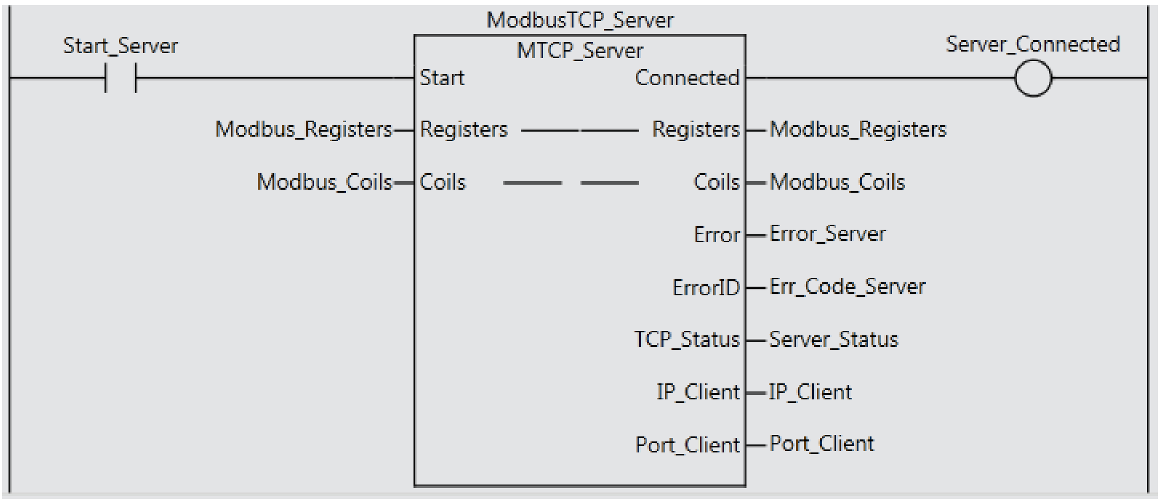
BlocFonction

Référence	**MTCP_NJ_Server**
Révision	**1.9**
Auteur	**JP Viskovic**
Date	**16/08/2016**
+ Support	http://support-omron.fr/

OMRON ELECTRONICS S.A.S.
14 Rue de Lisbonne
93561 Rosny-sous-Bois cedex

N° Indigo 0 825 825 679
0.15€ TTC/mn

Modbus TCP Server pour contrôleur NJ

Fonction	Serveur Modbus TCP sur port Ethernet de contrôleur NJ
Serveur	 ModbusTCP_Server MTCP_Server Start_Server Server_Connected Start Connected Modbus_Registers Registers Registers Modbus_Registers Modbus_Coils Coils Coils Modbus_Coils Error Error_Server ErrorID Err_Code_Server TCP_Status Server_Status IP_Client IP_Client Port_Client Port_Client
Fichier	MTCP_NJ.zip

Conditions d'utilisation	Le bloc fonction MTCP_NJ_Server propose certaines fonctionnalités Modbus conformément aux spécifications définies par l'organisation Modbus. Le bloc fonction MTCP_NJ_Server est proposé 'tel que' et peut servir de base de développement. Les utilisateurs doivent, au préalable, tester son adéquation avec l'application finale. **Omron France ne pourra en aucun cas être tenu pour responsable en cas de dysfonctionnement de l'application finale.**
Principe	Le FB MTCP_NJ_Server attend la connexion d'un client Modbus TCP lorsque l'entrée *Start* est activée. Il est recommandé de placer le FB dans une tâche périodique de manière à ne pas surcharger le programme principal exécuté dans la tâche primaire. Liste des commandes supportées : Code / Modbus Function: 0x01 Read Coils; 0x02 Read Discret Inputs; 0x03 Read Holding Registers; 0x04 Read Input Registers; 0x05 Write Single Coil; 0x06 Write Single Register; 0x10 Write Multiple Registers

Code	Modbus Function
0x01	Read Coils
0x02	Read Discret Inputs
0x03	Read Holding Registers
0x04	Read Input Registers
0x05	Write Single Coil
0x06	Write Single Register
0x10	Write Multiple Registers

1- Variable d’entrée/sortie du bloc MTCP_NJ_Server

Variables d’entrée

Nom	type	plage	Description
Start	Bool	OFF, ON	ON : Activation du serveur

Variables d’entrée/sortie

Nom	type	plage	Description
Registers	Tableau de 1024 mots	0-FFFF	Zone des registres
Coils	Tableau de 1024 booléens	OFF-ON	Zone des bobines

Variables de sortie

Nom	type	plage	Description
Connected	Bool	OFF, ON	ON : client connecté au serveur
Error	Bool	OFF, ON	Drapeau d’erreur
ErrorID	UINT	0 - FFFF	Code erreur renvoyé par le socket ou le serveur Modbus TCP(voir liste plus bas)
TCP_Status	_eCONNECTION_STATE	Enum	_CLOSED _LISTEN _SYN SENT _SYN RECEIVED _ESTABLISHED _CLOSE_WAIT _FIN WAIT1 _CLOSING _LAST ACK _FIN WAIT2 _TIME WAIT
IP_Client	String[24]	w.x.y.z	Adresse IP du client connecté
Port_Client	UINT	0-65535	N° de port du client connecté

Code erreur renvoyé ErrorID

Code		Description
0001	Modbus Exception	ILLEGAL FUNCTION
0002		ILLEGAL DATA ADDRESS
0003		ILLEGAL DATA VALUE
2000	Socket error	Local IP Address Setting Error
2001		TCP/UDP Port Already in Use
2002		Address Resolution Failed
2003		Status Error
2004		Local IP Address Not Set
2006		Socket Timeout
2007		Socket Handle Out of Range
2008		Socket Communications Resource Overflow

Precautions in Using Socket Services

Precautions for UDP and TCP Socket Services

• Communications processing are sometimes delayed when multiple functions of the built-in EtherNet/IP port are used simultaneously or due to the contents of the user program.
• Communications efficiency is sometimes reduced by high communications traffic on the network line.
• The close processing for a close request instruction discards all of the buffered send and receive data for the socket. For example, send data from a send request instruction immediately before the close processing is sometimes not sent.
• After a socket is open, the built-in EtherNet/IP port provides a receive buffer of 9,000 bytes per TCP socket and 9,000 bytes per UDP socket to enable data to be received at any time. If the receive buffer is full, data received by that socket is discarded. Make sure that the user application always executes receive requests to prevent the internal buffer from becoming full.

Precautions for UDP Socket Services

• The destination IP address can be set to a broadcast address for a UDP socket to broadcast data to all nodes on the network. However, in this case, the maximum length of send data is 1,472 bytes. Data lengths broken into multiple fragments (1,473 bytes or more in UDP) cannot be sent.
• For UDP socket, controls to confirm the reliability of communications, such as the confirmation of send data, are not performed. To improve the reliability of communications when you use UDP sockets, make sure the user program confirms that data is sent and resends data when necessary.

Precautions for TCP Socket Services

• If the TCP socket is closed on the remote node without warning during communications (i.e., if the connection is closed), the socket at the local node must also be closed. You can use the Read TCP Socket Status instruction (SktGetTCPstatus) to see if the connection is closed. Immediately close the socket at the local node if the TCP socket at the remote node is closed.
• If the remote node’s TCP socket closes without warning, the data to send may remain in the buffer at the local node. The remaining data is discarded in the local node’s TCP close processing. The steps that are required in applications to avoid this include sending data from the sending node that permits closing and closing the socket only after checking the remote node.
• While open processing is performed for a TCP socket, a port that was closed first cannot be opened again for 60 seconds from the time the close processing is performed for the remote socket. However, this is not true if you specified 0 (automatic assignment by the Unit) as the port for the SktTCPConnect instruction.
• You can use *Connect* from another socket to open a connection to a socket that was opened with *Accept*. A connection is not opened if you try to use *Connect* from another socket to open a connection to a socket that was opened with *Connect*. Also, a connection is not opened if you attempt to use *Accept* from another socket to open a socket that was opened with *Accept*. Furthermore, you cannot use *Connect* from more than one other node to establish multiple connections with a single TCP socket that was opened with *Accept* on the built-in EtherNet/IP port.
• You can use the keep-alive function for TCP sockets at the built-in EtherNet/IP port. The keep alive function checks whether a connection is normally established when no data is sent or received for a certain period on the communications line where the connection was established. The built-in EtherNet/IP port responds to checks from other nodes even if keep alive is not specified.