APN 048
JetSym STX Functions



Application Note

Jetter

60879851

Application Note: 048
Item # 60879851
Revision 1.00
August 2014 / Printed in Germany

This document has been compiled by Jetter AG with due diligence, and based on the known state of the art.

In the case of modifications, further developments or enhancements to products shipped in the past, a revised document will be supplied only if required by law, or deemed appropriate by Jetter AG. Jetter AG shall not be liable for errors in form or content, or for missing updates, as well as for damages or disadvantages resulting from such failure.

The logos, brand names, and product names mentioned in this document are trade marks or registered trade marks of Jetter AG, of associated companies or other title owners and must not be used without consent of the respective title owner.

Table of Contents

1 JetSym STX functions 5

1.1 Format conversion ... 6
BCD to binary conversion ... 7
Binary to BCD ... 8
1.2 Sorting data ... 9
Function description ... 10
Sample program ... 12
1.3 Modbus RTU CRC checksum ... 14
Calculating the CRC ... 15
Checking the CRC ... 17
1.4 Modbus/TCP client with STX variables ... 18
ModbusReadReg(): Acyclical reading of registers ... 20
ModbusWriteReg(): Acyclical writing of registers ... 22
Sample program: Acyclical communication with the Modbus/TCP server ... 24
RemoteScanConfig(): Configuring RemoteScan ... 26
RemoteScanStart(): Starting RemoteScan ... 30
RemoteScanStop(): Stopping RemoteScan ... 31
Sample program: Cyclic communication with the Modbus/TCP server ... 32
1.5 Data files ... 34
Writing a data file ... 35
1.6 Sending an e-mail ... 37
Sending an e-mail ... 38
1.7 NetCopyList ... 40
Programming the NetCopyList functions ... 41
Configuring a list ... 43
Sending a list ... 45
Deleting a list ... 46
Configuring, sending and deleting a list ... 47
Sample program: NetCopyList ... 49

1 JetSym STX functions

Introduction

This application note covers special STX functions. They correspond to the system functions. While system functions only use registers, the STX functions described below only use both localized and not localized STX variables.

Contents

Topic	Page
Format conversion	6
Sorting data	9
Modbus RTU CRC checksum	14
Modbus/TCP client with STX variables	18
Data files	34
Sending an e-mail	37
NetCopyList	40

1.1 Format conversion

Introduction

This chapter covers the STX functions for converting formats from BCD to binary and vice versa.

Contents

Topic	Page
BCD to binary conversion	7
Binary to BCD	8

BCD to binary conversion

Introduction

You may use this function, for example, to convert the multi-place notation of BCD switches to the internal numeric notation (binary). For this, the BCD switch is connected to digital inputs.

Function declaration

```
Function Bcd2Hex(Bcd: Int): Int;
```

Parameter	Description
Bcd	Value in BCD notation

Return value

The returned value is the numeric value in internal notation (binary).

JetSym STX program

- To the first JX3-DIO16 module at a JetControl 340, a 4-digit BCD switch has been connected.
- The units place is connected to inputs 1 through 4, the tens place is connected to inputs 5 through 8, etc.
- Access to the switch is made via register 100004062 which is overlaid on the inputs.
- The setting of the switch is 5432.

```
Var
    rInputs:    Int At %VL 100004062;
    nBCDValue: Int;
    nValue:     Int;
End_Var;

Task tMain Autorun
    ...

    nBCDValue := rInputs;
    nValue    := Bcd2Hex(nBCDValue);

    ...
End_Task;
```

Variable	Decimal value	Hexadecimal value
nBCDValue	21554	0x5432
nValue	5432	0x1538

Binary to BCD

Introduction

You may use these variables, for example, to convert a numeric value for output on a BCD display. For this, the BCD display is connected to digital outputs.

Function declaration

```
Function Hex2Bcd(Hex: Int): Int;
```

Parameter	Description
Hex	Value in internal notatation

Return value

The returned value is the numeric value in BCD notation. As a number of the "Int" type can only be assigned eight BCD numbers, the higher-order half bytes are left out if there is an overflow of figures.

JetSym STX program

- To the first JX3-DO16 module at a JetControl 340, a 4-digit BCD display has been connected.
- The units place is connected to outputs 1 through 4, the tens place is connected to outputs 5 through 8, etc.
- Access to the switch is made via register 100004262 which is overlaid on the outputs.
- The value to be output is 5432.

```
Var
    rOutputs:    Int At %VL 100004262;
    nBCDValue:    Int;
    nValue:      Int;
End_Var;

Task tMain Autorun
    ...

    nBCDValue := Hex2Bcd(nValue);
    rOutputs  := nBCDValue;

    ...
End_Task;
```

Variable	Decimal value	Hexadecimal value
nValue	5432	0x1538
nBCDValue	21554	0x5432

1.2 Sorting data

Introduction

This chapter describes the STX function for sorting data by assigning them to variables.

Contents

Topic	Page
Function description	10
Sample program	12

Function description

Introduction

To offer the user the greatest possible flexibility, the sort algorithm is separated from the data. The sort algorithm is provided by the operating system of the controller. The pointer pointing to the data to be sorted and the sorting parameters are specified at the function call.

Function declaration

```
Function QSort(DataPtr: Int, ElementCnt: Int, ElementSize: Int,
SortOffset: Int, SortType: STXBASETYPE, SortMode: QSORTMODE): Int;
```

Parameter	Description
DataPtr	Start address of the range of variables to be sorted
ElementCnt	Amount of variables to be sorted
ElementSize	Size of one of the variables to be sorted
SortOffset	Address offset within a variable containing the sorting value
SortType	Type of the variable at "SortOffset"
SortMode	Sorting mode

Type declaration

```
Type
    STXBASETYPE: Enum (
        TYPECHAR    = 0,
        TYPEBYTE    = 1,
        TYPEINT8    = 2,
        TYPESMALL   = 3,
        TYPEWORD    = 4,
        TYPEUSINT   = 5,
        TYPEINT16   = 6,
        TYPESINT    = 7,
        TYPESHORT   = 8,
        TYPEINT32   = 9,
        TYPELONG    = 10,
        TYPEINT     = 11,
        TYPEFLOAT   = 12,
        TYPEREAL    = 13,
        TYPEDOUBLE  = 14,
        TYPELREAL   = 15
    );

    QSORTMODE: Enum (
        ASCENDING   = 0,
        DESCENDING  = 1
    );
End_Type;
```

Function parameters

Parameter	Value	Comment
DataPtr	Valid variable address	Direct input
ElementCnt	2 ... 1000	Amount of variables to be sorted
ElementSize	1 ... 1000	Size of one of the variables to be sorted in bytes
SortOffset	0 ... ("ElementSize" - 1)	Address offset within a variable containing the sorting value
SortType	0 ... 15	Type of the variable at "SortOffset"
SortMode	0 ... 1	Sorting mode "ascending order" or "descending order"

Return value

The following return values are possible:

Return value	
> 0	Execution time [µs]
-3	"ElementCnt" smaller 2 or greater 1,000
-4	"ElementSize" smaller 1 or greater 1,000
-5	"SortMode" invalid
-6	"SortOffset" greater "ElementSize" - 1
-7	"SortType" greater 15
-8	"DataPtr" invalid

Sample program

Task

The data records stored in an array are to be sorted in ascending order based on the second element.

Solution

Use the STX function QSort() for the sorting operation.

JetSym STX program

```
#define SORTSIZE    4

Type
    SORTTYPE:    Struct
                    wData1:    Word;
                    nData1:    Int;
                    rData1:    Real;
                End_Struct;
End_Type;

Var
    aSort:          Array[SORTSIZE] Of SORTTYPE;
    nSortIdx:      Int;
    nSortType:     STXBASETYPE;
    nSortMode:     QSORTMODE;
    nResult:        Int;
End_Var;

Task tMain Autorun
    ...
    // Determining the offset of the second data record element.
    nSortIdx  := &aSort[0].nData1 - &aSort;
    // Defining the variable type
    nSortType := STXBASETYPE.TYPEINT;
    // Defining the sorting mode
    nSortMode := QSORTMODE.ASCENDING;
    // Sorting the data records
    nResult := QSort(&aSort, SORTSIZE, sizeof(SORTTYPE),
                     nSortIdx, nSortType, nSortMode);

    ...
End_Task;
```

Data records before the sorting process

In this setup window of JetSym, you can see the data records before sorting.

	Name	Number	Content	Type	Comment
3	nSortIdx		4		
4	nSortType		TYPEINT (11)		
5	nSortMode		ASCENDING (0)		
6	aSort	1001800	array[4]		
7	aSort[0]	1001800	struct		
8	aSort[0].wData1	1001800	1		
9	aSort[0].nData1	1001801	8		
10	aSort[0].rData1	1001802	15.75000		
11	aSort[1]	1001803	struct		
12	aSort[1].wData1	1001803	2		
13	aSort[1].nData1	1001804	1		
14	aSort[1].rData1	1001805	8.750000		
15	aSort[2]	1001806	struct		
16	aSort[2].wData1	1001806	3		
17	aSort[2].nData1	1001807	4		
18	aSort[2].rData1	1001808	20.89700		
19	aSort[3]	1001809	struct		
20	aSort[3].wData1	1001809	4		
21	aSort[3].nData1	1001810	20		
22	aSort[3].rData1	1001811	6.125000		
23					
24	nResult		0		
25					

Data records after the sorting process

In this setup window of JetSym, you can see the data records after sorting.

	Name	Number	Content	Type	Comment
3	nSortIdx		4		
4	nSortType		TYPEINT (11)		
5	nSortMode		ASCENDING (0)		
6	aSort	1001800	array[4]		
7	aSort[0]	1001800	struct		
8	aSort[0].wData1	1001800	2		
9	aSort[0].nData1	1001801	1		
10	aSort[0].rData1	1001802	8.750000		
11	aSort[1]	1001803	struct		
12	aSort[1].wData1	1001803	3		
13	aSort[1].nData1	1001804	4		
14	aSort[1].rData1	1001805	20.89700		
15	aSort[2]	1001806	struct		
16	aSort[2].wData1	1001806	1		
17	aSort[2].nData1	1001807	8		
18	aSort[2].rData1	1001808	15.75000		
19	aSort[3]	1001809	struct		
20	aSort[3].wData1	1001809	4		
21	aSort[3].nData1	1001810	20		
22	aSort[3].rData1	1001811	6.125000		
23					
24	nResult		4		
25					

1.3 Modbus RTU CRC checksum

Introduction

This chapter describes the STX functions for calculating and checking the Modbus RTU CRC checksum.

Contents

Topic	Page
Calculating the CRC	15
Checking the CRC	17

Calculating the CRC

Introduction

At Modbus RTU communication, data transmission must be confirmed by a 2-byte CRC checksum. This function calculates the CRC and adds the result to the end of the frame to be transmitted.

Function declaration

```
Function ModbusCRCgen(FramePtr: Int, Length: int): Int;
```

Parameter	Description
FramePtr	Address of the frame to be transmitted
Length	Amount of characters in the frame

Important note

The variable containing the frame must be at least 2 bytes greater than the value in the "Length" parameter. The function adds the two bytes for the CRC also to the end of the frame.

Return value

The return value is the CRC checksum which is also added to the frame

JetSym STX program

The frame buffer contains the data of the Modbus RTU frame of a 6-byte length. The function adds the two CRC bytes, so the entire frame of an 8-byte length can then be transmitted via serial interface, for example.

```
Var
    nResult:     Int;
    abyFrame:    Array[20] Of Byte;
    nLen:         Int;
End_Var;

Task tMain Autorun
    ...

    nLen         := 6;
    abyFrame[0] := 0x02;
    abyFrame[1] := 0x03;
    abyFrame[2] := 0x00;
    abyFrame[3] := 0x20;
    abyFrame[4] := 0x00;
    abyFrame[5] := 0x04;
    nResult := ModbusCRCgen(&abyFrame, nLen);

   ...
End_Task;
```

	Name	Number	Content	Type	Comment
12	nResult		0x000045f0		
13	nLen		6		
14	abyFrame		array[20]		
15	abyFrame[0]		0x02		
16	abyFrame[1]		0x03		
17	abyFrame[2]		0x00		
18	abyFrame[3]		0x20		
19	abyFrame[4]		0x00		
20	abyFrame[5]		0x04		
21	abyFrame[6]		0x45		CRC high byte
22	abyFrame[7]		0xf0		CRC low byte

Checking the CRC

Introduction

At Modbus RTU communication, data transmission must be confirmed by a 2-byte CRC checksum. This function checks the CRC in the last two bytes of the frame.

Function declaration

```
Function ModbusCRCcheck(FramePtr: Int, Length: int): Int;
```

Parameter	Description
FramePtr	Address of the frame to be transmitted
Length	Amount of characters in the frame, CRC included

Return value

The following return values are possible:

Return value	
0	CRC is not correct
1	CRC correct

1.4 Modbus/TCP client with STX variables

Introduction

The Modbus/TCP client included in the controller supports only Class 0 conformance.

In this class, commands for reading and writing multiple registers are used. One frame transmits up to 125 registers of 16 bits width.

As protocol ID, "0" is used. Assignment of transmitted and received frames is carried out using the transaction ID.

This chapter describes how to carry out noncyclical or cyclical transmission to a Modbus/TCP server using STX functions.

Amount of possible connections

Connections to eleven different Modbus/TCP servers may be opened at the same time.

Noncyclical data transmission

Functions `ModbusReadReg()` and `ModbusWriteReg()` are used to establish a noncyclical transmission channel to a Modbus/TCP server.

These functions copy data between registers of a Modbus/TCP server and STX variables. They establish a connection to the specified Modbus/TCP server, transmit the desired data and clear down the connection again.

If RemoteScan has already established a connection applying cyclical data transmission, this connection will be used. Setting-up and clearing-down the connection is, therefore, not required.

Cyclical data transmission

Cyclical data transmission is made through the configurable function `RemoteScanConfig()`. The data are cyclically transmitted from and to the Modbus/TCP servers by means of STX variables.

Only one connection is established to each Modbus/TCP server (IP address and port) irrespective of the number of communication units which have been configured on this server.

If several communication units are configured on one Modbus/TCP server, accesses are serialized since servers often do not support "command pipelining". If several servers have been configured, communication is carried out in parallel.

Unit ID

Converters from Modbus/TCP to Modbus RTU use the *Unit ID* for addressing the Modbus RTU servers. For this reason, the Unit ID can be set.

Contents

This chapter covers the following topics:

Topic	Page
ModbusReadReg(): Acyclical reading of registers	20
ModbusWriteReg(): Acyclical writing of registers	22
Sample program: Acyclical communication with the Modbus/TCP server	24
RemoteScanConfig(): Configuring RemoteScan	26
RemoteScanStart(): Starting RemoteScan	30
RemoteScanStop(): Stopping RemoteScan	31
Sample program: Cyclic communication with the Modbus/TCP server	32

ModbusReadReg(): Acyclical reading of registers

Introduction

The function ModbusReadReg() copies a register block from a Modbus/TCP server to the variables of the local memory.

Important notes

- While this STX function is being carried out, simultaneous calls of this function in other tasks are blocked.
- While this STX function is being executed, it is not advisable to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym. In the a.m. cases the connection remains open and it might happen that further communication is blocked.
- The IP address is always to be specified directly. It cannot be specified using names.

Function declaration

```
Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int;
```

Parameter	Description
MbParam	Structure of the type MODBUS_PARAM

Type declaration

```
Type
    MODBUS_TYPES: Enum (
                    MBTYPE_WORD          = 0,
                    MBTYPE_LONGLITTLE    = 1,
                    MBTYPE_LONGBIG       = 2,
                    MBTYPE_FLOATLITTLE   = 3,
                    MBTYPE_FLOATBIG          = 4
                  );
    MODBUS_PARAM: Struct
                    IpAddr:              Int;
                    IpPort:              Int;
                    UnitID:              Int;
                    RemoteReg:           Int;
                    RemoteRegCnt:        Int;
                    LocalVar:            Int;
                    LocalType:           MODBUS_TYPES;
                    Timeout:             Int;
                    Res:                 Int;
                  End_Struct;
End_Type;
```

Function parameters

Parameter	Value	Comment
IpAddr	IP-address of Modbus/TCP server	Direct input
IpPort	502	
UnitID	0 ... 255	Unit ID
RemoteReg	Start register number	Register number of source
RemoteRegCnt	1 ... 125	Amount of registers
LocalVar	Local memory address	Address of the target
LocalType	MODBUS_TYPES	Type of the local variable
Timeout	In ms	Timeout
Res	0	Internal usage

Return value

The following return values are possible:

Return value	
0	No error
-1 or -2	Error during connection set-up
-4	Error during data transfer
-5	Error message from Modbus/TCP server
-8	Timeout
-10	No Modbus/TCP license

ModbusWriteReg(): Acyclical writing of registers

Introduction

The function ModbusWriteReg() copies variable values of the local memory into a register block of a Modbus/TCP-server.

Important notes

- While this STX function is being carried out, simultaneous calls of this function in other tasks are blocked.
- While this STX function is being executed, it is not advisable to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym. In the a.m. cases the connection remains open and it might happen that further communication is blocked.
- The IP address is always to be specified directly. It cannot be specified using names.

Function declaration

```
Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int;
```

Parameter	Description
MbParam	Structure of the type MODBUS_PARAM

Type declaration

```
Type
    MODBUS_TYPES: Enum (
                    MBTYPE_WORD          = 0,
                    MBTYPE_LONGLITTLE    = 1,
                    MBTYPE_LONGBIG       = 2,
                    MBTYPE_FLOATLITTLE   = 3,
                    MBTYPE_FLOATBIG      = 4
                  );
    MODBUS_PARAM: Struct
                    IpAddr:              Int;
                    IpPort:              Int;
                    UnitID:              Int;
                    RemoteReg:           Int;
                    RemoteRegCnt:        Int;
                    LocalVar:            Int;
                    LocalType:           MODBUS_TYPES;
                    Timeout:             Int;
                    Res:                 Int;
                  End_Struct;
End_Type;
```

Function parameters

Parameter	Value	Comment
IpAddr	IP-address of the Modbus/TCP server	Direct input
IpPort	502	
UnitID	0 ... 255	Unit ID
RemoteReg	Start register number	Register number of destination
RemoteRegCnt	1 ... 125	Amount of registers
LocalVar	Local memory address	Source address
LocalType	MODBUS_TYPES	Type of the local variable
Timeout	In ms	Timeout
Res	0	Internal usage

Return value

The following return values are possible:

Return value	
0	No error
-1 or -2	Error during connection set-up
-4	Error during data transfer
-5	Error message from Modbus/TCP server
-8	Timeout
-10	No Modbus/TCP license

Sample program: Acyclical communication with the Modbus/TCP server

Task

On external request, the content of a single register is to be sent to the communication partner.

Solution

The value contained in a single register is sent to the communication partner in acyclic mode by means of the STX functions `ModbusWriteReg()` and `ModbusReadReg()`.

Setting flag "bWrite" triggers once a register block to be written to the Modbus/TCP server.

Setting flag "bRead" triggers once a register block sent by the Modbus/TCP server to be read.

JetSym STX program

```
Var
    bWrite, bRead        : Bool;
    MyData2Send          : Array[0..4] Of Word;
    MyData2Receive       : Array[0..4] Of Word;
    MyResult             : Int;

    MyModbusTCPWrite     : MODBUS_PARAM;
    MyModbusTCPRead      : MODBUS_PARAM;
End_Var;

Task tModBusTCPacyclic Autorun

Loop
If bwrite Then;
    bWrite := False;
    // Allocating a parameter
    // IP address of the server
    MyModbusTCPWrite.IpAddr       := IP#192.168.10.17;
    // Port number (502)
    MyModbusTCPWrite.IpPort       := 502;
    // UnitID
    MyModbusTCPWrite.UnitID       := 0x01;
    // Start register number in the server
    MyModbusTCPWrite.RemoteReg    := 5;
    // Amount of 16-bit registers
    MyModbusTCPWrite.RemoteRegCnt := 5;
    // Pointer to the local memory
    MyModbusTCPWrite.LocalVar     := &MyData2Send;
    MyModbusTCPWrite.LocalType    := MODBUS_TYPES.MBTYPE_WORD;
    // Timeout in ms
    MyModbusTCPWrite.Timeout      := 200;
    // Reserve
    // MyModbusTCPWrite.Res       := 0;

    MyResult := ModbusWriteReg(MyModbusTCPWrite);
End_If;
```

```
If bRead Then;
    bRead := False;
    // Allocating a parameter
    // IP address of the server
    MyModbusTCPRead.IpAddr       := IP#192.168.10.17;
    // Port number (502)
    MyModbusTCPRead.IpPort       := 502;
    // UnitID
    MyModbusTCPRead.UnitID       := 0x01;
    // Start register number in the server
    MyModbusTCPRead.RemoteReg    := 5;
    // Amount of 16-bit registers
    MyModbusTCPRead.RemoteRegCnt := 5;
    // Pointer to the local memory
    MyModbusTCPRead.LocalVar     := &MyData2Receive;
    MyModbusTCPRead.LocalType    := MODBUS_TYPES.MBTYPE_WORD;
    // Timeout in ms
    MyModbusTCPRead.Timeout      := 200;
    // Reserve
    // MyModbusTCPRead.Res       := 0;

    MyResult := ModbusReadReg(MyModbusTCPRead);
End_If;
End_Loop;

End_Task;
```

Related topics

- **STX function ModbusWriteReg(): Acyclical writing of registers** (see page 22)
- **STX function ModbusReadReg(): Acyclical reading of registers** (see page 20)

RemoteScanConfig(): Configuring RemoteScan

Introduction

The function RemoteScanConfig() configures RemoteScan. Nevertheless, communication (cyclic data transfer) does not start yet.

The function RemoteScanStart() starts and the function RemoteScanStop() stops communication.

Important notes

- The IP address is always to be specified directly. It cannot be specified using names. The IP address is always the first entry of a communication unit.
- For modules without input or output registers the corresponding amount must be set to 0.
- If the configuration comprises inputs and outputs, the outputs are sent first, then the inputs are read.

Restriction

You can only set a limited amount of communication units for RemoteScan:

Parameter	Description
Amount of communication units	1 ... 10

Function declaration

```
Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int,
Const Ref Configuration: RSCAN_DSCR): Int;
```

Parameter	Description
`Protocol`	Protocol identification (Modbus/TCP = 5)
`Elements`	Amount of communication units within the structure, which is referenced by the third parameter (1 ... 10)
`Configuration`	Data structure for configuration

Type declaration

```
Type
    RSCAN_STATBIT:  Enum (
                        RUNNING         = 0,
                        Error           = 1
                    );

    RSCAN_PROTOCOL: Enum (
                        MODBUSTCP       = 5
                    );
```

```
RSCAN_ERRTYPE:  Enum (
                    MBTIMEOUT       = -8,
                    MBTRANSACTION   = -7,
                    MBRECEIVE       = -6,
                    MBEXCEPTION     = -5,
                    MBREADINPUTS    = -4,
                    MBWRITEOUTPUTS  = -3,
                    MBCONNECT       = -2,
                    MBNETDRIVER     = -1,
                    NOERROR         = 0,
                    TIMEOUT         = 101,
                    LOCALACCESS     = 102,
                    COMMUNICATION1  = 103,
                    COMMUNICATION2  = 104
                );

RSCAN_STATUS:   Struct
                    Status:         RSCAN_STATBIT;
                    ErrType:        RSCAN_ERRTYPE;
                    ErrCnt:         Int;
                End_Struct;

RSCAN_DSCR:     Struct
                    IpAddr:         Int;
                    IpPort:         Int;
                    UnitID:         Int;
                    Update:         Int;
                    OutRegCnt:      Int;
                    OutRemoteReg:   Int;
                    OutLocalVar:    Int;
                    OutLocalType:   Int;
                    InRegCnt:       Int;
                    InRemoteReg:    Int;
                    InLocalVar:     Int;
                    InLocalType:    Int;
                    Timeout:        Int;
                    Status:         Pointer To RSCAN_STATUS;
                    Res:            Int;
                End_Struct;
End_Type;
```

Design of the structure for configuration

The third parameter of the function RemoteScanConfig() is the address of the description structure of the first communication unit. The description structure of further communication units (up to the amount given in the second parameter) must be stored following each other seamlessly. Creating a respective array of the type RCAN_DSCR suggests itself.

Parameter	Value	Comment
Communication unit 1		

Parameter	Value	Comment
IPAddr	IP-address of the Modbus/TCP server	Direct input
IpPort	502	Modbus/TCP: 502
UnitID	0 ... 255	
Update	10 ... 65.535 ms	
OutRegCnt	0 ... 125	Amount of output registers
OutRemoteReg	0 ... 65.535	First output register number
OutLocalVar	Local address	First output variable
OutLocalType	MODBUS_TYPES	Type of the local variable
InRegCnt	0 ... 125	Amount of input registers
InRemoteReg	0 ... 65,535	First input register number
InLocalVar	Local address	First input variable
InLocalType	MODBUS_TYPES	Type of the local variable
Timeout	In ms	
Status	Local address	Status while communication is going on
Res	0	Internal usage
Communication unit 2		
IpAddr	IP-address of the Modbus/TCP server	Direct input
...	...	...

Return value

The following return values are possible:

Return value	
> 0	Amount of initialized communication units
-1	Protocol not supported
-2	Set amount of communication units > 10
-3	Invalid local address
-4	The value of the remote register is greater than 65,535
-5	RemoteScan is already running
-10	Modbus/TCP is not available (R 202930 bit 5 = 0)

Structure of status register block RSCAN_STATUS

While RemoteScan is running, error messages of this communication unit are stored to the status register block.

Parameter	Value	Bemerkung
Status (bit-coded)	Bit 0 = 1	RemoteScan is running; the bit is set after each update cycle
	Bit 1 = 1	An error has occurred; the bit is set after each error
Error		The code of the latest error can always be read
	0	No error
	< 0	Application-specific error
		In the case of Modbus/TCP:
	-1	Error in network driver
	-2	Error in the connection management
	-3	Error when sending output register values
	-4	Error when reading input register values
	-5	Feedback: Exception
	-6	Error when receiving the feedback
	-7	Wrong transaction ID
	-8	Timeout
	101	Timeout
	102	Error when reading/writing the local variables
	103/104	Error in the lower-level communication layer
ErrCnt		Amount of errors; it is incremented after each error

RemoteScanStart(): Starting RemoteScan

Introduction

The function RemoteScanStart() starts RemoteScan.

RemoteScan has been configured beforehand by the RemoteScanConfig() function.

Function declaration

```
Function RemoteScanStart(Protocol: Int): Int;
```

Parameter	Description
`Protocol`	Protocol identification RSCAN_PROTOCOL (Modbus/TCP = 5)

Return value

The function RemoteScanStart() always returns value 0.

RemoteScanStop(): Stopping RemoteScan

Introduction

The function RemoteScanStop() is used to stop a running RemoteScan and to close all possibly existing communication connections.

Function declaration

```
Function RemoteScanStop(Protocol: Int): Int;
```

Parameter	Description
Protocol	Protocol identification RSCAN_PROTOCOL (Modbus/TCP = 5)

Return value

The function RemoteScanStop() always returns value 0.

Sample program: Cyclic communication with the Modbus/TCP server

Task

Between Modbus/TCP client and Modbus/TCP server, I/O data are to be interchanged cyclically.

Solution

For cyclic data transmission the function "RemoteScan" is used. First, configure the RemoteScan function by invoking the `RemoteScanConfig()` function. You can do this by entering the configuration data into the required structures. The starting address of these structures is transferred along with the `RemoteScanConfig()` function when it is invoked. If initialization was successful, RemoteScan function is started via `RemoteScanStart()` function and cyclic communication sets in. If required, stop cyclic communication invoking the `RemoteScanStop()` function.

JetSym STX program

```
Var
    bStart, bStop           : Bool;
    MyData2Send             : Array[0..9] Of Word;
    MyData2Receive          : Array[0..9] Of Word;
    MyResult                : Int;
    MyStatus                : RSCAN_STATUS;

    MyModbusTCPConnection1  : RSCAN_DSCR;
End_Var;

Task tModBusTCPzyklisch Autorun

// Allocating a parameter
// IP address of the server
MyModbusTCPConnection1.IpAddr         := IP#192.168.10.17;
// Port number (502)
MyModbusTCPConnection1.IpPort         := 502;
// UnitID
MyModbusTCPConnection1.UnitID         := 0x01;
// Update rate in ms
MyModbusTCPConnection1.Update         := 1000;
// Amount of registers
MyModbusTCPConnection1.OutRegCnt      := 5;
// Start register number
MyModbusTCPConnection1.OutRemoteReg   := 0;
// Pointer to the local memory
MyModbusTCPConnection1.OutLocalVar    := &MyData2Send;
MyModbusTCPConnection1.OutLocalType   :=
MODBUS_TYPES.MBTYPE_WORD;
// Amount of registers
MyModbusTCPConnection1.InRegCnt       := 5;
// Start register number
MyModbusTCPConnection1.InRemoteReg    := 100;
// Pointer to the local memory
MyModbusTCPConnection1.InLocalVar     := &MyData2Receive;
MyModbusTCPConnection1.InLocalType    :=
MODBUS_TYPES.MBTYPE_WORD;
```

```
// Timeout in ms
MyModbusTCPConnection1.Timeout        := 20;
// Status
MyModbusTCPConnection1.Status         := &MyStatus;
// Reserve
// MyModbusTCPConnection1.Res         := 0;

Loop
When bStart Continue;
bstart := False;

// Configuring
MyResult := RemoteScanConfig(RSCAN_PROTOCOL.MODBUSTCP, 1,
MyModbusTCPConnection1);

// Starting
If MyResult > 0 Then
    RemoteScanStart(RSCAN_PROTOCOL.MODBUSTCP);

    // Stopping
    When bStop Continue;
    bStop := False;
    RemoteScanStop(RSCAN_PROTOCOL.MODBUSTCP);
End_If;
End_Loop;

End_Task;
```

Related topics

- **STX function RemoteScanConfig(): Configuring RemoteScan** (see page 26)
- **STX function RemoteScanStart(): Starting RemoteScan** (see page 30)
- **STX function RemoteScanStop(): Stopping RemoteScan** (see page 31)

1.5 Data files

Introduction

The latest register values and flag states can be written into a file or read-out of a file with the help of two functions. This read/write process is controlled by the application program.

References

- The file format is supported both by JetSym and non-STX controllers.
- The files created this way can, for example, be read by JetSym or by using the function `FileDARead()`.
- The file format has been described in the user manuals of the controllers.

Contents

Topic	Page
Writing a data file	35

Writing a data file

Introduction

By this function, you can create a data file (data extension .da, not .stxda) or apply values to an already existing data file. This function lets you backup registers and flags.

Function declaration

```
Function FileDAWrite(Const Ref FileName: String, Const Ref Mode:
String, VarType: DAWRITE_TYPE, First: Int, Last: Int): Int;
```

Parameter	Description
FileName	Name of the data file
Mode	Creating a file or adding values
VarType	Registers or Flags
First	Number of the first register or flag
Last	Number of the last register or flag

Mode

Mode is used to indicate whether the file is opened for write or read access.

- The "fWrite" mode is for opening an already existing file or for creating a new one. In this case, the contents of an existing file is cleared.
- If to the contents of an already existing file further contents are to be added, mode "fAppend" must be specified.
- For read access, mode "fRead" must be specified.

fWrite, fAppend und fRead are predefined macros, which have been defined as the following constants:

```
Const
    fAppend = 'a';
    fWrite  = 'w';
    fRead   = 'r';
End_Const;
```

Type declaration

```
Type
    DAWRITE_TYPE: Enum (
        DAREGSITER    = 1,
        DAFLAG         = 3,
    );
End_Type;
```

Return value

The following return values are possible:

Return value	
0	No error
-1	Error at opening/creating the file
-2	Error at writing to the file
-6	Register/flag number exceeds the permitted range
-7/-13	The file name is too long
-10	The data file feature is not available
-11/-12	Internal OS error

JetSym STX program

The states of flags 10 through 25 and the values of registers 1000100 through 1000199 are to be stored to one data file.

```
Var
    nResult:    Int;
End_Var;

Task tMain Autorun
    ...

    nResult := FileDAWrite('MyDataFile.da', fWrite,
                           DAWRITE_TYPE.DAREGISTER,
                            1000100, 1000199);
    nResult := FileDAWrite('MyDataFile.da', fAppend,
                           DAWRITE_TYPE.DAFLAG,
                           10, 25);

    ...
End_Task;
```

1.6 Sending an e-mail

Introduction

This chapter describes the STX feature which allows sending previously written e-mails via the application program.

When sending an e-mail from the application program, the device creates the e-mail based on the e-mail template file and inserts variable values if required.

Processing within the application program

Sending an e-mail may take considerable time. Therefore, other tasks of the application program are processed while an e-mail is being sent. Only one e-mail function call is possible at a time. While an e-mail of a task is being sent, all other tasks which invoke the e-mail function are therefore blocked until this operation is completed.

Contents

Topic	Page
Sending an e-mail	38

Sending an e-mail

Introduction

The user creates template files for e-mails. Into these, the controller can enter variables for sending, if required. The controller sends e-mails to an e-mail server which will then forward the message.

Prerequisites

The e-mail feature is available in the controller and has been configured successfully.

Important notes

- While this STX function is being carried out, simultaneous calls of this function in other tasks are blocked.
- While this STX function is being executed, it is not permitted to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym.

Reference:

For a detailed description of the e-mail feature, please refer to the user manuals of the respective controllers.

Function declaration

```
Function EmailSend(Const Ref FileName: String): Int;
```

Parameter	Description
`FileName`	Name of the e-mail template file

Return value

The following return values are possible:

Return value	
0	No error
-1	Internal OS error
-2	FROM not defined
-3	TO not defined
-4	No connection to the e-mail server
-10	The e-mail feature is not available
-11/-12	Internal OS error

JetSym STX program

If an error occurs in the machinery, the controller is to send an e-mail.

```
Var
    nResult:    Int;
    bSendError: Bool;
End_Var;

Task tMain Autorun
    ...

    If bSendError Then
        nResult := EmailSend('/ErrorMails/MainError.email');
        bSendError := FALSE;
    End_If;

    ...
End_Task;
```

1.7 NetCopyList

Introduction

The NetCopyList functions can be used to combine several read/write accesses to registers located on another controller in one Ethernet telegram. In the following cases this results in a significant performance increase:

- If several register values are to be transmitted which are distributed across the memory
- If registers are to be read and written in one go

Minimum requirements

To use the NetCopyList features, JetSym version 4.1.3 or higher is required.

Operating principle

- The following definitions have to be made for communication units:
  - Which register arrays are transmitted?
  - Which registers are read?
  - Which registers are written?
- One or several communication units can be combined in a list.
- Several lists of different length can be configured.
- When sending a list, all communication units of this list are sent in an Ethernet telegram.

Technical specifications

Property	Value
Max. amount of lists	10
Total number of communication units	500
Maximum amount of registers in a list	64

Contents

Topic	Page
Programming the NetCopyList functions	41
Configuring a list	43
Sending a list	45
Deleting a list	46
Configuring, sending and deleting a list	47
Sample program: NetCopyList	49

Programming the NetCopyList functions

Introduction

The NetCopyList functions which are used for programming this feature are included in the programming language of the controller. To program this feature proceed as follows:

Step	Action
1	Initializing the NetCopyList feature (once on program start)
2	Configuring the list(s)
3	Transmitting data by sending the list(s)
4	Optionally deleting the list(s)

Overview of functions

The following functions are available:

Number	Function	Description
1	NetCopyListInit	Initialization which must be carried out at least once each time the application program is launched.
2	NetCopyListConfig	Configuring a list
3	NetCopyListSend	Sending a list
4	NetCopyListDelete	Deleting a list
5	NetCopyList	Processing of functions 2 through 4 in one call

Predefined data types

The following data type has been predefined for a communication unit. To create the parameter list for functions 2 and 5, this data type can be used in the application program.

```
Type
    TNetCopyListElement : Struct
        nCommand     : Int;
        nMode        : Int;
        nCount       : Int;
        nLocalReg    : Int;
        nRemoteReg   : Int;
    End_Struct;

    TNetCopyListL    : Array Of TNetCopyListElement;
End_Type;
```

Communication unit

In each communication unit, one network command is defined. Several communication units add up to a list of network instruction. Die communication units of a list must be stored to the memory one after another. The parameters have the following meaning:

TNetCopyListElement	
nCommand	
0 =	End of list
1 =	Copy values from remote registers into local registers
2 =	Copy values from local registers into remote registers
3 =	Copy values from remote registers into local variables
4 =	Copy values from local variables into remote registers
nMode	
0 =	Target address is not changed
1 =	Autoincrement the target address
2 =	Autodecrement the target address
nCount	
1 ... 64	Amount of registers to be copied
nLocalReg	
nCommand = 1 or 2	Number of the first local register
nCommand = 3 or 4	Address of the first local variable
nRemoteReg	
	Number of the first remote register

Configuring a list

Introduction

Before a list can be used, it must be configured. This function configures the list.

Function declaration

```
Function NetCopyListConfig(IPAddr : Int,
                           IPPort : Int,
                  Const Ref List : TNetCopyListL) : Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
IPAddr	Valid IP address	Address of the device to which the list is to be sent
IPPort	Valid IP port	
List	Variable address	Start address of the list containing the communication units

Return value

The following return values are possible:

Return value	
> 0	Handle associated with this list; this value has to be stored since it will be needed as parameter for sending and deleting the list.
-1	All lists are already being used; no available list has been found
-2	All communication units are already being used; no available communication units have been found
-3	Empty list
-4	Invalid list
-5	Invalid IP address
-6	Invalid instruction
-8	Invalid mode
-9	Maximum amount of registers exceeded
-10	Maximum size of requested transmit buffer exceeded
-11	Maximum size of requested receiving buffer exceeded

How to use this function

This way, you can invoke the function and assign its return value to a variable for further utilization:

```
hList := NetCopyListConfig(IP#192.168.25.123, 50000,
                           NetCopyListParam);
```

Operating principle

The controller processes this function in the following steps:

Step	Description	
1	The controller creates a new list in its internal memory.	
2	The controller reads the communication units from the variable that has been passed on. The controller adds instructions to the list until a communication unit is detected where `nCommand = 0` or an error has occurred.	
3	**If ...**	**... then ...**
	... no error has occurred,	... the function is terminated and a positive value is returned as handle for further access to the list.
	an error has occurred,	the function is terminated and a negative value is returned.

Sending a list

Introduction

This function is for sending a previously configured list.

Important note

While this STX function is being executed, it is not permitted to carry out `TaskBreak` or `TaskRestart` for the tasks transmitting the list or to restart the program via JetSym. Failure to take this into account may result in data loss upon receiving.

Function declaration

```
Function NetCopyListSend(Handle : Int) : Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
Handle	Handle associated with the list	Value returned by the function `NetCopyListConfig()`

Return value

The following return values are possible:

Return value	
0	No error
-7	Invalid handle
-12	Error during data transfer

Using this function

This way, you can invoke the function and assign its return value to a variable for further utilization:

```
nResult := NetCopyListSend(hList);
```

Operating principle

The controller processes this function in the following steps:

Step	Description
1	The controller compiles an Ethernet telegram with the values to be written and sends it to the communication partner.
2	The task waits at this instruction until it receives a response and assigns the received values to the local variables/registers.
3	The function stores the result to the specified variables.

Deleting a list

Introduction

This function is for deleting a previously configured list.

Function declaration

```
Function NetCopyListDelete(Handle : Int) : Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
Handle	Handle associated with the list	Value returned by the function `NetCopyListConfig()`

Return value

The following return values are possible:

Return value	
0	No error
-7	Invalid handle

How to use this function

This way, you can invoke the function and assign its return value to a variable for further evaluation:

```
nResult := NetCopyListDelete(hList);
```

Operating principle

The controller processes this function in the following steps:

Step	Description
1	The controller deletes the list from its internal memory.

Configuring, sending and deleting a list

Introduction

This function consecutively invokes the above mentioned functions for configuring, sending and deleting a list.

Function declaration

```
Function NetCopyList(IPAddr : Int,
                     IPPort : Int,
               Const Ref List : TNetCopyListL) : Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
IPAddr	Valid IP address	Address of the device to which the list is to be sent
IPPort	Valid IP port	
List	Variable address	Start address of the list containing the communication units

Return value

The following return values are possible:

Return value	
> 0	Handle associated with this list; this value has to be stored since it will be needed as parameter for sending and deleting the list.
-1	All lists are already being used; no available list has been found
-2	All communication units are already being used; no available communication units have been found
-3	Empty list
-4	Invalid list
-5	Invalid IP address
-6	Invalid instruction
-8	Invalid mode
-9	Maximum amount of registers exceeded
-10	Maximum size of requested transmit buffer exceeded
-11	Maximum size of requested receiving buffer exceeded
-12	Error during data transfer

Using this function

This way, you can invoke the function and assign its return value to a variable for further evaluation:

```
nResult := NetCopyList(IP#192.168.25.123, 50000,
                       NetCopyListParam);
```

Operating principle

The controller processes this function in the following steps:

Step	Description
1	The controller invokes the function `NetCopyListConfig()`.
2	The controller invokes the function `NetCopyListSend()`.
3	The controller invokes the function `NetCopyListDelete()`.

Related topics

- **Configuring a list** (see page 43)
- **Sending a list** (see page 45)
- **Deleting a list** (see page 46)

Sample program: NetCopyList

Task

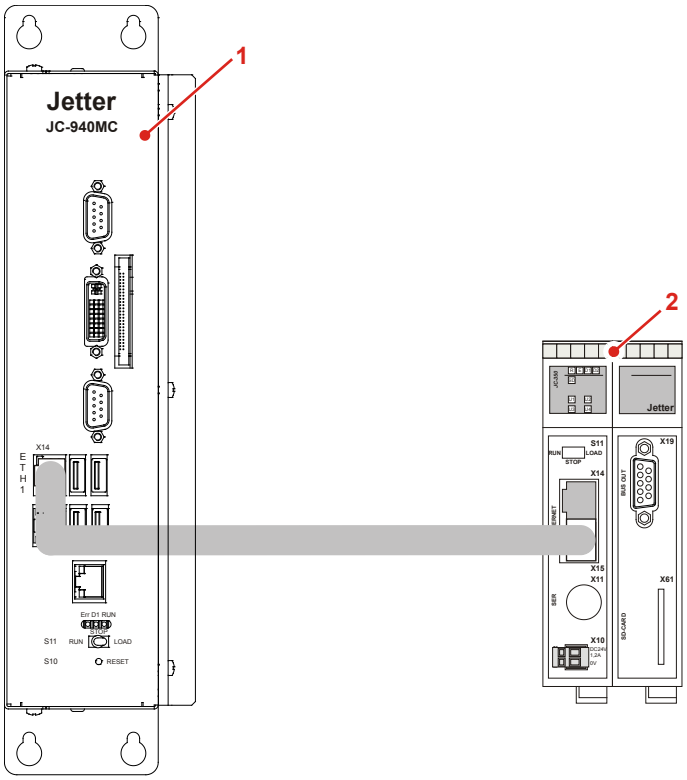
The register and variable contents located on a JetControl 940MC are to be copied into the registers of a JetControl 350. Furthermore, register contents of the JC-350 are to be read out by JC-940MC and copied into local registers and variables. This task is a cyclic one.

Solution

In the application program of JC-940MC, configure a list using the NetCopyList functions, which reads out and writes to JC-350 registers in one telegram.

Sample configuration

This example is based on the following configuration:



Number	Part	Description
1	JC-940MC	Controller
2	JC-350	Controller

Due to the platform-independent implementation of the NetCopyList function this sample program can be used for other configurations simply by changing the register numbers.

JetSym STX program

```
Const
    cNetCopyReadReg     = 1;
    cNetCopyWriteReg    = 2;
    cNetCopyReadVar     = 3;
    cNetCopyWriteVar    = 4;
End_Const;

Var
    nLocalVar          : Int;
    anLocalVar        : Array[5] Of Int;
    LocalReg           : Int At %VL 100;
    aLocalReg          : Array[3] Of Int At %VL 200;

    NetCopyListParam : Array[5] Of TNetCopyListElement;
    hList, nRes        : Int;
End_Var;

Task tNetCopyListText Autorun

    NetCopyListInit();

    // Creating a list of four elements
    NetCopyListParam[0].nCommand   := cNetCopyReadReg;
    NetCopyListParam[0].nMode       := 1;
    NetCopyListParam[0].nCount      := 1;
    NetCopyListParam[0].nLocalReg  := &LocalReg;
    NetCopyListParam[0].nRemoteReg := 1000100;

    NetCopyListParam[1].nCommand   := cNetCopyWriteReg;
    NetCopyListParam[1].nMode       := 1;
    NetCopyListParam[1].nCount      := 3;
    NetCopyListParam[1].nLocalReg  := &aLocalReg;
    NetCopyListParam[1].nRemoteReg := 1000200;

    NetCopyListParam[2].nCommand   := cNetCopyReadVar;
    NetCopyListParam[2].nMode       := 1;
    NetCopyListParam[2].nCount      := 5;
    NetCopyListParam[2].nLocalReg  := &anLocalVar;
    NetCopyListParam[2].nRemoteReg := 1000300;

    NetCopyListParam[3].nCommand   := cNetCopyWriteVar;
    NetCopyListParam[3].nMode       := 1;
    NetCopyListParam[3].nCount      := 1;
    NetCopyListParam[3].nLocalReg  := &nLocalVar;
    NetCopyListParam[3].nRemoteReg := 1000400;

    // Ende der Parameterliste
    NetCopyListParam[4].nCommand   := 0;
```

```
    // Configuring a list
    hList := NetCopyListConfig(IP#192.168.10.208,
                               50000,
                               NetCopyListParam);

    Loop
        If hList > 0 Then
            // Sending a list
            nRes := NetCopyListSend(hList);
        End_If;
        Delay(T#1s);
    End_Loop;
End_Task;
```

Jetter AG

Graeterstrasse 2

D-71642 Ludwigsburg

Germany

Phone:	+49 7141 2550-0
Phone - Sales:	+49 7141 2550-433
Fax - Sales:	+49 7141 2550-484
Hotline:	+49 7141 2550-444
Internet:	http://www.jetter.de
E-Mail:	sales@jetter.de

Jetter Subsidiaries

Jetter (Switzerland) AG

Straussbergstrasse 11

CH-8570 Weinfelden

Switzerland

Phone:	+41 71 91879-50
Fax:	+41 71 91879-59
E-Mail:	info@jetterag.ch
Internet:	http://www.jetterag.ch

Jetter Automation Technology (Shanghai) Co., Ltd.

NO.49, Lane 2927, Lai Yang Road,

Pudong New District

CN-200137 Shanghai

China

Phone:	+86 215 869-1233
Fax:	-
E-Mail:	contact@jetterat.cn
Internet:	http://www.jetterat.cn

Jetter USA Inc.

13075 US Highway 19 North

Florida - 33764 Clearwater

U.S.A

Phone:	+1 727 532-8510
Fax:	+1 727 532-8507
E-Mail:	bschulze@jetterus.com
Internet:	http://www.jetter.de