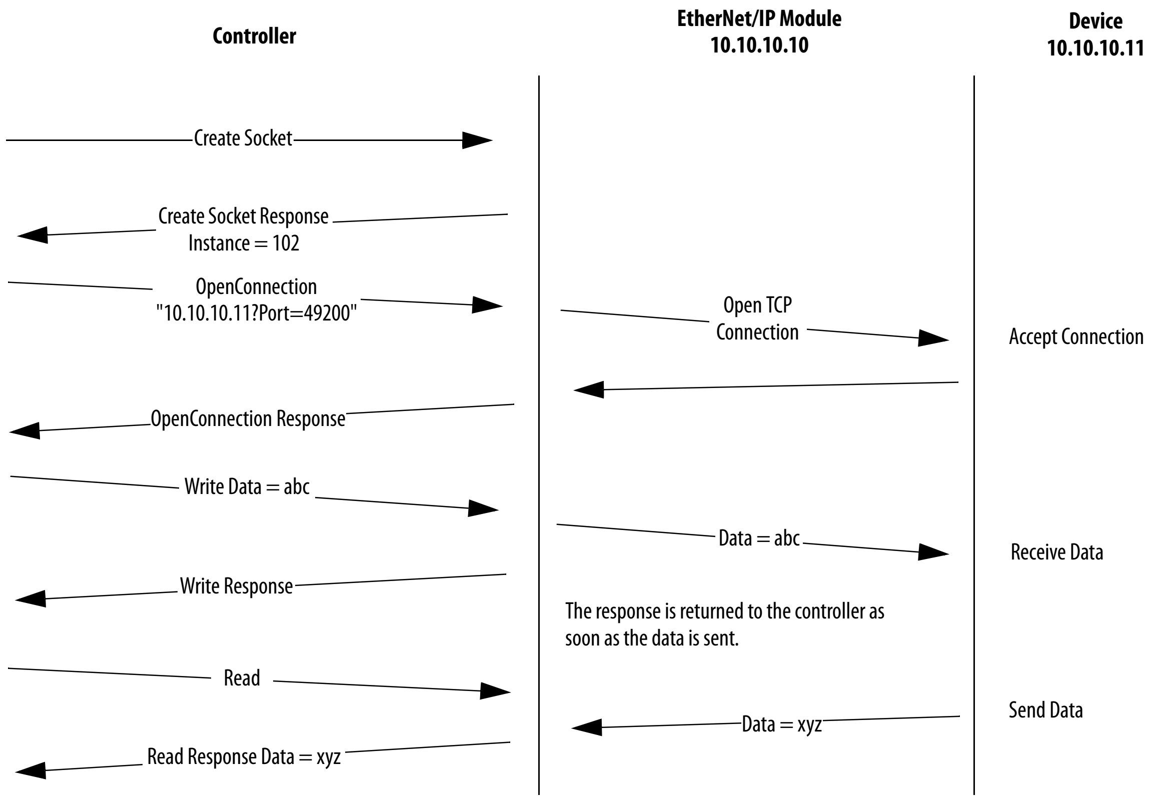
Once you open a connection on a client socket instance, you cannot use the same socket instance to accept incoming connections. Similarly, if you accept connections on a socket instance, you cannot then use the instance to open outgoing connections. This behavior is consistent with standard socket API behavior.

Typical Sequence of Transactions for a TCP Client

The following diagram shows a typical sequence of socket interface transactions with the Logix5000 controller acting as a TCP client. Each transaction between the Logix5000 controller and the EtherNet/IP module is a MSG instruction.

The following example shows the Logix5000 controller sending data to a device and then the device sending a response. This is a typical sequence of transactions. Depending on the application protocol, the device could instead initiate sending data to the Logix5000 controller once the connection is open.

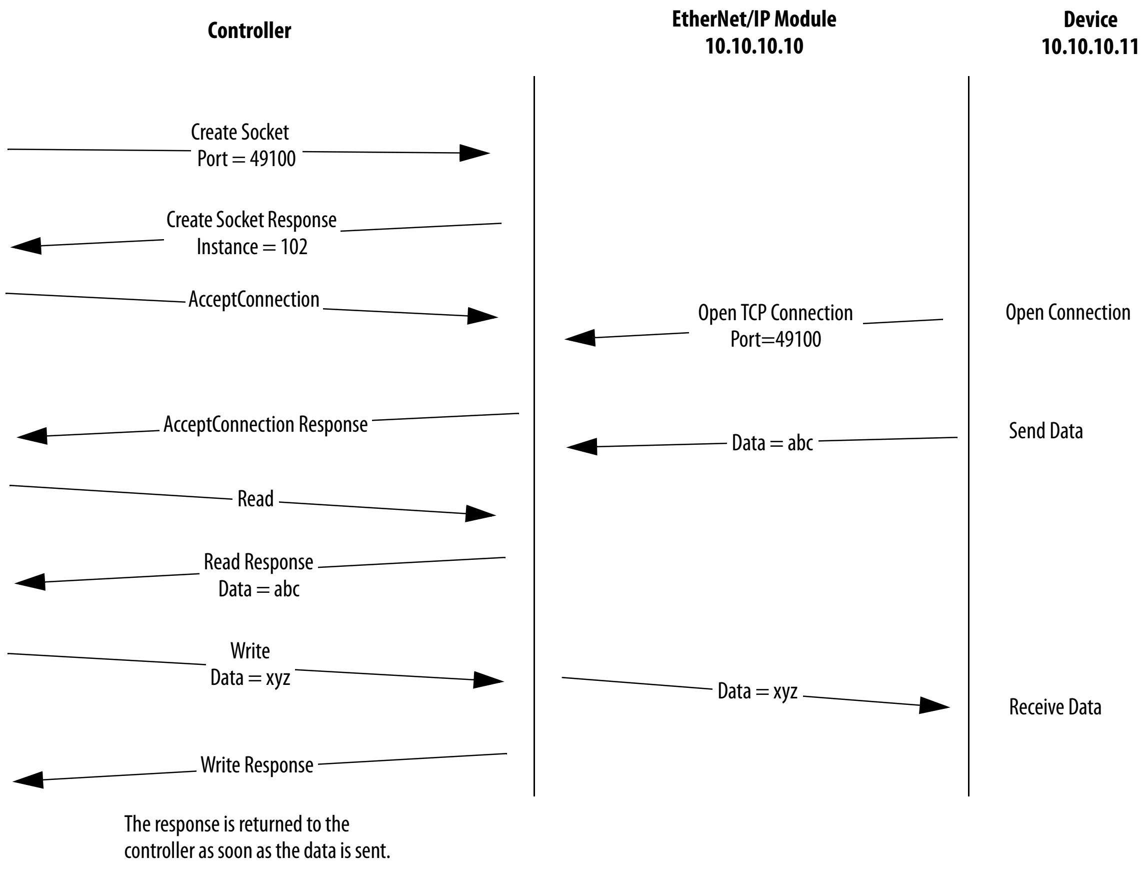
Also, each write transaction does not require an application response or acknowledgement. The application protocol determines the exact sequence of application transactions.



Typical Sequence of Transactions for a TCP Server

The following diagram shows a typical sequence of socket interface transactions with the Logix5000 controller as a TCP server. Each transaction between the Logix5000 controller and EtherNet/IP module is a MSG instruction.

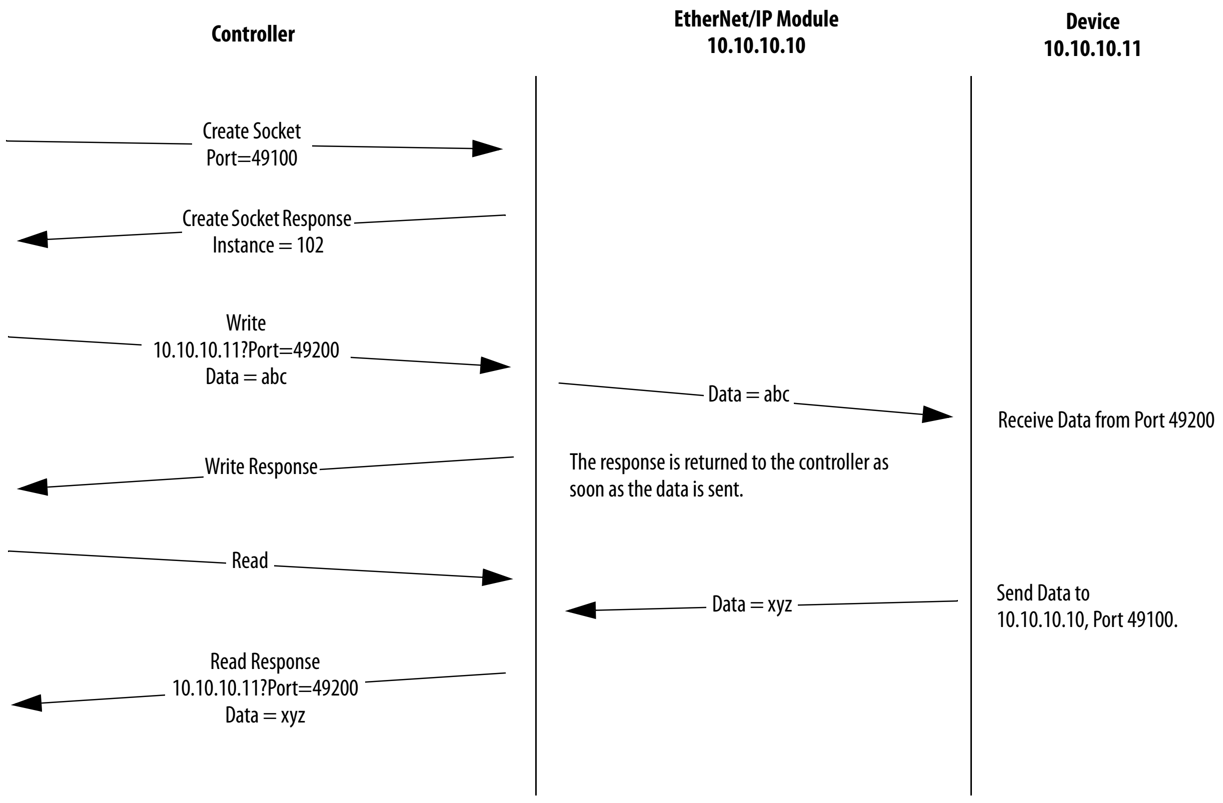
The following is a typical sequence of transactions. The exact sequence of sending and receiving data depends on the application protocol.


Controller
EtherNet/IP Module 10.10.10.10
Device 10.10.10.11
Create Socket Port = 49100
Create Socket Response Instance = 102
AcceptConnection
Open TCP Connection Port=49100
Open Connection
AcceptConnection Response
Data = abc
Send Data
Read
Read Response Data = abc
Write Data = xyz
Data = xyz
Receive Data
Write Response
The response is returned to the controller as soon as the data is sent.

Typical Sequence of Transactions for UDP without OpenConnection

The following diagram shows a typical sequence of socket interface transactions for UDP communication without using the OpenConnection service to specify the destination address. In this case, the Logix5000 controller specifies the destination for each datagram and receives the sender's address along with each datagram it receives. Each transaction between the Logix5000 controller and the EtherNet/IP module is a MSG instruction.

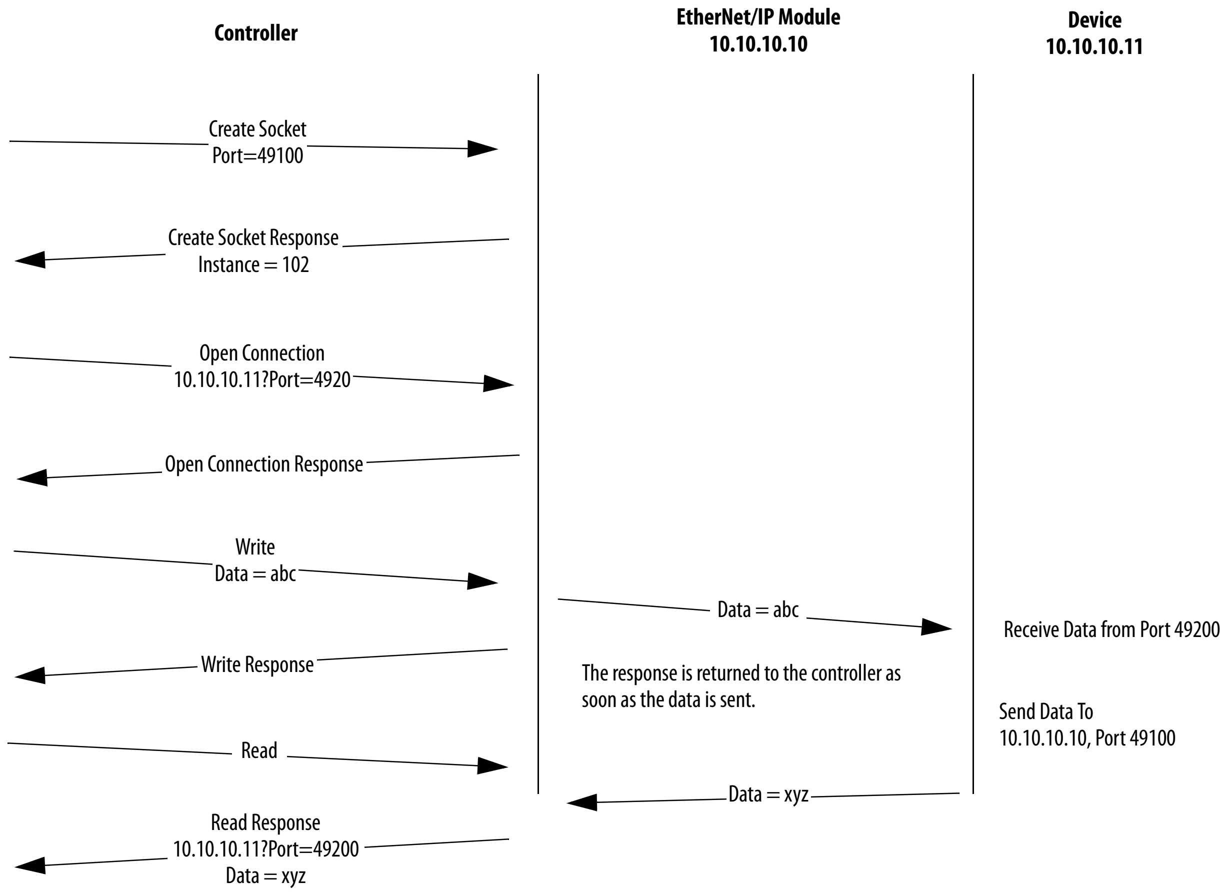
The example below shows the Logix5000 controller sending data to a device and then the device sending a response. This is a typical sequence of transactions. Depending on the application protocol, the device could instead initiate sending data to the Logix5000 controller. Also, each Write transaction does not require an application response or acknowledgement. The application protocol determines the exact sequence of application transactions.


Controller
EtherNet/IP Module 10.10.10.10
Device 10.10.10.11
Create Socket Port=49100
Create Socket Response Instance = 102
Write 10.10.10.11?Port=49200 Data = abc
Data = abc
Receive Data from Port 49200
Write Response
The response is returned to the controller as soon as the data is sent.
Read
Data = xyz
Send Data to 10.10.10.10, Port 49100.
Read Response 10.10.10.11?Port=49200 Data = xyz

Typical Sequence of Transactions for UDP with OpenConnection

The following diagram shows a typical sequence of socket interface transactions for UDP communication when using the OpenConnection service to specify the destination address. Each transaction between the Logix5000 controller and the EtherNet/IP module is a MSG instruction.

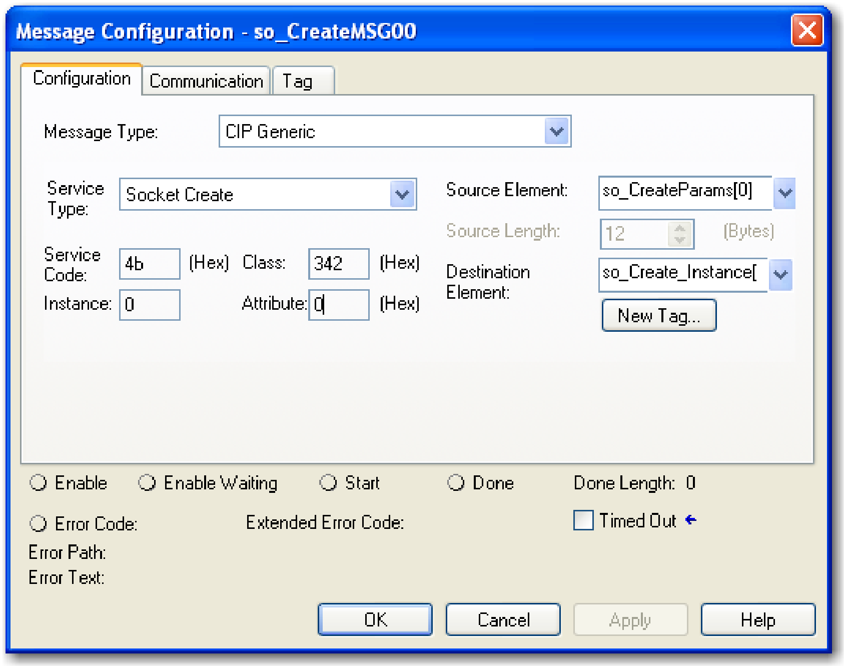
The following is a typical sequence of transactions. The exact sequence of sending and receiving data depends on the application protocol.


Controller
EtherNet/IP Module 10.10.10.10
Device 10.10.10.11
Create Socket Port=49100
Create Socket Response Instance = 102
Open Connection 10.10.10.11?Port=4920
Open Connection Response
Write Data = abc
Data = abc
Receive Data from Port 49200
Write Response
The response is returned to the controller as soon as the data is sent.
Send Data To 10.10.10.10, Port 49100
Read
Data = xyz
Read Response 10.10.10.11?Port=49200 Data = xyz

Communicate with the Socket Object via a MSG Instruction

In a Logix5000 controller program, use a CIP Generic MSG instruction to request socket services.

On the Configuration tab, configure the parameters as described in Table 1.



Message Type: CIP Generic

Service Type: Socket Create

Source Element: so_CreateParams[0]

Source Length: 12 (Bytes)

Service Code: 4b (Hex) Class: 342 (Hex)

Destination Element: so_Create_Instance[

Instance: 0 Attribute: 0 (Hex)

Enable Enable Waiting Start Done Done Length: 0

Error Code: Extended Error Code: Timed Out

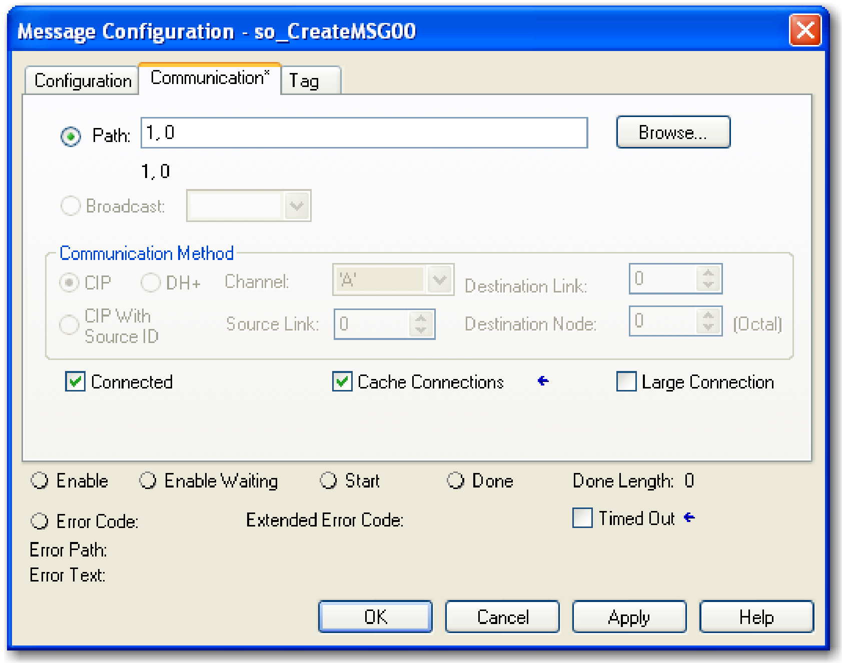
Error Path:

Error Text:

Table 1 - Configuration Tab

Field	Description
Message Type	Choose CIP Generic.
Service Type	Depending on your version of RSLogix 5000™ software, do one of the following: • With RSLogix 5000 software, version 15 or later, choose a socket service type. The software automatically completes the Service Code and Class fields. • With RSLogix 5000 software, version 15 or earlier, choose Custom. Manually complete the Service Code and Class fields. Choose Get Attributes Single or Set Attributes Single when getting or setting a Socket Object attribute. For more information, see Access Socket Attributes on page 41.
Service Code	Type the unique service code that corresponds to the socket service you chose in the Service Type field.
Class	Type 342 (hexadecimal) for the socket object.
Instance	Type one of these values: • 0 for Socket Create, Delete All Sockets, and ClearLog services • Instance number returned by Socket Create for other services Use a relay ladder instruction or structured text statement to move the returned instance number from a Socket Create service into the .Instance member of a MSG instruction.
Attribute	Type an attribute value only when getting or setting an attribute, but not when using other services.
Source Element	Choose the tag that contains the request parameters for the socket service. To define the request parameters, create a user-defined data type for the tag.
Source Length	Enter the length of the source element.
Destination Element	Choose the tag that contains the response data returned by the socket service. To define the response data, create a user-defined data type for the tag.

On the Communication tab, configure the parameters described in Table 2.


Message Configuration - so_CreateMSG00

Configuration | Communication* | Tag

[✓ Path:] 1, 0

1, 0

Broadcast:

Communication Method

[✓ CIP] DH+ Channel: 'A' Destination Link: 0

CIP With Source ID Source Link: 0 Destination Node: 0 (Octal)

[✓ Connected] [✓ Cache Connections] Large Connection

Enable Enable Waiting Start Done Done Length: 0

Error Code: Extended Error Code: Timed Out

Error Path:

Error Text:

IMPORTANT	All CompactLogix 5370 controllers must use **unconnected** MSG instructions. If you are configuring a message for a CompactLogix 5370 controller, make sure the Connected checkbox on the Message Configuration dialog box is cleared.

Table 2 - Communication Tab

Field	Description
Path	Enter the communication path to the EtherNet/IP module.
Large Connection(1)	Select the checkbox to use a large 4000 byte connection size, or clear the checkbox use a standard 500 byte connection size. A large connection is only available with connected MSG instructions. For information about using the Connected or Cach Connections options, refer to the Logix5000 Controllers Messages Programming Manual, publication 1756-PM012. **IMPORTANT:** To efficiently use controller memory, use large connections only for ReadSocket or WriteSocket services that require more than the standard 500-byte connection size, as shown in Table 3.

(1) Large connections are supported only by 1756-EN2xx ControlLogix modules in RSLogix 5000 software, version 20 or later.

The maximum amount of data you can send or receive depends on how you configure the MSG instruction, as shown in Table 3. The size of the data does not include the parameters in the ReadSocket and WriteSocket services.

Table 3 - Maximum Packet Sizes

Service	Unconnected Size	Standard Connection Size	Large Connection Size
ReadSocket	484 bytes	484 bytes	3984 bytes
WriteSocket	462 bytes	472 bytes	3972 bytes

If a MSG requests more than the maximum packet size (standard or large), the module might return a failure status and the MSG instruction might set the .ER bit:

- For TCP sockets, if the application data is larger than the maximum size, you can issue multiple ReadSocket or WriteSocket services to receive or send the entire application message.
- For UDP sockets, the size of application data cannot exceed the maximum sizes for the ReadSocket and WriteSocket services.

Service Timeouts

You must specify a time-out parameter in milliseconds for any service that might not complete immediately, such as OpenConnection, AcceptConnection, ReadSocket, and WriteSocket services. The timeout tells the socket object the maximum amount of time it should wait when attempting to complete the service. While waiting for the service to complete, the MSG instruction is enabled.

If the requested service does not complete before the time-out period expires, the socket object returns a response to the service request. See the service descriptions in Chapter 2 for the content of the response.

IMPORTANT	Make the value of the service time-out parameter shorter than the MSG instruction timeout. Otherwise, application data could be lost.

MSG Instruction Timeouts

The default MSG instruction timeout is 30 seconds. The maximum MSG timeout is approximately 35 minutes. Specify the MSG instruction timeout by setting the appropriate member of the MSG tag:

- If the MSG is unconnected, set the UnconnectedTimeout member.
- If the MSG is connected, set the ConnectionRate and TimeoutMultiplier member.

The MSG timeout is determined by multiplying the ConnectionRate by the TimeoutMultiplier. A TimeoutMultiplier of 0 corresponds to multiplier of 4, 1 corresponds to multiplier of 8, and so on.

Socket Instance Timeouts

Each socket instance has an inactivity timeout with a default of 5 minutes. If a socket instance receives no service requests for the amount of time specified by the inactivity timeout, the socket instance is deleted. If you then try to use the socket instance, the MSG instruction receives the error class or instance not supported.

You can change the timeout by setting the inactivity time-out attribute via the Set Attribute service. See Socket Instance Attributes on page 43.

If you put the controller in Program mode and then back into Run mode before existing socket instances time out, you can receive errors when the program tries to create socket instances. Eventually the socket instances time out and you can create more instances.

IMPORTANT	Make sure the inactivity timeout is longer than the longest interval between socket operations. If the inactivity timeout is too short, socket instances may time out, resulting in MSG instruction errors.

Programming Considerations

Observe these programming considerations.

TCP Connection Loss

Your application program may encounter conditions that result in TCP connection loss. For example, a network cable can be unplugged, or a target device can be turned off.

Your application program should detect the loss of TCP connections and handle those events appropriately. You can detect connection loss when one of the following occurs:

- The ReadSocket service returns with an error.
- The WriteSocket service returns an extended error code other than 16#0000_0046. See Error Codes for Socket Services on page 47.

Depending on the application, try these actions:

- Try to reestablish the connection, such as in the case of a client connection.
- Wait for another incoming connection to be established, such as in the case of a server connection.

If you want to reestablish communication with the other device, complete these actions:

- Delete the socket instance for the lost connection.
- If the connection is a client connection, create a new socket instance and issue an OpenConnection service to the target device.
- If the connection is a server connection, issue an AcceptConnection service to wait for another connection from the remote device.

ControlLogix Enhanced Redundancy

IMPORTANT	Socket instances created in EtherNet/IP modules are **not** crossloaded in an enhanced redundancy system.

If your application uses sockets in an enhanced redundancy system, your application program must manage switchovers in these ways:

- After a switchover, socket instances in the EtherNet/IP module in the old primary chassis must be recreated in the EtherNet/IP module in the new primary chassis via controller logic.
- Sockets connected outside of the enhanced redundancy system must recognize that communication is lost with socket instances in the EtherNet/IP module in the old primary chassis after a switchover. This loss of communication, as described in TCP Connection Loss on page 17, is caused by a change in the EtherNet/IP module's IP address after a switchover.
- Although socket instances in the EtherNet/IP module in the old primary chassis are automatically deleted once their inactivity timeout expires, it is possible that a second switchover may occur before the timeout expires. To be sure that these non-functioning socket instances are deleted prior to a second switchover, your application program can issue a message to delete all the sockets in the event of a switchover before creating functioning socket instances.

To learn more about enhanced redundancy systems, refer to the ControlLogix Enhanced Redundancy System User Manual, publication 1756-UM535.

EtherNet/IP Module Reset

If the EtherNet/IP module is reset, for example by cycling power or with removal and insertion under power (RIUP), all socket instances are lost.

If you create new socket instances while MSG instructions are still using the old instance numbers, there is a possibility that the new instance numbers will match the old instance numbers. In this situation, your old MSG instructions may succeed but may not be communicating with the correct remote device.

Handle this situation by monitoring the status of the EtherNet/IP module via a GSV instruction. If you lose communication with the EtherNet/IP module, the Logix5000 program should reinitialize its socket communication.

Change Controller Mode between Run and Program

If the Logix5000 controller transitions from Run mode to Program mode while socket requests are active, the transition does not complete until all of the outstanding MSG requests complete or time out. If you have long time-out values, you can experience an unexpectedly long time for the Run-to-Program transition to complete.

Alleviate long transition times by appropriately setting the time-out parameter for the socket services. In the Logix5000 program, you can also set the .TO bit for any outstanding socket-related MSG instruction. This causes the MSG instruction to time out and set the .ER bit.

If the controller transitions from Run mode to Program mode, then back to Run mode again, previous socket instances may still exist on the EtherNet/IP module. The previous socket instances time out eventually. Depending on the number of sockets you need, your program may encounter errors during Run-Program-Run transitions because all of the available socket instances are in use.

To alleviate this situation, follow this procedure:

1. Wait for all socket instances to time out before putting the controller in Run mode.
2. When the Logix5000 program starts, use the DeleteAllSockets service to delete any previous instances.

The DeleteAllSockets service deletes all socket instances, not just those created by the controller calling the service.

Application Messages and TCP

A TCP connection is a byte stream between two application entities. The application protocol determines the message formats. Messages can be fixed size or variable size.

If an application sends variable size messages, a common strategy is to first send a fixed-size header containing the size of the message followed by the message. The receiving device can first issue a ReadSocket service of the fixed size header to determine the remaining size, and then issue a subsequent ReadSocket service to receive the remaining data.

Application Messages and Uninhibited Modules

Unlike I/O connected via an EtherNet/IP module, communication via messaging to socket instances can continue when a module is inhibited. If you want to stop socket communication when a module is inhibited, your application code must detect the status of the module and take the appropriate action.

Partial Reads

It is possible for a read service to return a BufLen that is less than the requested amount of data. For example, your program may request 100 bytes of data. Because TCP is a byte stream and not a datagram protocol, you can receive less than 100 bytes when the read service returns.

Depending on the application protocol, issue additional read requests to receive all of the data. If the application protocol dictates that all messages are 100 bytes, then you must issue additional read requests until you receive 100 bytes. If the application protocol uses variable size messages, your program needs additional logic to handle variable message sizes as defined by the application protocol.

When issuing multiple read requests, be careful to adjust the destination tag that receives the data so that data is not overwritten.

If the read request times out before any data is received, a BufLen of 0 is returned with success (0) status.

This fragment of structured text logic shows an example of handling a partial read request.

```
/* copy the message we just read */

COP (ReadResponse.Buf[0], ReadBuf[CurrentLen],

ReadResponse.BufLen);

CurrentLen := CurrentLen + ReadResponse.BufLen;

/* do we need to read more data get a complete message? */

if (CurrentLen < ApplicationMsgLen) then

/* issue another read */

ReadParams.BufLen := ApplicationMsgLen - CurrentLen;

MSG (ReadMSG0);

       end_if;
```

Partial Writes

Although uncommon, your program may need to handle a situation where a write service is unable to send all of the specified bytes. Such a situation can occur if the write service is called multiple times before the target application can receive the data.

If the write service is not able to send all of the requested data, your program should issue subsequent writes to send the remaining data. Your program should also adjust the source tag, so that old data is not sent.

If the number of bytes written is less than requested, an extended error is returned, as well as the actual length of the data sent.

This fragment of structured text logic shows an example of handling a partial write service.

```
if (WriteMSG0.ER) then

/* write failed. if the extended error code was 16#0000_0046,
then it means less than the requested byte were sent. */

if (WriteMSG0.EXERR = 70) then

/* need to issue another write, with the data that was not
sent */

       SentLen := WriteResponse;  /* here's what was sent */

       /* adjust the size  */

       WriteParams.BufLen := WriteParams.BufLen - SentLen;

       /* copy remaining data to send to MSG buffer */

       COP (WriteBuf[SentLen], WriteParams.Buf[0],
WriteParams.BufLen);

/* BufLen = Timeout + Sockaddr + data length */

WriteMSG0.REQ_LEN := 4 + 12 + WriteParams.BufLen;

       MSG (WriteMSG0);

end_if;

       end_if;
```

Performance Considerations

As noted previously, the socket interface enables a Logix5000 controller to communicate via an EtherNet/IP module with Ethernet devices, such as bar code scanners, RFID readers, or other standard Ethernet devices, that do not support the EtherNet/IP application protocol. The socket interface, via messaging, is not well suited for real-time control as communication with this method is not scheduled or deterministic.

There are a variety of factors that can affect the performance of the socket interface. For examples of some of the factors to consider, search the Knowledgebase for answer ID 36682. To access the Knowledgebase, log on to the Support Center at http://rockwellautomation.custhelp.com/.

Socket Object Services

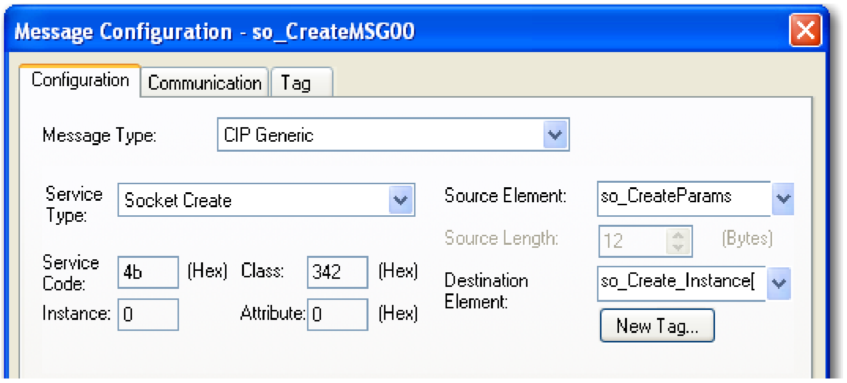
For a socket object, application data has no inherent byte order. The service receives data in the same byte order as it is sent. However, Logix5000 controllers store data in CIP byte order (little endian). For example, if you issue a write service with 1 DINT, that DINT is sent over a TCP connection or in a UDP datagram in CIP byte order. If you issue a read service and your destination tag for the response contains a DINT, the Logix5000 controller assumes the incoming data is in CIP byte order. Depending on the native byte order of the application you are communicating with, you may need to convert the byte order in your Logix5000 program or in the application.

To check your MSG configuration in RSLogix 5000 software, version 15 or later, choose a service type from the Service Type pull-down menu on the Configuration tab of the Message Configuration dialog box. The software automatically completes the Service Code and Class fields. With RSLogix 5000 software, version 15 and earlier, choose Custom from the Service Type pull-down menu and manually complete the Service Code and Class fields.

Socket Create

The Socket Create service creates an instance of the socket object. The service returns an instance number that you use in the subsequent socket operations. Call the Socket Create service with instance 0 (Socket object class).

Parameter	Value
Service Type	Socket Create
Service Code	4b
Class	342
Instance	0
Attribute	0


Message Configuration - so_CreateMSG00

Message Type: CIP Generic

Service Type: Socket Create

Service Code: 4b (Hex) Class: 342 (Hex)

Instance: 0 Attribute: 0 (Hex)

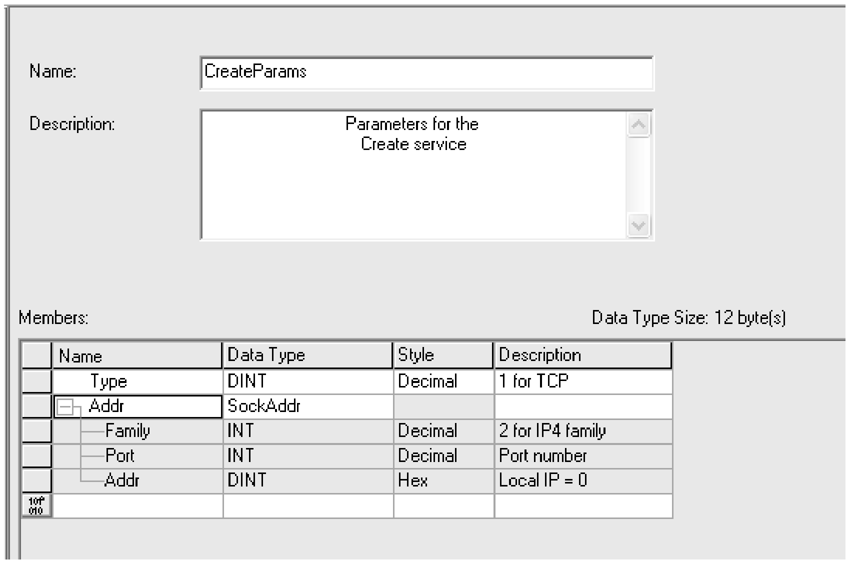
Source Element: so_CreateParams

Source Length: 12 (Bytes)

Destination Element: so_Create_Instance[

MSG Source Element

Choose a tag with a user-defined data type. Use the information in Table 4 to define the data type.


Name: CreateParams

Description: Parameters for the Create service

Members: Data Type Size: 12 byte(s)

Name	Data Type	Style	Description
Type	DINT	Decimal	1 for TCP
Addr	SockAddr		
Family	INT	Decimal	2 for IP4 family
Port	INT	Decimal	Port number
Addr	DINT	Hex	Local IP = 0

Table 4 - Data Type for Socket Create Source Element

Member Name	Data Type	Description
Type	DINT	Specify one of these values: • 1 for TCP • 2 for UDP
Addr	structure	A user-defined structure that specifies the address for the socket.
Family	INT	Specify the address family. Must be 2.
Port	INT	Specify the local port number on which an application is listening and receiving, or set to 0 if you want the EtherNet/IP module to choose the local port number: • For TCP client operations, specify 0 unless you want a specific local port number.For TCP server communication, specify the port number on which to accept incoming connection requests. • For UDP, specify a local port number to receive datagrams on a specific port.
Addr	DINT	Specify an IP address. Typically, set to 0 (any address).

MSG Source Length

Specify the size of the user-defined structure for the source element. In this example, CreateParams is 12 bytes.

MSG Destination Element

The MSG instruction returns the instance number of the socket it just created to the destination element. Specify a DINT tag.

Considerations

Use the instance returned by the Socket Create service on subsequent service requests.

Use a MOV instruction to move the instance to another MSG tag (the .Instance field).

If you use a local port number that is already in use by the EtherNet/IP module, you receive extended error code 16#0000_0030. The EtherNet/IP module uses these port numbers:

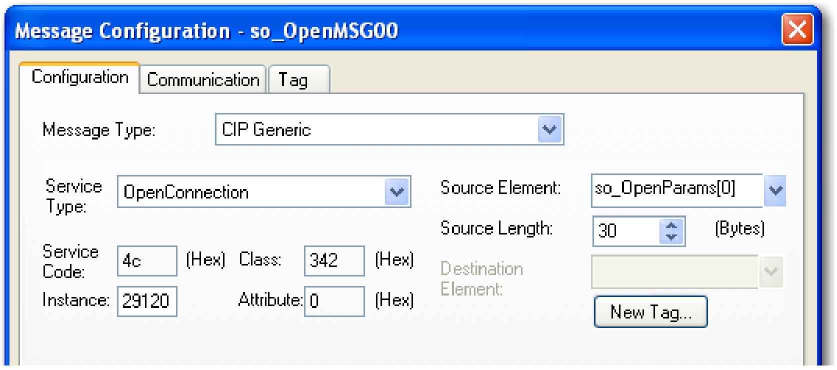
- 20, 21—FTP
- 25—SMTP
- 80—HTTP
- 123—NTP
- 161—SNMP
- 2222—EtherNet/IP
- 44818—EtherNet/IP

OpenConnection

The OpenConnection service does one of the following:

- Opens a TCP connection with the specified destination address
- For UDP, associates a destination IP address and port number with the specified socket

Parameter	Value
Service Type	OpenConnection
Service Code	4c
Class	342
Instance	from Socket Create
Attribute	0



Message Configuration - so_OpenMSG00

Configuration | Communication | Tag

Message Type: CIP Generic

Service Type: OpenConnection

Source Element: so_OpenParams[0]

Source Length: 30 (Bytes)

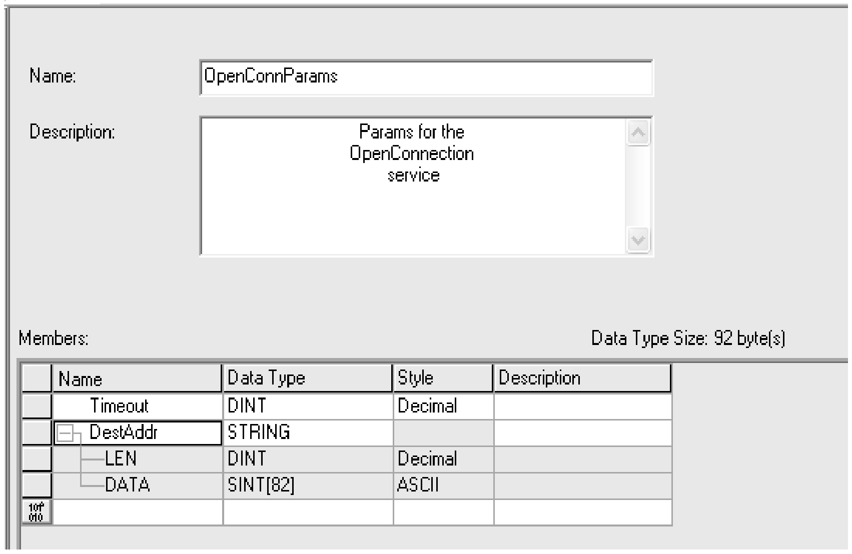
Service Code: 4c (Hex) Class: 342 (Hex)

Destination Element:

Instance: 29120 Attribute: 0 (Hex)

MSG Source Element

Choose a tag with a user-defined data type. Use the information in Table 5 to define the data type.



Name: OpenConnParams

Description: Params for the OpenConnection service

Members: Data Type Size: 92 byte(s)

Name	Data Type	Style	Description
Timeout	DINT	Decimal	
DestAddr	STRING		
LEN	DINT	Decimal	
DATA	SINT[82]	ASCII	

.

Table 5 - Data Type for OpenConnection Source Element

Member Name	Data Type	Description
Timeout	DINT	Specify the timeout in milliseconds.
DestAddr	STRING	Specify an array of characters (maximum of 64) to define the destination of the connection. Specify either of these: • Hostname?port=*xxxzaz* • IPAddr?port=*xxx* For example, to specify an IP address, enter 10.88.81.10?port=2813
.LEN	DINT	The length of the destination address.
.DATA	SINT array	The array containing the destination address.

The MSG instruction that issues the OpenConnection service should have a source length of 8 (Timeout + AddrLen) plus the number of characters in the destination address.

MSG Source Length

Specify 8 bytes (Timeout + AddrLen) + number of characters in the destination address.

MSG Destination Element

Not used. The MSG instruction does not return any data.

Considerations

In some cases, the OpenConnection service can return before the time-out period without creating a TCP connection. For example, if the destination device is running, but is not listening for connections on the specified port number, the OpenConnection service returns with an error before the time-out period.

For UDP, the information you must specify depends on whether you use the OpenConnection service:

- If you use the OpenConnection service, you do not have to specify the IP address and port number each time you send data. If you do not specify an IP address and port number, you can only receive data from the previously specified IP address and port number until you call the OpenConnection service to specify a different IP address and port number.
- If you do **not** use the OpenConnection service, you must specify the destination address each time you call the WriteSocket service to send data. When you call the ReadSocket service, in addition to the data, you receive the address of the sender. You can then use the address of the sender to send a response via the WriteSocket service.

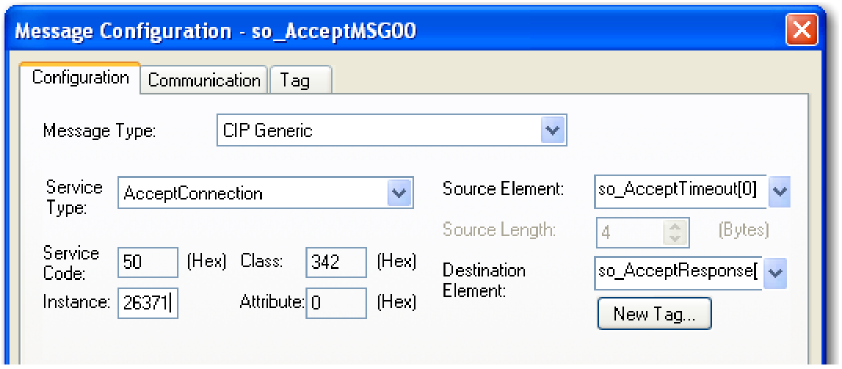
If you call the OpenConnection service on a UDP socket with an AddrLen of 0, this removes the association with the destination address.

AcceptConnection

The AcceptConnection service accepts a TCP connection request from a remote destination. Before calling the AcceptConnection service, call the Socket Create service and specify the local port number that will accept the connection. When the AcceptConnection service completes, it returns a socket instance that you use for sending and receiving data on the newly-created connection.

The AcceptConnection service is not valid for UDP sockets.

Parameter	Value
Service Type	AcceptConnection
Service Code	50
Class	342
Instance	from Socket Create
Attribute	0



Message Configuration - so_AcceptMSG00

Message Type: CIP Generic

Service Type: AcceptConnection

Service Code: 50 (Hex) Class: 342 (Hex)

Instance: 26371 Attribute: 0 (Hex)

Source Element: so_AcceptTimeout[0]

Source Length: 4 (Bytes)

Destination Element: so_AcceptResponse[

MSG Source Element

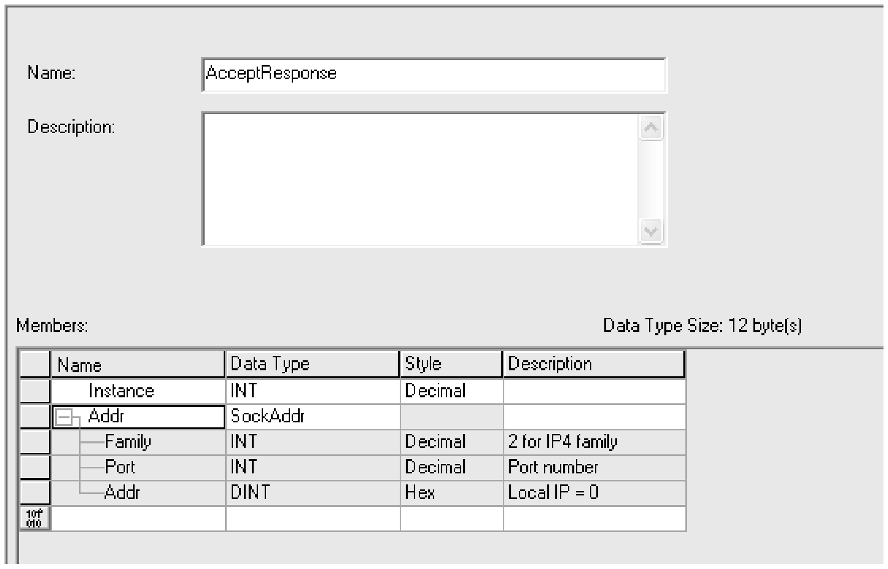
Choose a DINT tag to contain the timeout in milliseconds.

MSG Source Length

Specify 4 bytes (Timeout).

MSG Destination Element

Choose a tag with a user-defined data type. Use the information in Table 6 to define the data type.



Name: AcceptResponse

Description:

Members: Data Type Size: 12 byte(s)

Name	Data Type	Style	Description
Instance	INT	Decimal	
Addr	SockAddr		
Family	INT	Decimal	2 for IP4 family
Port	INT	Decimal	Port number
Addr	DINT	Hex	Local IP = 0

Table 6 - Data Type for AcceptConnection Destination Element

Member Name	Data Type	Description
Instance	DINT	Contains the instance for this service. Use this Instance on subsequent Read and Write services for this connection.
Addr	structure	A user-defined structure that contains the address for the socket.
Family	INT	Contains the address family. Must be 2.
Port	INT	Contains a local port number.
Addr	DINT	Contains an IP address.

Considerations

Create a separate socket instance by using the Socket Create service for each port number that will accept connections. After you create socket instances, call the AcceptConnection service to wait for an incoming connection request. You can accept connections on the same port number. Each call to the AcceptConnection service returns a different instance number to use when subsequently reading and writing data.

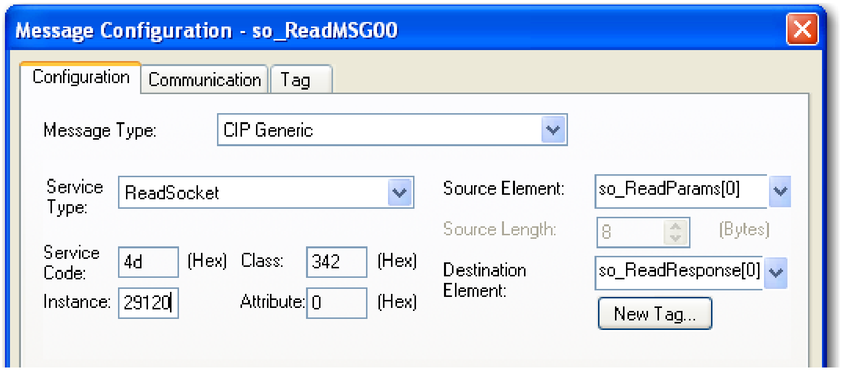
ReadSocket

The ReadSocket service reads data on a socket. You specify the number of bytes to receive. The service returns the number of bytes received.

For TCP, the ReadSocket service returns when any data is received, up to the requested number of bytes. If no data is received before the time-out period, the service returns a status of success by setting a message instruction Done Bit (.DN) and a BufLen of 0. The service can return fewer bytes than were requested. Your application might need to issue multiple read requests to receive an entire application message.

For UDP, the ReadSocket service completes when a datagram is available.

Parameter	Value
Service Type	ReadSocket
Service Code	4d
Class	342
Instance	from Socket Create
Attribute	0



Message Type: CIP Generic

Service Type: ReadSocket

Service Code: 4d (Hex) Class: 342 (Hex)

Instance: 29120 Attribute: 0 (Hex)

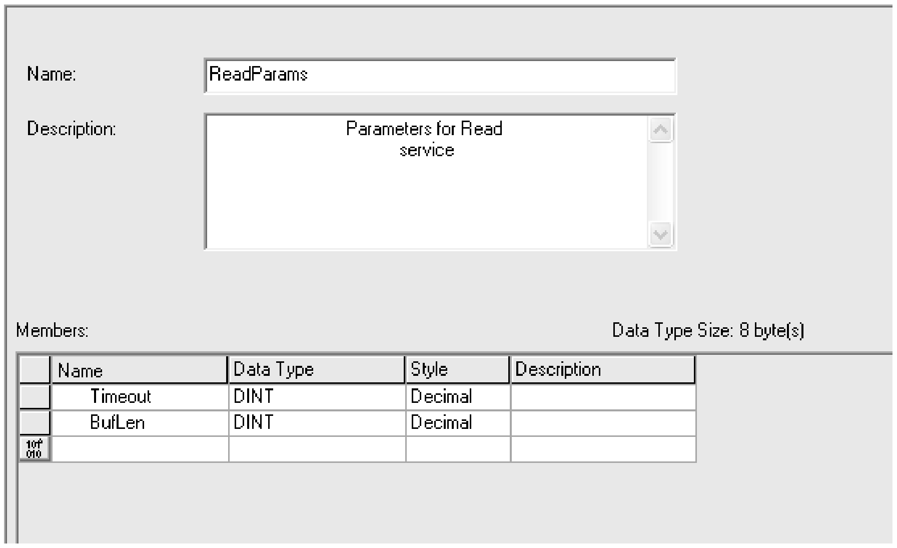
Source Element: so_ReadParams[0]

Source Length: 8 (Bytes)

Destination Element: so_ReadResponse[0]

MSG Source Element

Choose a tag with a user-defined data type. Use the information in Table 7 to define the data type.



Name: ReadParams

Description: Parameters for Read service

Members: Data Type Size: 8 byte(s)

Name	Data Type	Style	Description
Timeout	DINT	Decimal	
BufLen	DINT	Decimal	

Table 7 - Data Type for ReadSocket Source Element

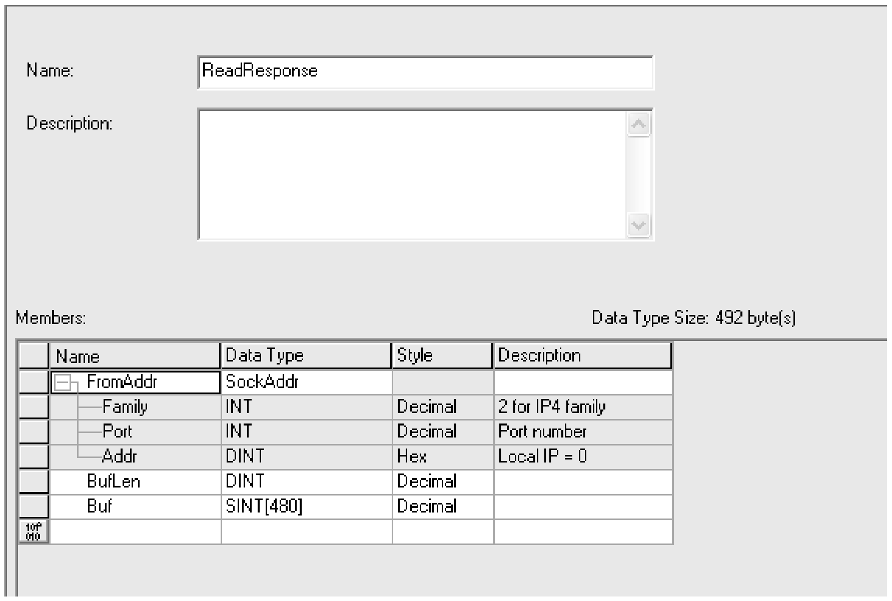
Member Name	Data Type	Description
Timeout	DINT	Specify the timeout in milliseconds.
BufLen	DINT	Specify the number of bytes of data to receive.

MSG Source Length

Specify 8 bytes (Timeout + BufLen).

MSG Destination Element

Choose a tag with a user-defined data type. Use the information in Table 8 to define the data type.



Name: ReadResponse

Description:

Members: Data Type Size: 492 byte(s)

Name	Data Type	Style	Description
FromAddr	SockAddr		
Family	INT	Decimal	2 for IP4 family
Port	INT	Decimal	Port number
Addr	DINT	Hex	Local IP = 0
BufLen	DINT	Decimal	
Buf	SINT[480]	Decimal	

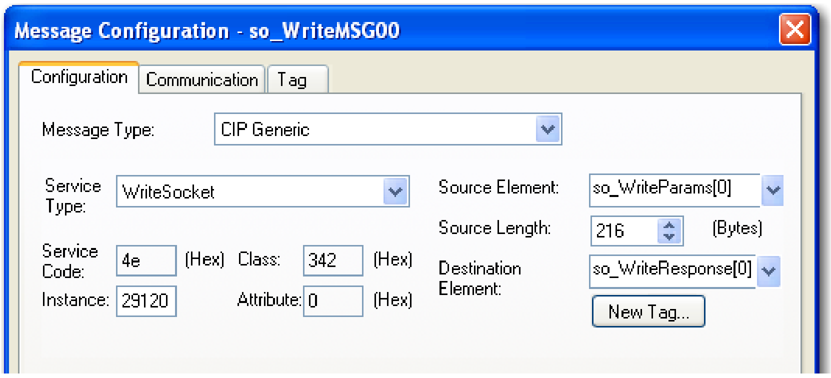
Table 8 - Data Type for ReadSocket Destination Element

Member Name	Data Type	Description
FromAddr	structure	A user-defined structure that will contain the address of the device sending UDP data. For TCP, this structure is not used and will contain all zeros. The TCP connection conveys all remote address information.
Family	INT	Contains the address family for UDP. Must be 2.
Port	INT	Contains the remote port number for UDP. This is the port that the remote device uses for receiving.
Addr	DINT	Contains the remote IP address for UDP
BufLen	DINT	Contains the number of bytes of data received.
Buf	SINT array	Contains the data. This number must be large enough to contain the maximum amount of data expected. For a standard connection, the maximum is SINT[484]; for a large connection the maximum is SINT [3984].

WriteSocket

The WriteSocket service sends data on a socket. You specify the number of bytes to send. The service attempts to send the requested number of bytes and returns the number of bytes sent.

Parameter	Value
Service Type	WriteSocket
Service Code	4e
Class	342
Instance	from Socket Create
Attribute	0



Message Configuration - so_WriteMSG00

Configuration | Communication | Tag

Message Type: CIP Generic

Service Type: WriteSocket

Service Code: 4e (Hex) Class: 342 (Hex)

Instance: 29120 Attribute: 0 (Hex)

Source Element: so_WriteParams[0]

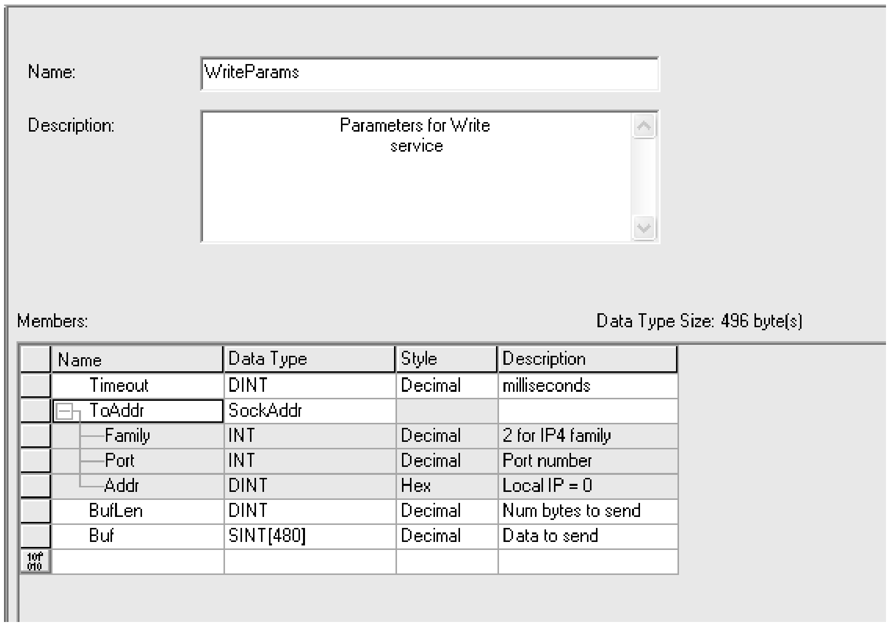
Source Length: 216 (Bytes)

Destination Element: so_WriteResponse[0]

New Tag...

MSG Source Element

Choose a tag with a user-defined data type. Use the information in Table 7 to define the data type.



Name: WriteParams

Description: Parameters for Write service

Members: Data Type Size: 496 byte(s)

Name	Data Type	Style	Description
Timeout	DINT	Decimal	milliseconds
ToAddr	SockAddr		
Family	INT	Decimal	2 for IP4 family
Port	INT	Decimal	Port number
Addr	DINT	Hex	Local IP = 0
BufLen	DINT	Decimal	Num bytes to send
Buf	SINT[480]	Decimal	Data to send

Table 9 - Data Type for WriteSocket Source Element

Member Name	Data Type	Description
Timeout	DINT	Specify the timeout in milliseconds.
ToAddr	structure	A user-defined structure that will contain the address to which to write UDP data. For TCP, this structure is not used and will contain all zeros. The TCP connection conveys all required remote address information.
Family	INT	Specify the address family. Must be 2 for UDP.
Port	INT	Specify the remote port number for UDP. This is the port that the remote device uses for receiving.
Addr	DINT	Specify the remote IP address for UDP.
BufLen	DINT	Specify the number of bytes of data to write.
Buf	SINT array	Contains the data. This number must be large enough to contain the maximum amount of data expected. For a standard connection, the maximum is SINT[472]; for a large connection the maximum is SINT [3972].

MSG Source Length

Specify 16 bytes (Timeout + Addr + BufLen) + number of bytes to write.

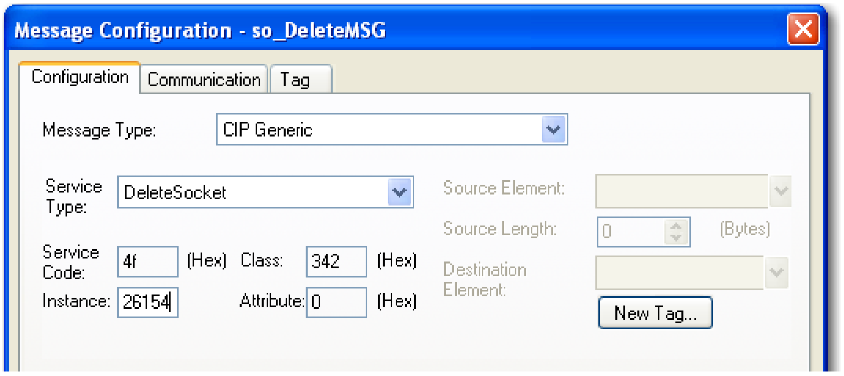
MSG Destination Element

The MSG instruction returns the number of bytes that were written. Choose a DINT tag.

DeleteSocket

Parameter	Value
Service Type	DeleteSocket
Service Code	4f
Class	342
Instance	from Socket Create
Attribute	0

The DeleteSocket service deletes a socket instance. For a TCP connection, the DeleteSocket service also closes the connection prior to deleting the instance.



Message Type: CIP Generic
Service Type: DeleteSocket
Service Code: 4f (Hex) Class: 342 (Hex)
Instance: 26154 Attribute: 0 (Hex)
Source Element:
Source Length: 0 (Bytes)
Destination Element:

MSG Source Element

Not used.

MSG Source Length

Specify 0 bytes.

MSG Destination Element

Not used.

Considerations

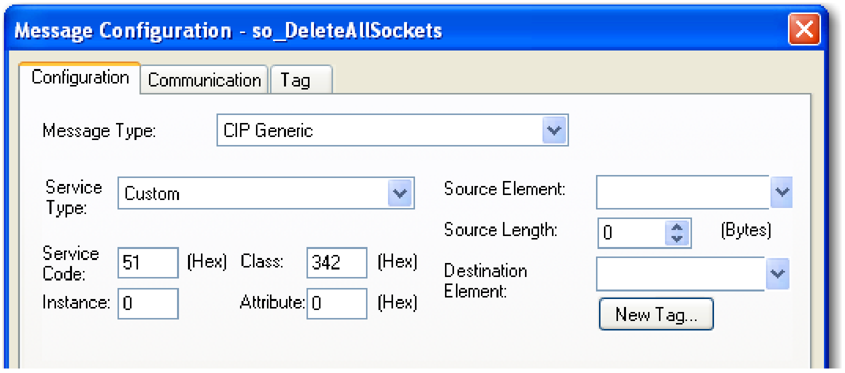
Delete a socket instance if it is no longer needed. If unused instances are not deleted and you continue to create additional instances, you can exceed the maximum number of instances.

DeleteAllSockets

The DeleteAllSockets service deletes all currently created socket instances. For TCP, the DeleteAllSockets service also closes all connections prior to deleting the instances.

Choose Custom for the service type. DeleteAllSockets is not an available option from the Service Type pull-down menu.

Parameter	Value
Service Type	Custom
Service Code	51
Class	342
Instance	0
Attribute	0


Message Configuration - so_DeleteAllSockets

Message Type: CIP Generic

Service Type: Custom

Service Code: 51 (Hex) Class: 342 (Hex)

Instance: 0 Attribute: 0 (Hex)

Source Element:

Source Length: 0 (Bytes)

Destination Element:

MSG Source Element

Not used.

MSG Source Length

Specify 0 bytes.

MSG Destination Element

Not used.

Considerations

Call the DeleteAllSockets service with instance 0.

IMPORTANT Be careful when using the DeleteAllSockets service when there are multiple controllers using the socket interface of the EtherNet/IP module. The service deletes all socket instances created by all controllers, not just the controller calling the service.

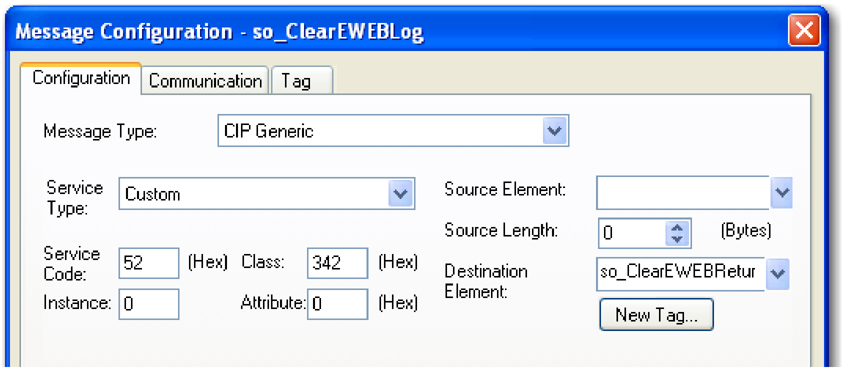
Use the DeleteAllSockets service as the first operation when the program first begins to operate.

ClearLog

The ClearLog service clears the debug log on the TCP/IP Socket Object web page. This service does not change the logging options.

Choose Custom for the service type. ClearLog is not an available option from the Service Type pull-down menu.

Parameter	Value
Service Type	Custom
Service Code	52
Class	342
Instance	0
Attribute	0


Message Configuration - so_ClearEWEBLog

Message Type: CIP Generic

Service Type: Custom

Service Code: 52 (Hex) Class: 342 (Hex)

Instance: 0 Attribute: 0 (Hex)

Source Element:

Source Length: 0 (Bytes)

Destination Element: so_ClearEWEBRetur

MSG Source Element

Not used.

MSG Source Length

Specify 0 bytes.

MSG Destination Element

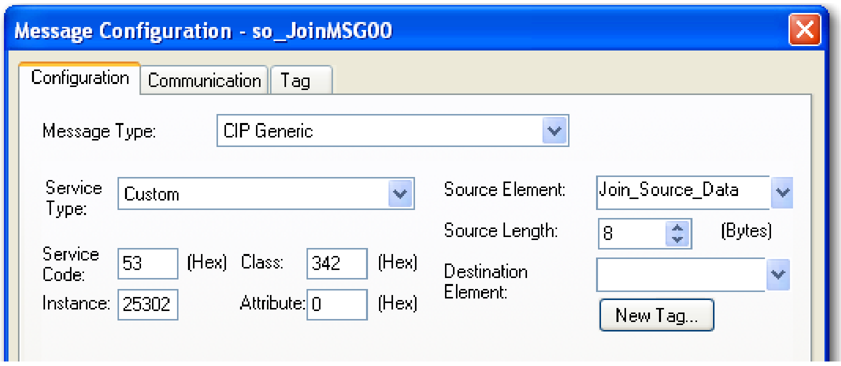
Not used.

JoinMulticastAddress

Joining a multicast group lets a socket receive multicast data. When a join is executed, it sends an IGMP membership packet and enables the hardware filters to receive the multicast data. A specific address can be joined only once. Subsequent joins receive an error message until the multicast address is dropped. Multicast joins are system wide. Two sockets cannot join the same multicast address at the same time. When the socket that the join was executed on is deleted, the multicast address is dropped. Each socket can join one or more multicast groups.

Choose Custom for the service type. JoinMulticastAddress is not an available option from the Service Type pull-down menu.

Parameter	Value
Service Type	Custom
Service Code	53
Class	342
Instance	from Socket Create
Attribute	0


Message Configuration - so_JoinMSG00

Message Type: CIP Generic

Service Type: Custom

Service Code: 53 (Hex) Class: 342 (Hex)

Instance: 25302 Attribute: 0 (Hex)

Source Element: Join_Source_Data

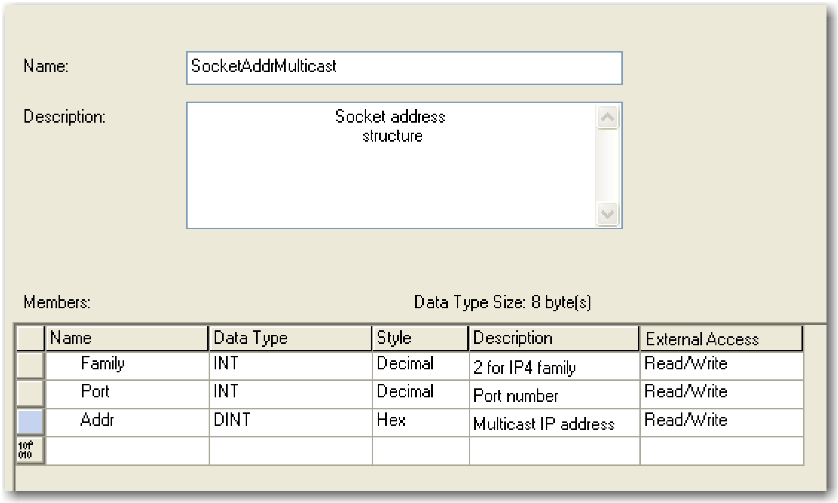
Source Length: 8 (Bytes)

Destination Element:

MSG Source Element

Choose a tag with a user-defined data type. Use the information in Table 10 to define the data type.

Populate the Join_Source_Data.Addr field with a multicast IP address in hexadecimal format. The value must be a hexidecimal representation of the IP address. For example, for address 239.1.2.100, enter 16#EF010264.


Name: SocketAddrMulticast

Description: Socket address structure

Members: Data Type Size: 8 byte(s)

Name	Data Type	Style	Description	External Access
Family	INT	Decimal	2 for IP4 family	Read/Write
Port	INT	Decimal	Port number	Read/Write
Addr	DINT	Hex	Multicast IP address	Read/Write

Table 10 - Data Type for JoinMulticastAddress Source Element

Member Name	Data Type	Description
SocketsAddr	structure	A user-defined structure that specifies the multicast address to join.
Family	INT	Specify the address family. Must be 2.
Port	INT	Not used. The port is determined when the socket is created.
Addr	DINT	Specify the multicast IP address to receive from.

MSG Source Length

Specify 8 bytes.

MSG Destination Element

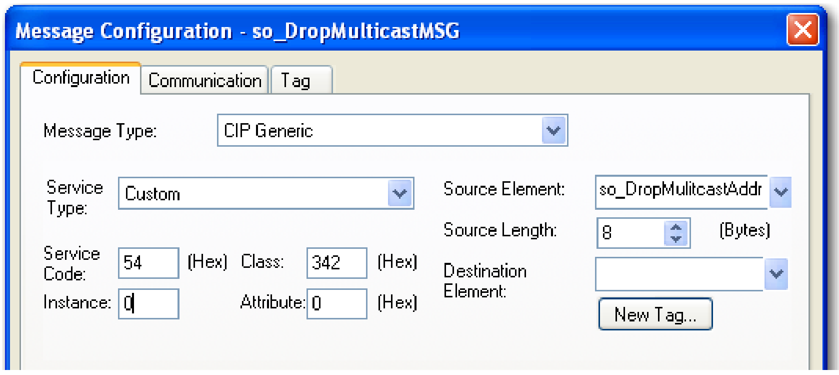
Not used.

DropMulticastAddress

Dropping a multicast address disables a socket from receiving multicast data. When a drop is executed, it sends an IGMP leave group packet and disables the hardware filters from receiving the multicast data.

Choose Custom for the service type. DropMulticastAddress is not an available option from the Service Type pull-down menu.

Parameter	Value
Service Type	Custom
Service Code	54
Class	342
Instance	0
Attribute	0


Message Configuration - so_DropMulticastMSG

Configuration | Communication | Tag

Message Type: CIP Generic

Service Type: Custom

Service Code: 54 (Hex) Class: 342 (Hex)

Instance: 0 Attribute: 0 (Hex)

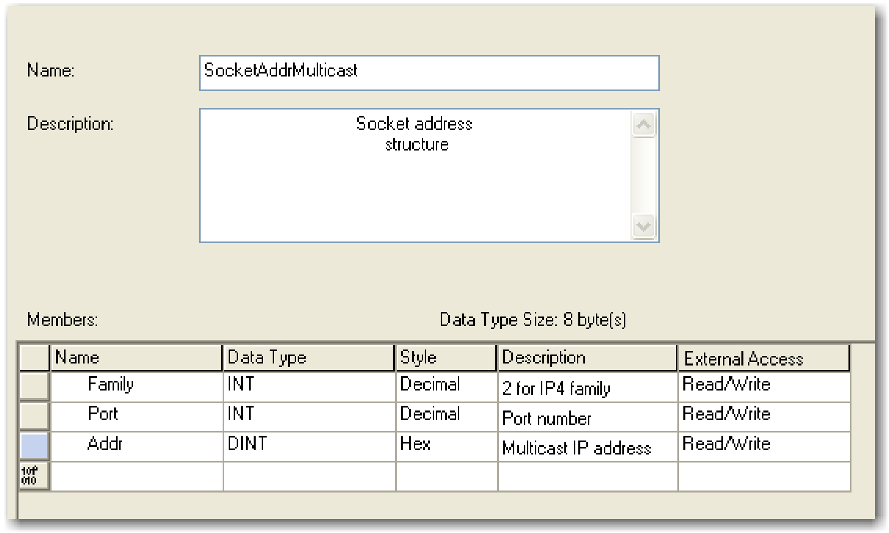
Source Element: so_DropMulitcastAddr

Source Length: 8 (Bytes)

Destination Element:

MSG Source Element

Choose a tag with a user-defined data type. Use the information in Table 11 to define the data type.



Name: SocketAddrMulticast

Description: Socket address structure

Members: Data Type Size: 8 byte(s)

Name	Data Type	Style	Description	External Access
Family	INT	Decimal	2 for IP4 family	Read/Write
Port	INT	Decimal	Port number	Read/Write
Addr	DINT	Hex	Multicast IP address	Read/Write

Table 11 - Data Type for DropMulticast Address Source Element

Member Name	Data Type	Description
SocketsAddr	structure	A user-defined structure that specifies the multicast address to drop.
Family	INT	Specify the address family. Must be 2.
Port	INT	Not used. The port is determined when the socket is created.
Addr	DINT	Specify the multicast IP address to drop.

MSG Source Length

Specify 8 bytes.

MSG Destination Element

Not used.

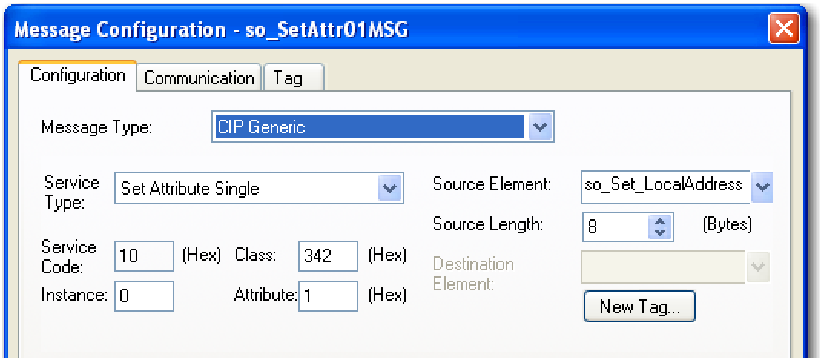
Notes:

Socket Attributes

Access Socket Attributes

You access socket attributes by configuring a CIP Generic MSG instruction to get or set the specific attribute:

- To change an attribute value for a socket, choose Set Attribute Single from the Service Type pull-down menu.



Message Configuration - so_SetAttr01MSG

Configuration | Communication | Tag

Message Type: CIP Generic

Service Type: Set Attribute Single

Service Code: 10 (Hex) Class: 342 (Hex)

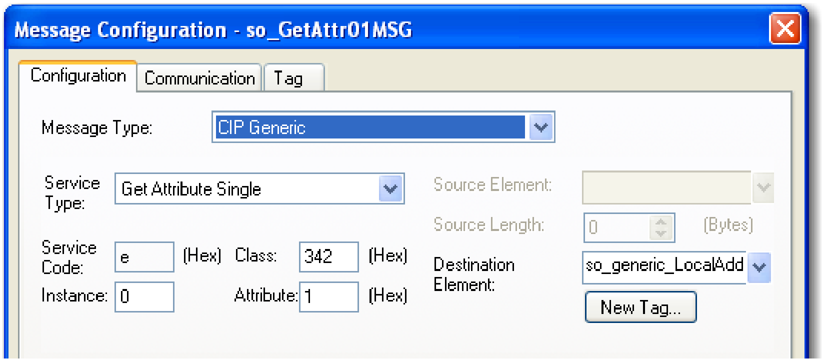
Instance: 0 Attribute: 1 (Hex)

Source Element: so_Set_LocalAddress

Source Length: 8 (Bytes)

Destination Element:

- To get a socket value, choose Get Attribute Single from the Service Type pull-down menu.



Message Configuration - so_GetAttr01MSG

Configuration | Communication | Tag

Message Type: CIP Generic

Service Type: Get Attribute Single

Service Code: e (Hex) Class: 342 (Hex)

Instance: 0 Attribute: 1 (Hex)

Source Element:

Source Length: 0 (Bytes)

Destination Element: so_generic_LocalAdd

Some socket attributes apply to all sockets, and some apply to specific socket instances:

- For information about all sockets, type 0 in the Instance field. See Socket Class Attributes below.
- For information about a specific socket instance, type the specific socket instance number in the Instance field. The instance number is returned by a Socket Create or AcceptConnection service. See Socket Instance Attributes on page 43.

Socket Class Attributes

Class attributes apply to the socket object, not to specific socket instances. When you get or set a Class attribute, set the instance to 0.

Class Attribute	Name	Data Type	Access	Description
1	Revision	INT	Get	Object revision.
2	Max Instance	INT	Get	Largest socket instance number currently created.
3	Number of Instances	INT	Get	Number of socket instances currently created.
8	Log Enable	DINT	Get Set	Enable (1) or disable (0) logging to the Socket Object Log web page. Each socket service has a corresponding bit: • If enabled, requests for that service request are logged. • If disabled, then requests for that service are not logged. Bit 0: Socket Create requests Bit 1: OpenConnection requests Bit 2: AcceptConnection requests Bit 3: Read requests Bit 4: Write requests Bit 5: DeleteSocket and DeleteAllSockets requests Bit 6: Get / Set Attribute requests Bit 7: Log all service errors

If you use the Get Attributes All service to get class attributes, the response contains all of the class attributes in the table above in the order shown with a total size of 10 bytes.

If you use the Set Attributes All service to set class attributes, the request contains only the Log Enable class attribute.

Socket Instance Attributes

The socket object provides a number of instance attributes that apply to specific socket instances. To get or set an instance attribute, specify a valid instance number.

Instance Attribute	Name	Data Type	Access	Description
1	LocalAddr	Struct SockAddr	Get	Local address for the socket.
2	RemoteAddr	Struct SockAddr	Get	Remote address for the socket.
3	SendBufSize	DINT	Get Set	Size of the socket send buffer (bytes).
4	RecvBufSize	DINT	Get Set	Size of the socket receive buffer (bytes).
5	TCPKeepAlive	DINT	Get Set	Enable (1) or disable (0) TCP Keep Alive for the socket. Enabled by default.
6	TCPNoDelay	DINT	Get Set	Enable (1) or disable (0) the TCP No Delay behavior. Enabled by default.
7	InactivityTimeout	DINT	Get Set	Time for the inactivity timeout (default of 5 minutes). If a socket instance receives no service requests for the amount of time specified by the inactivity timeout, the socket instance is deleted. If you then try to use the socket instance, the MSG instruction receives the error Class or instance not supported.
8	MulticastTTL	DINT	Get Set	Set the TTL value for UDP multicast, transmitted packets.
9	UDPBroadcast	DINT	Get Set	Enable (1) or disable (0) the ability to transmit broadcast packets on UDP. Disabled by default.
10	LingerOnOff	DINT	Get Set	Specifies whether the socket performs an orderly close (1) or an immediate close (0). Defaults to no linger (immediate close). For TCP sockets, setting linger to 0 results in a TCP RST packet to close the connection. Setting linger to non-zero results in the standard TCP connection close sequence (3-way FIN, FIN-ACK, ACK handshake followed by TIME_WAIT).

If you use the Get Attributes All service to get instance attributes, the response contains all of the attributes in the table above in the order shown with a total size of 36 bytes.

If you use the Set Attributes All service, the request must include attributes 3, 4, 5, 6 and 7 in that order with a total size of 20 bytes.

Notes:

Troubleshoot Socket Applications

Diagnostic Web Pages

To help debug and troubleshoot applications, the socket interface provides a set of web pages:

- For communication modules and controllers, go to Diagnostics > Advanced Diagnostics > Miscellaneous > System Data > Socket Object.
- For web server modules, go to Diagnostics > Advanced Diagnostics.

Web Page	Description
Socket Object Diagnostics	Displays information about each instance: • Instance number • Socket type—client, server, or listen • Local and remote ports and IP addresses • Send and receive buffer sizes • Socket up time and inactivity time • Socket state and last error state
Socket Object Attributes	Displays attribute settings for each instance
Socket Object Logs	Displays a log of service requests with a maximum of 100 log entries: • Service requests made to the socket object • Parameters passed for each service request • Whether the service request was a success or failure You can enable or disable logging for some services by using the Log Enable class attribute. See Socket Class Attributes on page 42.

Debugging Tips

The following table describes tips for debugging problems by category.

Category	Consideration
EtherNet/IP module	Make sure the EtherNet/IP module has a valid IP address. Also, if you communicate with devices on different subnets, configure the EtherNet/IP module with a valid subnet mask and gateway address.
Socket Create service	Make sure the Destination tag is a DINT tag. After creating the socket with the Socket Create service, make sure you use the instance number that the service returns in the subsequent socket services you call.
MSG instruction	Make sure the Source Element is of a type that matches the request parameters for the requested service. Also make sure the Source Length is the correct length for the service parameters. There is a limit to the number of active MSG instructions in a Logix5000 controller. If a MSG instruction is enabled and exceeds the maximum number of active MSG instructions, the MSG instruction receives an error (.ER bit set).
OpenConnection service	Make sure the Source Length includes the size of the Timeout parameter + Address Length parameter + the Length of the address itself.
Service Timeout parameter	Make sure the Timeout parameter is sufficient for the service. Also make sure the Timeout parameter is less than the MSG instruction timeout. If the timeout set to 0, the service returns immediately.
TCP protocol	Your program should handle the possibility of loss of TCP connections. A TCP connection is a byte stream with no inherent message boundaries. The application defines how to interpret message boundaries. For example, the application might use a fixed length for all messages. For a variable-length message, the application might use a fixed-length header that contains the length of the remainder of the message. Both ends of the TCP connection must agree on the application protocol that is used.
Ethernet sniffer	An Ethernet sniffer is useful to monitor the messages between the EtherNet/IP module and other devices. You can capture network traffic and set up filters to isolate messages between particular devices and particular messages between those devices.

Error Codes for Socket Services

If a socket object encounters an error with a service request, the following occurs:

- Socket object returns an error code.
- MSG instruction sets the .ER bit.
- MSG instruction sets error codes in the Error (.ERR) and Extended Error (.EXTERR) fields.

The table below describes common error codes. For more a comprehensive list of error codes, search the Knowledgebase for answer ID 34290.

Error Code	Extended Error Code	Description
16#0009	—	Invalid socket descriptor. To resolve this error, do the following: • Make sure a valid socket instance exists. • Make sure the message source data format and source values are correct.
16#00ff	16#0000_0030	The address is already in use. This error can occur when multiple Socket Create requests are issued to the same port address.
16#00ff	16#0000_0036	A connection was forced closed by a peer. This error can occur when a remote device closes a connection with a Logix module without notifying the module. To resolve this error, try deleting the socket and then reconnecting to the remote device.
16#00ff	16#0000_0041	A socket operation could not find a route to the remote host. This error typically occurs in these scenarios: • A remote IP address specified in the Message instruction is not on the same subnet as the Logix module. and The IP address of the gateway or router is not specified in the Logix module's properties. • UDP multicast messages to an unpingable IP address require you to specify a gateway address in the Logix module's properties even if a gateway address does not exist or is not required. For more information, search the Knowledgebase for answer ID 34358.
16#00ff	16#0000_0043	The remote device or gateway is not responding. This error can occur if a UDP multicast message is sent to a gateway address that is not specified in the Logix module's properties. For more information, search the Knowledgebase for answer ID 34358.
16#00ff	16#0000_0046	The socket operation timed out. This error can occur when read and write messages for the same socket are active at the same time.
16#0005	16#0000_0000 or 16#0000_0001	The socket instance does not exist. This error can occur in these scenarios: • The socket instance number returned by the Socket Create service does not match the instance number in the socket read or write message. • The socket instance closed due to inactivity. • The socket was deleted by the DeleteSocket service.

Knowledgebase Articles

Search the Knowledgebase by using the answer IDs in the table below to find additional help. To access the Knowledgebase, log on to the Rockwell Automation Support Center at http://rockwellautomation.custhelp.com/.

Answer ID	Description
32962	Sample applications for the 1756-EWEB module.
34290	Descriptions of possible socket error codes.
40626	Helpful hints for socket services.
48879	Summary of major sockets topics and functionality descriptions.
50122	Using sockets in AOIs (Add-on Instructions).

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://www.rockwellautomation.com/support, you can find technical manuals, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools. You can also visit our Knowledgebase at http://www.rockwellautomation.com/knowledgebase for FAQs, technical information, support chat and forums, software updates, and to sign up for product notification updates.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect^{SM} support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://www.rockwellautomation.com/support/.

Installation Assistance

If you experience a problem within the first 24 hours of installation, review the information that is contained in this manual. You can contact Customer Support for initial help in getting your product up and running.

United States or Canada	1.440.646.3434
Outside United States or Canada	Use the Worldwide Locator at http://www.rockwellautomation.com/support/americas/phone_en.html, or contact your local Rockwell Automation representative.

New Product Satisfaction Return

Rockwell Automation tests all of its products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning and needs to be returned, follow these procedures.

United States	Contact your distributor. You must provide a Customer Support case number (call the phone number above to obtain one) to your distributor to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for the return procedure.

Documentation Feedback

Your comments will help us serve your documentation needs better. If you have any suggestions on how to improve this document, complete this form, publication RA-DU002, available at http://www.rockwellautomation.com/literature/.

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846