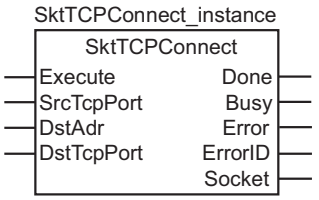
SktTCPConnect

The SktTCPConnect instruction connects to a remote TCP port from the EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPConnect	Connect TCP Socket	FB		SktTCPConnect_instance(Execute, SrcTcpPort, DstAdr, DstTcpPort, Done, Busy, Error, ErrorID, Socket);

Variables

	Meaning	I/O	Description	Valid range	Unit	Default
SrcTcpPort	Local TCP port number.	Input	Local TCP port number. If 0 is specified, an available TCP port that is 1024 or higher is automatically assigned. Well-known port numbers are not assigned.	Depends on data type.	---	0
DstAdr	Destination address		Destination IP address or host name	200 bytes max.		---
DstTcpPort	Destination TCP port number		Destination TCP port number	1 to 65,535		1
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							OK													
DstAdr																				OK
DstTcpPort							OK													
Socket	Refer to *Function* on page 2-1189 for details on the structure _sSOCKET.																			

Function

The SktTCPConnect instruction requests a connection between local TCP port number *SrcTcpPort* and destination TCP port number *DstTcpPort* at destination address *DstAdr*. To do this, it executes the Connect() socket function.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

The connection is established when the instruction is completed normally.

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr	Local address	Local IP address and port number	_sSOCK-ET_AD-DRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*1	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''
DstAdr	Destination address	Destination IP address and port number	_sSOCK-ET_AD-DRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''

*1. NULL is output for this member.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2			
_EIP2_EtnOnlineSta*3			

*1. Use this variable name for an NJ-series CPU Unit.
*2. Use this variable name for port 1 on an NX-series CPU Unit.
You can specify _EIP_EtnOnlineSta instead of _EIP1_EtnOnlineSta.
*3. Use this variable name for port 2 on an NX-series CPU Unit.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* for details on the socket service functions.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.

Or you can use this instruction through a port on an NX-series EtherNet/IP Unit by setting **IP Forward** to **Use** in the NX502 CPU Unit.

- You cannot use this instruction in an event task. A compiling error will occur.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- Except for NX502 CPU Units and NX102 CPU Units, you can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, SktSetOption, ModbusTCPCmd, ModbusTCPRead, and ModbusTCPWrite.
  For NX502 CPU Units and NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  a) There is a setting error for the local IP address.
  b) The value of *DstAdr* is outside of the valid range.
  c) The value of *DstTcpPort* is outside of the valid range.
  d) The TCP port that is specified with *SrcTcpPort* is already open.
  e) The remote node that is specified with *DstAdr* does not exist.
  f) The remote node that is specified with *DstAdr* and *DstTcpPort* is not waiting for a connection.
  g) Address resolution failed for the host name that is specified with *DstAdr*.
  h) A connection is already open for the same client (IP address and TCP port).

Version Information

- The number of sockets that you can open at the same time depends on the unit version of the CPU Unit as shown in the following table. These limits are the totals for both UDP and TCP sockets.

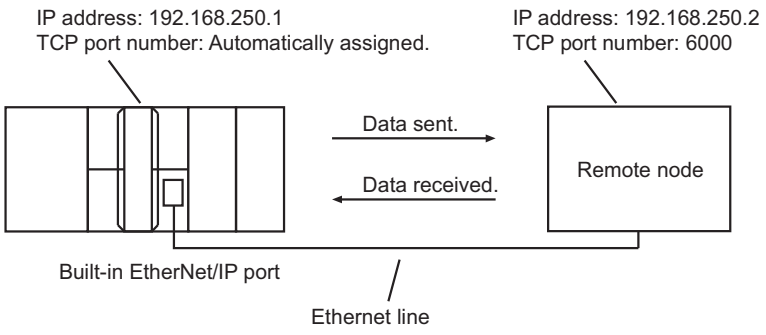
Unit version of CPU Unit	Number of sockets
1.03 or later	30 max.*1
1.02 or earlier	16 max.

*1. For NX502 CPU Units and NX102 CPU Units, you can execute a maximum of 60 sockets.

- For CPU Unit version 1.10 or later, the value of *Socket* does not change even if *Error* changes to TRUE. For version 1.09 or earlier, the value of *Socket* changes to *0*.

Sample Programming

In this sample, the TCP socket service is used for data communications between the NJ/NX-series CPU Unit and a remote node.



User program of NJ/NX-series CPU Unit

The processing procedure is as follows:

1. The SktTCPConnect instruction is used to request connecting to the TCP port on the remote node.
2. The SktClearBuf instruction is used to clear the receive buffer for a TCP socket.
3. The SktGetTCPStatus instruction is used to read the status of a TCP socket.
4. The SktTCPSend instruction is used to request sending data. The data in SendSocketDat[] is sent.
5. The SktTCPRcv instruction is used to request receiving data. The received data is stored in RcvSocketDat[].
6. The SktClose instruction is used to close the socket.

- **ST**

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoTCP	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktTCPConnect_instance	SktTCPConnect		
	SktClearBuf_instance	SktClearBuf		
	SktGetTCPStatus_instance	SktGetTCPStatus		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	✉	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage:=INT#1;
    SktTCPConnect_instance(Execute:=FALSE);  // Initialize instance.
    SktClearBuf_instance(Execute:=FALSE);  // Initialize instance.
    SktGetTCPStatus_instance(Execute:=FALSE);  // Initialize instance.
    SktTCPSend_instance(  // Initialize instance.
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);  // Dummy
    SktTCPRcv_instance(  // Initialize instance.
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);  // Dummy
    SktClose_instance(Execute:=FALSE);  // Initialize instance.
END_IF;

IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 :  // Request a connection.
        SktTCPConnect_instance(
            Execute   :=TRUE,
            SrcTcpPort:=UINT#0,  // Local TCP port number: Automatically assigned.
            DstAdr    :='192.168.250.2',  // Remote IP address
            DstTcpPort:=UINT#6000,  // Destination TCP port number
            Socket    =>WkSocket);  // Socket

        IF (SktTCPConnect_instance.Done=TRUE) THEN
            Stage:=INT#2;  // Normal end
        ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
            Stage:=INT#10;  // Error end
        END_IF;
    2 :  // Clear receive buffer.
        SktClearBuf_instance(
            Execute:=TRUE,
            Socket :=WkSocket);  // Socket

        IF (SktClearBuf_instance.Done=TRUE) THEN
            Stage:=INT#3;  // Normal end
        ELSIF (SktClearBuf_instance.Error=TRUE) THEN
            Stage:=INT#20;  // Error end
        END_IF;

    3 :  // Request reading status.
        SktGetTCPStatus_instance(
            Execute:=TRUE,
            Socket :=WkSocket);  // Socket
```

```
    IF (SktGetTCPStatus_instance.Done=TRUE) THEN
        Stage:=INT#4;  // Normal end
    ELSIF (SktGetTCPStatus_instance.Error=TRUE) THEN
        Stage:=INT#30;  // Error end
    END_IF;

4 :  // Request sending data
    SktTCPSend_instance(
        Execute:=TRUE,
        Socket :=WkSocket,  // Socket
        SendDat:=SendSocketDat[0],  // Send data
        Size   :=UINT#2000);  // Send data size

    IF (SktTCPSend_instance.Done=TRUE) THEN
        Stage:=INT#5;  // Normal end
    ELSIF (SktTCPSend_instance.Error=TRUE) THEN
        Stage:=INT#40;  // Error end
    END_IF;

5 :  // Request receiving data
    SktTCPRcv_instance(
        Execute:=TRUE,
        Socket :=WkSocket,  // Socket
        TimeOut:=UINT#0,  // Timeout time
        Size   :=UINT#2000,  // Receive data size
        RcvDat :=RcvSocketDat[0]);  // Receive data

    IF (SktTCPRcv_instance.Done=TRUE) THEN
        Stage:=INT#6;  // Normal end
    ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
        Stage:=INT#50;  // Error end
    END_IF;

6 :  // Request closing.
    SktClose_instance(
        Execute:=TRUE,
        Socket :=WkSocket);  // Socket

    IF (SktClose_instance.Done=TRUE) THEN
        Stage:=INT#0;  // Normal end
    ELSIF (SktClose_instance.Error=TRUE) THEN
        Stage:=INT#40;  // Error end
    END_IF;
0 :  // Normal end
    DoTCP  :=FALSE;
    Trigger:=FALSE;
```

```
    ELSE  // Interrupted by error.
        DoTCP  :=FALSE;
        Trigger:=FALSE;
    END_CASE;

END_IF;
```

Programming in the Remote Node

In this example, programming is also required in the remote node. The order of sending and receiving is reversed in comparison with the above procedure.

1 The SktTCPAccept instruction is used to request accepting a TCP socket.

2 The SktTCPRcv instruction is used to request receiving data. The received data is stored in RcvSocketDat[].

3 The SktTCPSend instruction is used to request sending data. The data in SendSocketDat[] is sent.

4 The SktClose instruction is used to close the socket.

- **ST**

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoTCP	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktTCPAccept_instance	SktTCPAccept		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	☑	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage:=INT#1;
    SktTCPAccept_instance(Execute:=FALSE);   // Initialize instance.
    SktTCPSend_instance(                     // Initialize instance.
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);          // Dummy
    SktTCPRcv_instance(                      // Initialize instance.
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);           // Dummy
    SktClose_instance(Execute:=FALSE);       // Initialize instance.
END_IF;

IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 :                                 // Request accepting a socket connection.
        SktTCPAccept_instance(
            Execute   :=TRUE,
            SrcTcpPort:=UINT#6000,           // Local TCP port number
            TimeOut   :=UINT#0,              // Timeout time
            Socket    =>WkSocket);           // Socket

        IF (SktTCPAccept_instance.Done=TRUE) THEN
            Stage:=INT#2;                     // Normal end
        ELSIF (SktTCPAccept_instance.Error=TRUE) THEN
            Stage:=INT#10;                    // Error end
        END_IF;
    2 :                                 // Request receiving data
        SktTCPRcv_instance(
            Execute:=TRUE,
            Socket :=WkSocket,               // Socket
            TimeOut:=UINT#0,                 // Timeout time
            Size   :=UINT#2000,              // Receive data size
            RcvDat :=RcvSocketDat[0]);       // Receive data

        IF (SktTCPRcv_instance.Done=TRUE) THEN
            Stage:=INT#3;                     // Normal end
        ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
            Stage:=INT#20;                    // Error end
        END_IF;
```

```
    3 :                                 // Request sending data.
        SendSocketDat:=RcvSocketDat;
        SktTCPSend_instance(
            Execute:=TRUE,
            Socket :=WkSocket,                  // Socket
            SendDat:=SendSocketDat[0],          // Send data
            Size   :=UINT#2000);                // Send data size

        IF (SktTCPSend_instance.Done=TRUE) THEN
            Stage:=INT#4;                       // Normal end
        ELSIF (SktTCPSend_instance.Error=TRUE) THEN
            Stage:=INT#30;                      // Error end
        END_IF;

    4 :                                 // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);                 // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                       // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                      // Error end
        END_IF;

    0 :                                 // Normal end
        DoTCP  :=FALSE;
        Trigger:=FALSE;

    ELSE                                // Interrupted by error.
        DoTCP  :=FALSE;
        Trigger:=FALSE;
    END_CASE;

END_IF;
```