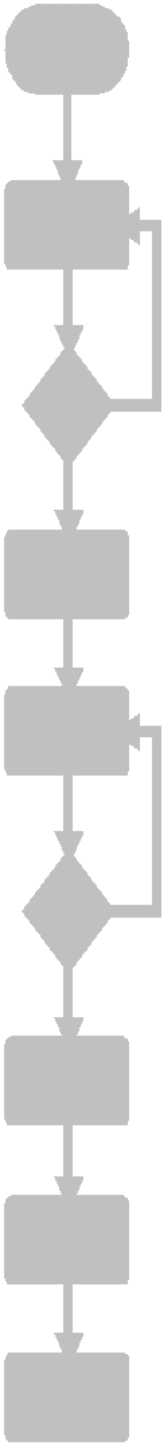


JetControl 647

Version Update

V 3.03 to V 3.50

Jetter

Edition 1.00

4 May 2005 / Printed in Germany

Jetter AG reserves the right to make alterations to its products in the interest of technical progress. These alterations need not to be documented in every single case.

This manual and the information contained herein have been compiled with due diligence. Jetter AG shall not be liable for printing errors contained herein or for other consequential damage.

Table of Contents

1 Introduction

Version Updates - Survey			
Version	**Function**	**upgraded**	**corrected**
V. 3.03	Application program	✓	
	Interpreter		✓
	C-Task	✓	
V 3.50	E-mail	✓	✓
	http-server	✓	✓
	FTP-server	✓	✓
	Files system	✓	✓
	Interpreter	✓	✓
	Registers	✓	
	C-Task	✓	
	Modbus TCP	✓	✓

2 Eliminated Software Bugs

2.1 Interpreter

2.1.1 OUT Instructions with IO64 Boards

Up to this version, there was an error when assignment to outputs was to be made. If direct assignment by means of output numbers between 501 and 864 had been carried out, an incorrect value would be transmitted to the left output. Yet, this error would only refer to the constellation mentioned below. If other output numbers or, for example, an indirect output instruction was used, this function would be fault-free.
The following term would lead to an error:

Assignment OUT xxx = Out501-864

This error has been corrected.

2.1.2 SHIFT instructions including a count with zero

Up to this version, there was an error when shift instructions were to be given. If the amount 0 had been transmitted for giving a SHIFT instruction, the result would be altered without permission.

This error has been corrected.

2.1.3 POS instructions, when the axis number is greater than 255, and when this number is identified directly

Up to this version, there was an error when POS instructions were to be given. Transmitting a POS instruction to an intelligent JX2-module would only function properly when directed to module bus slot 1 and 2. On module bus slot 3, the POS instruction would not be carried out. The error occurred with axis numbers > 255 after specifying one axis number directly. After indirectly specifying the axis number, the POS instruction would be carried out properly.

This error has been corrected.

3 RemoteScan

3.1 General Information

The configurable RemoteScan function is used to cyclically copy register contents from the JetControl to registers of network nodes. On the other hand, they can be read by the nodes and copied in JetControl registers.
The RemoteScan function can be accessed by using special functions 80, 81, and 82.
Presently, only the Modbus/TCP Remote Scan is supported.

3.2 Special Functions

3.2.1 Configuring RemoteScan

To configure RemoteScan, special function 80 is used. However, this function does not start communication (see 3.2.2).

SPECIAL_FUNCTION(80, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the first register of the description block.
<Destination Register Number> Specifies the result register number for this function.

The description block specifies the protocol and the number of communication units. A communication unit specifies the register blocks to be transferred, as well as the address of the communication partner. Up to 10 communication units can be specified. During communication with a partner several different register blocks can be exchanged.

This function can be invoked only with no RemoteScan running. It is not possible to change the configuration of the flash disk during running operation.

3.2.1.1 Description block

Register offset	Description	
0	Protocol	1 = JetWay 3 = JetIP 5 = Modbus/TCP
1	Amount of following communication units	1 .. 10
	Communication unit 1	
2	Address	
3	Port number	Modbus/TCP: 502
4	Update rate	10 .. 65535 ms
5	Amount of output registers	Modbus/TCP: 0 .. 125
6	Output source register number	local Modbus/TCP: 66000 .. 6699
7	Output destination register number	Remote
8	Amount of input registers	Modbus/TCP: 0 .. 125
9	Input source register number	Remote
10	Input destination register number	local Modbus/TCP: 66000 .. 6699
11	Number of the first register of the status register block	
12	Timeout	in milliseconds

With modules without input or output registers the corresponding amount has to be set to 0.
If the configuration comprises inputs and outputs, the outputs are sent first, then the inputs are read.

3.2.1.2 Status register block

The number of the first register of the status register block, consisting of 3 consecutive registers, has to be specified in the description block of each communication unit. Error messages of this communication unit are stored to this block when RemoteScan is running.
The status block has got the following structure:

<table>
<tr><th>Register offset</th><th>Meaning</th><th></th></tr>
<tr><td>0</td><td>Status (bit-coded)</td><td>
<table>
<tr><th>Bit #</th><th>Meaning</th><th></th></tr>
<tr><td>0</td><td>Scan is running</td><td>is set after each update cycle</td></tr>
<tr><td>1</td><td>An error has occurred</td><td>is set each time an error has occurred</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Error code</td><td>
<table>
<tr><th>The code of the last error is displayed. Register contents</th><th>Meaning</th></tr>
<tr><td>0</td><td>No error</td></tr>
<tr><td>< 0</td><td>Application-specific error
<table>
<tr><th>Modbus/TCP: Code</th><th>Meaning</th></tr>
<tr><td>-1</td><td>Error in the network driver</td></tr>
<tr><td>-2</td><td>Error in the connemanagement</td></tr>
<tr><td>-3</td><td>Error when sending output regi</td></tr>
<tr><td>-4</td><td>Error when reading input regist</td></tr>
<tr><td>-5</td><td>Exception response</td></tr>
<tr><td>-6</td><td>Error when receiving the respo</td></tr>
<tr><td>-7</td><td>Wrong transaction ID</td></tr>
<tr><td>-8</td><td>Timeout</td></tr>
</table>
</td></tr>
<tr><td>101</td><td>Timeout</td></tr>
<tr><td>102</td><td>Error when reading/writing local registers</td></tr>
<tr><td>103 / 104</td><td>Error in the lower-level communication layer</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>Number of errors</td><td>The number is incremented each time an error occurs.</td></tr>
</table>

Note: It is useful to initialize the contents of the status register blocks with 0 before starting the RemoteScan.

3.2.1.3 Error Messages

After the function has been carried out, the results can be seen in the results register.

Register contents	Meaning
> 0	Amount of configured communication units
-1	Protocol not supported
-2	Set amount of communication units > 10
-3	Invalid address or port number
-4	Invalid register number
-10	RemoteScan is already running

3.2.2 Starting RemoteScan

Special function 81 is used to start a RemoteScan that has been configured using special function 80.

SPECIAL_FUNCTION(81, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the parameter register.
<Destination Register Number> Specifies the result register number for this function.

No parameters are transferred to this function. Thus, the content of the register to which <Source Register Number> is pointing, is irrelevant.
This function always returns value 0 as result.

3.2.3 Stopping RemoteScan

Special function 82 is used to stop a running RemoteScan. When doing so, all possibly existing communication connections are closed.

SPECIAL_FUNCTION(82, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the parameter register.
<Destination Register Number> Specifies the result register number for this function.

No parameters are transferred to this function. Thus, the content of the register to which <Source Register Number> is pointing, is irrelevant.
This function always returns value 0 as result.

Please mind: Execution of this function may take a relatively long time, depending on the configuration, since it waits until all currently running transfers will be terminated.

3.3 Registers

RemoteScan Protocol Registers	
JC 647	**JC 24x**
63020	**2965**
Function	**Description**
Read	Protocol 1 = JetWay 3 = JetIP 5 = Modbus/TCP
Write	not possible
Value range	32 bits
Value after reset	0

RemoteScan – Amount of Units	
JC 647	**JC 24x**
63021	**2966**
Function	**Description**
Read	Amount of communication units
Write	not possible
Value range	32 bits
Value after reset	0

RemoteScan Activation	
JC 647	**JC 24x**
63022	**2967**
Function	**Description**
Read	0 = Not active 1 = active (running)
Write	not possible
Value range	
Value after reset	0

3.4 Description

For using the communication protocol 'Modbus/TCP', a licence file will not be needed. This protocol is available as of version 3.04.

3.5 Registers

The bit-coded register 63827 has been expanded by 2 bits.

<table>
<tr><th colspan="2">Register 63827: Web Functions</th></tr>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>Read</td><td>
<table>
<tr><th>Bit Number</th><th>Meaning</th></tr>
<tr><td>0</td><td>0 = FTP server not available
1 = FTP server available</td></tr>
<tr><td>1</td><td>0 = HTTP server not available
1 = HTTP server available</td></tr>
<tr><td>2</td><td>0 = E-mail function not available
1 = E-mail function available</td></tr>
<tr><td>3</td><td>0 = Data file function not available
1 = Data file function available</td></tr>
<tr><td>4</td><td>0 = No Modbus/TCP
1 = Modbus/TCP has been licensed</td></tr>
<tr><td>5</td><td>0 = Modbus/TCP server not available
1 = Modbus/TCP server has been started</td></tr>
</table>
</td></tr>
<tr><td>Write</td><td>not possible</td></tr>
<tr><td>Value range</td><td>0 .. 255</td></tr>
<tr><td>Value after reset</td><td>Depends on initialization</td></tr>
</table>

4 Server

4.1 Description

After starting the Modbus/TCP server, an external client can access registers, inputs and outputs. In doing so, 4 connections may be opened at the same time.

4.2 Supported Instructions

4.2.1 Register Access

Since only registers with a width of 16 bits can be transferred via Modbus/TCP, access to the high-order 16 bits of JetControl registers is not possible. When receiving register values, sign extension to 32 bits will not be carried out.

4.2.2 Class 0

4.2.2.1 read multiple registers (fc 3)

Reading register sets.
The number of the start register corresponds to the register number within the JC-647.

4.2.2.2 write multiple registers (fc 16)

Writing register sets.
The number of the start register corresponds to the register number within the JC-647.

4.2.3 Class 1

4.2.3.1 read coils (fc 1)

Reading outputs.
The output number has to be transferred in the internal format of the JC-647.

JetSym – User-oriented numbering format	JC-647 – internal numbering
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
etc.	etc.

4.2.3.2 read input discretes (fc 2)

Reading inputs.
The input number has to be transferred in the internal format of the JC-647.

JetSym – User-oriented numbering format	JC-647 – internal numbering
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
etc.	etc.

4.2.3.3 read input registers (fc 4)

Reading inputs summarized in 16-bit words.

Inputs	Register Number
101 .. 116	0
201 .. 216	2
301 .. 316	4
etc.	etc.

4.2.3.4 write coil (fc 5)

Activating/deactivating an individual output.
The output number has to be transferred in the internal format of the JC-647.

JetSym – User-oriented numbering format	JC-647 – internal numbering
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
etc.	etc.

4.2.3.5 write single register (fc 6)

Entering values into the low-order 16 bits of a JC-647 register.

4.2.4 Class 2

4.2.4.1 force multiple coils (fc 15)

Activating/deactivating several outputs.
The output number has to be transferred in the internal format of the JC-647.

JetSym – User-oriented numbering format	JC-647 – internal numbering
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
etc.	etc.

4.2.4.2 read / write registers

Writing and simultaneously reading registers.
The number of the start register corresponds to the register number within the JC-647. First, the registers polled by the client are read, then, the registers transferred from the client are stored.

5 Client

5.1 General Information

The Modbus/TCP client in JetControl 647 supports Class 0 Conformance (see 4.2.2). This means that read and write multiple registers instructions are used. Up to 125 registers with a width of 16 bits can be transmitted in one frame. When sending 32-bit registers only the lower-order 16 bits are transmitted. When assigning incoming register values to the JetControl-internal 32-bit registers sign extension to 32 bits will not be carried out.
As protocol ID "0" is used, as unit ID "1". Assignment of sent and received frames is carried out using the transaction ID.
Connections to 11 different servers may be opened at the same time.

5.2 RemoteScan

This function is for cyclically transferring the inputs and outputs 14001 through 19999 that are combined in the 16-bit registers 66000 through 66999 from and to the configured servers. One connection is established to each server (IP address and port) irrespective of the number of communication units configured on this server.
If several communication units are configured on one server, accesses are serialized since servers, as a rule, do not support "command pipelining". If several servers have been configured, communication is carried out in parallel.

5.2.1 Overlaying of inputs/outputs on registers

JC 24x		JC 647	
Registers	Inputs / Outputs	Registers	Inputs / Outputs
8000	20001 – 20016	66000	14001 – 14016
8001	20017 – 20032	66001	14017 – 14032
8002	20033 – 20048	66002	14033 – 14048
etc.	etc.	etc.	etc.
8999		66375	19999
		66998	Not assigned
		66999	Not assigned

Since the registers and the inputs/outputs overlaid on them merely are memory cells located in the RAM, and no direct mapping to hardware takes place, it is not determined whether a register contains inputs or outputs. Assignment is made not until configuration in the communication units takes place.

Formula:

RegNo = (IONo-14001)/16+66000
BitNo = (IONo-14001) Modulo 16

e.g.
IO no. 14033 results in register 66002, bit no. 0

5.3 Special Functions

As acyclic transmission channel to a Modbus/TCP server the special functions 65 (reading registers) and 66 (writing registers) can be used (the functions are available independent of RemoteScan).
While one of these two special functions is being carried out simultaneous calls of this functions in other tasks are blocked until this function will be terminated.
These functions establish a connection to the specified server, transmit the desired data and clear down the connection. In case a connection exists that has been established by RemoteScan, this connection will be used. Setting-up and clearing-down the connection is not required.
Please mind: It is not advisable to issue TaskBreak or TaskRestart instructions for this task or to restart the program through JetSym while one of these functions is carried out since in such a case the connection remains established which may block additional transmissions.

5.3.1 Reading registers

SPECIAL_FUNCTION(65, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the first register of a description block.
<Destination Register Number> Specifies the result register number for this function.

Description Block		
Register offset	Meaning	
0	IP-address	
1	Port number	502
2	Timeout	in milliseconds
3	Number of the source register	remote
4	Number of the designation register	local
5	Amount of registers	1 .. 125

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

5.3.2 Writing to registers

SPECIAL_FUNCTION(66, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the first register of a description block.
<Destination Register Number> Specifies the result register number for this function.

Description Block		
Register offset	Meaning	
0	IP-address	
1	Port number	502
2	Timeout	in milliseconds
3	Number of the source register	local
4	Number of the designation register	remote
5	Amount of registers	1 .. 125

Result	Meaning
0	No error
-1 or -2	Error during connection set-up
-3	Error during data transfer
-5	Error message from server

-8	Timeout
-10	No Modbus/TCP license

5.3.3 ST sample program

```
// ********************************************************************************
// *** Program: ModbusTCPTest.stp
// *** Version: 1.0
// *** Date:      24-07-2003
// *** Author:    Jetter AG
// ********************************************************************************
// ********************************************************************************
// *** This Software has the purpose to test the ModbusTCP function on the JC 647
// ********************************************************************************
// JC 647 :       192.168.10.240
// Phoenix :      192.168.10.25

type
        REMOTESCAN_CFGHEADER:           struct
                nProtocol:                      INT;
                nNoOfRemotes:                   INT;
        end_struct;
        REMOTESCAN_CFG:  struct
                nAddress:                       INT;
                nPortNo:                        INT;
                ntUpdateRate:           INT;
                nNoOfOutputRegs:        INT;
                nOutputSource:          INT;
                nOutputDest:            INT;
                nNoOfInputRegs:                 INT;
                nInputSource:           INT;
                nInputDest:                     INT;
                nFirstStatusReg:        INT;
                nTimeout:                       INT;
        end_struct;
        REMOTESCAN_STATE: struct
                nStatus:                        INT;
                nErrorCode:                     INT;
                nNoOfErrors:            INT;
        end_struct;
        MODBUS_RW: struct
                nAddress:                       INT;
                nPortNo:                        INT;
                nTimeout:                       INT;
                nSource:                        INT;
                nDest:                          INT;
                nNoOfRegs:                      INT;
        end_struct;
end_type;
const
        cMaxNoOfRemotes             = 1;
        cModbusTCP                  = 5;
        cModbusTCPReadReg           = 65;
        cModbusTCPWriteReg = 66;
        cConfigRemoteScan     = 80;
        cStartRemoteScan      = 81;
        cStopRemoteScan       = 82;
#ifdef _CONTROLLER_JC_24X
        cStartRegModbus       = 8000;  // 24x :8000 64x: 66000
#endif
#ifdef _CONTROLLER_JC_64X
```

```
        cStartRegModbus      = 66000;  // 24x :8000 64x: 66000
#endif
        zPosNFText             = 3;
end_const;
var
        stModbusReadWrite:          MODBUS_RW:
                                    at %vl 90;
        nModbusRwResult:            INT
                                    at %vl 90 + sizeof(MODBUS_RW);
        nRemoteScanCfgResult:       INT
                                    at %vl 99;
        stRemoteScanCfgHdr:         REMOTESCAN_CFGHEADER
                                    at %vl 100;
        astRemoteScanConfig: ARRAY[cMaxNoOfRemotes] of REMOTESCAN_CFG
        at %vl 100 + sizeof(REMOTESCAN_CFGHEADER);
        astRemoteScanStatus: ARRAY[cMaxNoOfRemotes] of REMOTESCAN_STATE
        at %vl 200;
        nServerOutputDest:          INT
                                    at %vl 200 +
sizeof(REMOTESCAN_STATE)*cMaxNoOfRemotes;
        nServerInputSource:         INT
                                    at %vl &nServerOutputDest ; // Überlagerung mit
nServerOutputDest !!!!
        nCountOfModules:            INT
                                    at %vl 300;
        nTestReg1:                  INT
                                    at %vl 301;
        nDelayModbusTask:           INT
                                    at %vl 302;
        aStartRegModbus:            ARRAY[100] of INT
                                    at %vl cStartRegModbus ;
        fModbusTest1Flag:           BOOL
                                    at %mx 200;
#ifdef _CONTROLLER_JC_64X
        nRegRemoteScanProtocol:     INT                 at %vl 63020;
        nRegRemoteScanNoCommUnits:  INT                 at %vl 63021;
        nRegRemoteScanActivityState: INT                at %vl 63022;
#endif
end_var;

task 0
        nCountOfModules                 :=  cMaxNoOfRemotes;
        stRemoteScanCfgHdr.nProtocol    := cModbusTCP;
        stRemoteScanCfgHdr.nNoOfRemotes  := nCountOfModules;
#ifdef _CONTROLLER_JC_64X
        // Wait for Starting ...
        flags[100] := FALSE;
        when flags[100] continue;
#endif

        nRemoteScanCfgResult := 22;         // set it to a value != zero

        SYSTEMFUNCTION(cStopRemoteScan,             &stRemoteScanCfgHdr,
&nRemoteScanCfgResult);

        if NOT (nRemoteScanCfgResult = 0)
        then
        // Result 0
```

```
        DISPLAY_TEXT(0, zPosNFText,'Modbus E001$');
        GOTO sError
        end_if;

        // PhoenixContact Smart IO
        astRemoteScanConfig[0].nAddress          := IP#192.168.10.25;
        astRemoteScanConfig[0].nPortNo           := 502;
        astRemoteScanConfig[0].ntUpdateRate      := 10;
        astRemoteScanConfig[0].nNoOfOutputRegs   := 1;
        astRemoteScanConfig[0].nOutputSource     := cStartRegModbus+3;
        astRemoteScanConfig[0].nOutputDest       := 384;
        astRemoteScanConfig[0].nNoOfInputRegs    := 3;
        astRemoteScanConfig[0].nInputSource := 0;
        astRemoteScanConfig[0].nInputDest        := cStartRegModbus+8;
        astRemoteScanConfig[0].nFirstStatusReg   :=
&astRemoteScanStatus[0].nStatus;
        astRemoteScanConfig[0].nTimeout                  := 10;
        // Reset Errors
        astRemoteScanStatus[0].nStatus                   := 0;
        astRemoteScanStatus[0].nErrorCode        := 0;
        astRemoteScanStatus[0].nNoOfErrors       := 0;
        nRemoteScanCfgResult := 22;          // set it to a value != zero
        SYSTEMFUNCTION(cConfigRemoteScan,            &stRemoteScanCfgHdr,
&nRemoteScanCfgResult);
        if NOT (nRemoteScanCfgResult = nCountOfModules)
        then
        // Result 3
        DISPLAY_TEXT(0, zPosNFText,'Modbus E002$');
        GOTO sError
        end_if;
        when nRemoteScanCfgResult > 0 continue;
        nRemoteScanCfgResult := 22;          // set it to a value != zero
        SYSTEMFUNCTION(cStartRemoteScan,             &stRemoteScanCfgHdr,
&nRemoteScanCfgResult);
        if NOT (nRemoteScanCfgResult = 0)
        then
        // Result 0
        DISPLAY_TEXT(0, zPosNFText,'Modbus E003$');
        GOTO sError
        end_if;
label loop:
        // PhoenixContact Smart IO
        stModbusReadWrite.nAddress       := IP#192.168.10.214;
        stModbusReadWrite.nPortNo        := 502;
        stModbusReadWrite.nTimeout       := 100;
        stModbusReadWrite.nSource        := 0;
        stModbusReadWrite.nDest          := cStartRegModbus+6;
        stModbusReadWrite.nNoOfRegs              := 1;
        SYSTEMFUNCTION(cModbusTCPReadReg,            &stModbusReadWrite,
&nModbusRwResult);
        // PhoenixContact Smart IO
        stModbusReadWrite.nAddress       := IP#192.168.10.214;
        stModbusReadWrite.nPortNo        := 502;
        stModbusReadWrite.nTimeout       := 100;
        stModbusReadWrite.nSource        := cStartRegModbus+11;
        stModbusReadWrite.nDest          := 0;
        stModbusReadWrite.nNoOfRegs              := 1;
```

```
SYSTEMFUNCTION(cModbusTCPWriteReg, &stModbusReadWrite,
&nModbusRwResult);
goto loop;
label sError:
goto sFehler;
end_task
```

6 Special Functions 60 and 61 Modbus CRC

6.1 Modbus RTU CRC Checksum

Special functions 60 and 61serve for generating and testing the Modbus RTU CRC checksum.
It is assumed that the characters of the Modbus telegram have been stored to registers one after the other.
Please mind: other than in the special functions, in which the first function parameter specifies a register by means of the input data, while the second prameter specifies a register for the resulting data of the function, the first parameter of Modbus CRC routines specifies the beginning of the Modbus telegram, while the second parameter specifies its end.
The special functions are activated by transferring registers directly or indirectly, e.g.

```
SPECIALFUNCTION (60, 100, 103)
SPECIALFUNCTION (60, 100, @103)
SPECIALFUNCTION (61, 100, 103)
SPECIALFUNCTION (61, @100, @103)
```

6.1.1 Function 60: Calculating the checksum

Operating principle This special function calculates a two-byte checksum from the transferred telegram and adds the two bytes to the end of the telegram.
Parameter 1 Number of the register with the first datum of the Modbus protocol.
Parameter 2 Number of the register with the last data of the Modbus protocol without the two bytes for the CRC check sum.
Potential errors - the number of the last register is smaller than the number of the first register;
- each register may contain useful data in the lowest 8 bits only.

Result in case of error Undefined
Computing time approx. 68 µs when the data length is 100 registers.

6.1.2 Function 61: Checking the Check Sum

Operating Principle This special function checks the check sum of a telegram and adds the result to the end of this telegram.
Parameter 1 Number of the register with the first datum of the Modbus protocol.
Parameter 2 Number of the register with the last datum of the Modbus protocol with the two bytes for the CRC check sum.
Potential errors - the number of the last register is smaller than the number of the first register;
- each register may contain useful data in the lowest 8 bits only.

Result in case of error Undefined
Computing time approx. 68 µs when the data length is 100 registers.

6.1.3 Sample Program

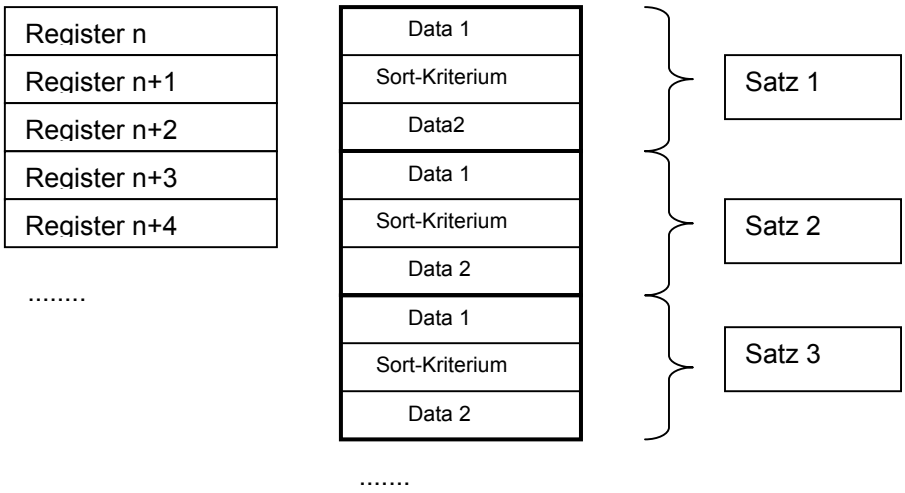
In the following example, the CRC check sum of a received Modbus RTU telegram is checked by means of special function 61. If the check sum is correct, a 1 is returned. Else, a 0 is attached to the telegram received.

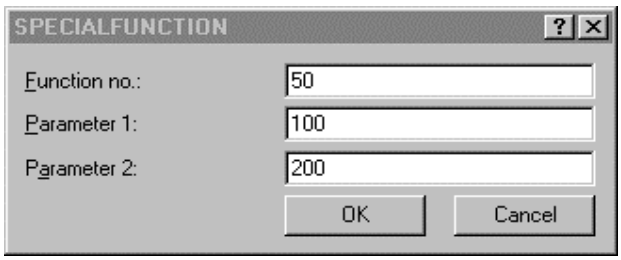
```
REGISTER_LOAD (100, 0x02)          // Slave Address
REGISTER_LOAD (101, 0x03)          // Function code
REGISTER_LOAD (102, 0x00)          // Starting number
REGISTER_LOAD (103, 0x20)          // Starting number
REGISTER_LOAD (104, 0x00)          // Amount
REGISTER_LOAD (105, 0x04)          // Amount
REGISTER_LOAD (106, 0x45)          // CRC check sum
REGISTER_LOAD (107, 0xF0)          // CRC check sum
SPECIALFUNCTION (61, 100, 107)     // Check CRC check sum
// The result is contained in register 108
IF REG 108 = 1
THEN ...                           // CRC check sum correct
ELSE ...                           // CRC check sum incorrect
```

7 Special Function 50: Sorting Data

7.1 Introduction

In order to sort data in the controller, up to now the sorting algorithm had to be written in SYMPAS, respectively in JetSym ST. The advantage of sorting was that each source line was known; the disadvantage was that the sorting performance was not the best.
In order to grant the greatest possible flexibility to the user, the sorting algorithm is separated from the data. The sorting algorithm has been stored to the operating system of the controller. The data that are to be sorted are indirectly addressed by parameter 1. Indirect addressing is necessary, as the value of parameter 2 specifies a register, where the desciptor has been stored. The descriptor contains a register (offset 0) that is pointing to the data. Further, the way of sorting etc. is specified in the descriptor, see "Input descriptor, parameter 1". The descriptor is necessary, as the special functions have got two transfer parameters only. Parameter 2 specifies a register address, to which an error code and the processing time are to be stored. By activating special function 50, the data specified in parameter 1 can be sorted.





SPECIALFUNCTION (50, 100, 200)

Parameter 1 specifies the pointer to the input descriptor.
Parameter 2 specifies the pointer to the result.

7.1.1 Input descriptor (parameter 1)

Parameter 1 specifies, to which part of the controller memory the descriptor will be stored. Before the function is activated, this descriptor must be initialized by entering useful values.

Offset / Parameters / Value range	Parameters	Value range	Description
0	Data Start Register	0-20479	Start Register, which contains the data th to be sorted
1	Amount of data sets	2-1000	Amount of data records, 1000 max.
2	Data length	2-1000	Amount of registers per data record
3	Sorting element	0-999	Element within a data record; this el serves as a sorting criterion.
4	Mode	0-1	This parameter specifies the way of so in ascending or descending sequence Bit position 0: deleted: ascending; set: descending 1: unused 2: unused 3: unused
5	Vacant	unused	kept for further expansions
6	Vacant	unused	kept for further expansions
7	Vacant	unused	kept for further expansions
8	Vacant	unused	kept for further expansions
9	Vacant	unused	kept for further expansions

7.1.2 Return values (parameter 2)

Offset	Parameters	Description
0	**Error code**	**See "Error code (parameter 2)".**
1	**Execution time in [us]**	**Calculation time for sorting the data**
2	**Vacant**	**kept for further expansions**
3	**Vacant**	**kept for further expansions**

7.1.3 Error code (parameter 2), offset 0

After having been activated, parameter 2, offset 0, will contain the return value.

Return value	Meaning
0	**OK; no error has occurred**
1000	**Start register > 20479**
2000	**Target register > 20479**
3000	**Amount of data records > 240**
4000	**Data length > 1000**
5000	**Sorting mode > 256**
6000	**Sorting element > (data length-1)** **The sorting element is outside the valid range**

7.1.4 Execution time (parameter 2), offset 1

The expired time for sorting will be stored to this register.

7.2 Example of data structuring

Parameter 1 of special function 50 = 100
Parameter 2 of special function 50 = 200

Registers	Offset	Parameter / Description	Value
100	0	Data start register	400
101	1	Amount of data records	3
102	2	Amount of elements per data record	4
103	3	Master element_3 is used	2
104	4	Mode	0
105	5	Vacant	
106	6	Vacant	
107	7	Vacant	
108	8	Vacant	
109	9	Vacant	

Registers	Description
400	**DataRecord[0].Element_1**
401	**DataRecord[0].Element_2**
402	**DataRecord[0].Element_3**
403	**DataRecord[0].Element_4**
404	**DataRecord[1].Element_1**
405	**DataRecord[1].Element_2**
406	**DataRecord[1].Element_3**
407	**DataRecord[1].Element_4**
408	**DataRecord[2].Element_1**
409	**DataRecord[2].Element_2**
410	**DataRecord[2].Element_3**
411	**DataRecord[2].Element_4**

8 Sample Program

```
//*********************************************************************************************
//************************ Specialfunction 50 – Sorting of Data in Direct and Indirect Manner ******
//*********************************************************************************************

 REGISTER_LOAD (100, 3000)                  // Data start Register
 REGISTER_LOAD (101, 4)                     // Amount of data records
 REGISTER_LOAD (102, 2)                     // Data length
 REGISTER_LOAD (103, 0)                     // Sorting criterion Element here 0-1
 REGISTER_LOAD (104, 0)                     // Mode: Bit 0  = 0 - Register 3000 contains the smallest value
                                            // Mode: Bit 0  = 1 - Register 3000 contains the greatest value
 REGISTER_LOAD (105, 0)                     // not used at the moment
 REGISTER_LOAD (106, 0)                     // not used at the moment
 REGISTER_LOAD (107, 0)                     // not used at the moment
 REGISTER_LOAD (108, 0)                     // not used at the moment
 REGISTER_LOAD (109, 0)                     // not used at the moment

 // Generate data
```

```
REGISTER_LOAD (3000, 8) // Data
REGISTER_LOAD (3001, 1) // Record 1
REGISTER_LOAD (3002, 1) // Data
REGISTER_LOAD (3003, 2) // Record 2
REGISTER_LOAD (3004, 4) // Data
REGISTER_LOAD (3005, 3) // Record 3
REGISTER_LOAD (3006, 20) // Data
REGISTER_LOAD (3007, 4) // Record 4


// Specify default values for error recognition

REGISTER_LOAD (200, 55555)                  // Offset 0: Specify return value
REGISTER_LOAD (201, 55555)                  // Offset 1: Sorting time in
microseconds. 100 data records in about
                                            //         4000 microseconds
REGISTER_LOAD (202, 0)                      // Offset 2: not used
REGISTER_LOAD (203, 0)                      // Offset 3: not used

SPECIALFUNCTION (50, 100, 200)

IF
        REG 200 # 0
THEN
        DISPLAY_TEXT (0, zPosNFText, "SortFct d/d 1")
        GOTO sError
ELSE

IF
        REG 201 > 1000
THEN
        DISPLAY_TEXT (0, zPosNFText, "SortFct d/d 1.1")
        GOTO sError
ELSE


IF
        REG 3000 # 1 OR  // DATA
        REG 3001 # 2 OR  // record 2
        REG 3002 # 4 OR  // DATA
        REG 3003 # 3 OR  // record 3
        REG 3004 # 8 OR  // DATA
        REG 3005 # 1 OR  // record 1
        REG 3006 # 20 OR // DATA
        REG 3007 # 4     // record 4
THEN
        DISPLAY_TEXT (0, zPosNFText, "SortFct d/d 2")
        GOTO sError
ELSE
```

9 Start Delay

9.1 Description

When the JetControl 647 (supplied with 24 V) is started, a delay time is taken, which has been set in register 63892, before the modules at the system bus are initialized, and before the user program is started.
Only the red LED (ERR) is lit while this time is elapsing. The value in register 63892 specified the delay time in multiples of 100 ms. The min. value is 0 (function disabled). The maximum value is 300, i.e. 30 seconds.

10 Data Files

10.1 General Information

The latest register values and flag states can be written into a file or read-out of a file with the help of some special functions. This read/write process is controlled by the application program.
The file format is identical to the format of the "data dump" files created by JetSym (see 11).
The file names consist of two constant parts and a register content. So, files can be selected by different register values (see 10.2.2).
Written files are stored to the root directory. Files to be loaded must also be located in the root directory. Access to data files is carried out with administrator rights and cannot be restricted.

10.2 Special Functions

10.2.1 Implementation

Since file operations may take considerable long time, especially with large files, other application tasks are processed while one of the file operations is running. However, only one function can be processed at a time. Tasks which invoke one of these functions while a file operation of another task is running are therefore blocked until this operation is completed.
That implies that data consistency of value blocks to be written or read is not ensured. Data consistency has to be ensured by accordingly programming the application program.
The state of the currently running operation can be polled through the registers specified below (see 10.3).

10.2.2 File names

File names always start with "Data_" followed by a numerical value and the extension "da". The numerical value for drawing a distinction between various files is acquired from the parameter register of the special functions.
Examples:
Data_123456789.da
Data_0.da

Please mind: Observe capitalization. The file system is case-sensitive.

10.2.3 Saving values – creating a file

Special function **90** is for creating a new data file and inserting a selectable register or flag block into this file.

SPECIAL_FUNCTION(90, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the first register of the parameter block.
<Destination Register Number> Specifies the result register number for this function.

10.2.3.1 Parameter block

The function parameters are specified starting from register<Source Register Number>.

Register offset	Meaning	
0	"File Name"	Numerical component of the file name (see 10.2.2)
1	Type	1 = Register 3 = Flag
2	Start of data block	Number of the first register or flag
3	End of data block	Number of the last register or flag

10.2.3.2 Function result

The result of the function can be read out of register <Destination Register Number>.

Register Contents	Meaning
0	No error
-1	Error when creating a file (e.g. disk is full)
-2	Error when writing data
-4	Error when closing a file
-6	Invalid register / flag number
-10	Data file function not available (see 10.3)
-20	Internal OS error

10.2.4 Saving values – append to file

Special function **91** is for appending a selectable register or flag block to an existing file. In case this file does not exist, a new file will be created.

SPECIAL_FUNCTION(91, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the first register of the parameter block.
<Destination Register Number> Specifies the result register number for this function.

10.2.4.1 Parameter block

The function parameters are specified starting from register <Source Register Number>.

Register offset	Meaning	
0	"File Name"	Numerical component of the file name (see 10.2.2)
1	Type	1 = Register 3 = Flag
2	Start of data block	Number of the first register or flag
3	End of data block	Number of the last register or flag

10.2.4.2 Function result

The result of the function can be read out of register <Destination Register Number>.

Register Contents	Meaning
0	No error
-1	Error when opening or creating file
-2	Error when writing data
-4	Error when closing a file
-6	Invalid register / flag number
-10	Data file function not available (see 10.3)
-20	Internal OS error

10.2.5 Reading values from a file

Special function **92** is for reading register values and flag states out of a data file and entering them into the corresponding registers or flags. The information is processed in the order specified by the content of the file.

SPECIAL_FUNCTION(92, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the register containing the numerical component of the file name.

<Destination Register Number> Specifies the result register number for this function.

10.2.5.1 Function result

The result of the function can be read out of register <Destination Register Number>.

Register Contents	Meaning
0	No error
-1	Error when opening file (e.g. file not found)
-3	Error when reading data
-4	Error when closing a file
-10	Data file function not available (see 10.3)
-20	Internal OS error

10.2.6 Deleting a file

Special function **96** is for deleting a data file from the flash disk.

SPECIAL_FUNCTION(96, <Source Register Number>, <Destination Register Number>)

<Source Register Number> Specifies the number of the register containing the numerical component of the file name.
<Destination Register Number> Specifies the result register number for this function.

10.2.6.1 Function result

The result of the function can be read out of register <Destination Register Number>.

Register Contents	Meaning
0	No error
-5	Error when deleting the file (e.g. file not found)
-10	Data file function not available (see 10.3)
-20	Internal OS error

10.3Registers

10.3.1 Availability

The bit-coded register 63827 has been expanded by 1 bit.

Register: Web Functions	
JC 647	**JC 24x**
63827	**2930**

Function	**Description**	
Read	Bit Number	Meaning
	0	0 = FTP server not available 1 = FTP server available
	1	0 = HTTP server not available 1 = HTTP server available
	2	0 = E-mail function not available 1 = E-mail function available
	3	0 = Data file function not available 1 = Data file function available
	4	0 = No Modbus/TCP 1 = Modbus/TCP has been licensed
	5	0 = Modbus/TCP server not available 1 = Modbus/TCP server has been started
Write	not possible	
Value range	0 .. 255	
Value after reset	Depends on initialization	

10.3.2 Status register

The processing state of a file operation and the number of the task carrying out the operation can be read from two registers.

Register: Processing State	
JC 647	**JC 24x**
63835	**2977**
Function	**Description**
Read	0: No data file operation in progress 1: Processing transferred to file module 2: Data is being read/written 3: File operation completed
Write	illegal
Value range	0 .. 255
Value after reset	0

Register: Task number	
JC 647	**JC 24x**
63836	**2978**
Function	**Description**
Read	Number of the task performing a file operation 0 .. 99: Task number 255: No task
Write	illegal
Value range	0 .. 255
Value after reset	255

11 Appendices

A.1 File Format

These files are pure text files with one line for each entry. The entries are to be terminated with "carriage return / line feed". Comment lines are allowed.
A data file is to start with the entry "SD1001".
Data lines start with an identifier for the variable. Now follows the variable number, also separated by a blank or tab. Now follows the value of the variable also separated by a blank or tab.
The IDs at the beginning of a line must not be indented.

Variable ID	Variable type
FS	Flags
QS	Floating point numeral register
RS	Integer Register

All lines that do not start with such a variable ID are regarded as comment lines with the exception of the first line containing the file ID.

Example:

```
SD1001
; JC-647 DATA FILE - Jetter AG
FS      111     1
This is a comment
RS      20112   110
  FS    113     1
QS      65024   -3.141593
QS      65025   6.789e-05
```

The third line from below is also a comment line since the variable's ID ("FS") does not stand at the beginning of the line.