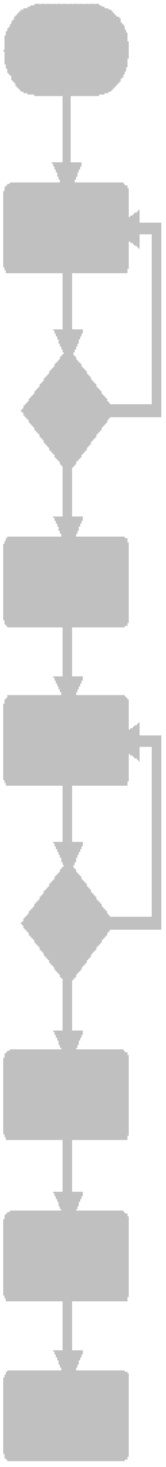


JetControl 647
Versions Update
von V3.03 auf V3.50

Jetter

Auflage 1.00

15.01.04 / Printed in Germany

Inhaltsverzeichnis

1 Einleitung

Versions-Update Übersicht			
Version	**Funktion**	**erweitert**	**korrigiert**
V3.03	Anwenderprogramm	✓	
	Interpreter		✓
	C-Task	✓	
V3.50	E-Mail	✓	✓
	http-Server	✓	✓
	ftp-Server	✓	✓
	Dateisystem	✓	✓
	Interpreter	✓	✓
	Register	✓	
	C-Task	✓	
	Modbus TCP	✓	✓

2 Beseitigte Software-Bugs

2.1 Interpreter

2.1.1 OUT Befehle mit IO64 Karten

Bis zu dieser Version war ein Fehler bei der Zuweisung auf Ausgänge vorhanden. Wurde eine direkte Zuweisung mit Ausgangsnummern zwischen 501 und 864 ausgeführt, dann wurde ein falscher Wert in den linken Ausgang geschrieben. Dieser Fehler bezog sich nur auf die unten erwähnte Konstellation. Wurden andere Ausgangsnummern oder z.b. ein indirekter Ausgangsbefehl verwendet, so war die Funktion fehlerfrei.
Folgender Ausdruck führte zu einem Fehler:

Zuweisung OUT xxx = Out501-864

Dieser Fehler wurde korrigiert.

2.1.2 Shift Befehle bei Count mit NULL

Bis zu dieser Version war ein Fehler bei den Shift Befehlen vorhanden. Wurde den Shift Befehlen die Anzahl 0 übergeben, so wurde das Ergebnis unzulässiger Weise verändert.

Dieser Fehler wurde korrigiert.

2.1.3 POS Befehle bei Achsnummern > 255 und direkter Angabe dieser

Bis zu dieser Version war ein Fehler bei den POS Befehlen vorhanden. Wurde ein POS Befehl auf ein intelligentes JX2 Modul ausgeführt, dann hat dies nur korrekt auf Modulbus-Steckplatz 1 und 2 funktioniert. Auf Modulbus-Steckplatz 3 wurde der POS Befehl nicht ausgeführt. Der Fehler ist bei Achsnummern > 255 aufgetreten, und dann nur bei direkter angabe dieser. Wurde die Achsnummer indirekt angegeben, dann hat der POS Befehl funktioniert.

Dieser Fehler wurde korrigiert.

3 RemoteScan

3.1 Allgemein

Über die konfigurierbare RemoteScan-Funktion werden zyklisch Registerinhalte vom JetControl auf Register von Netzwerk-Teilnehmern kopiert oder von diesen gelesen und in JetControl-Registern abgelegt.
Der Zugriff auf die RemoteScan-Funktion erfolgt über die Spezialfunktionen 80, 81 und 82.
Momentan wird nur der Modbus/TCP Remote Scan unterstützt.

3.2 Spezialfunktionen

3.2.1 RemoteScan konfigurieren

Über die Spezialfunktion 80 wird der RemoteScan konfiguriert. Es wird aber noch nicht mit der Kommunikation begonnen (siehe 3.2.2).

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

<Source Register Number> Gibt die Nummer des ersten Registers des Beschreibungsblocks an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

Der Beschreibungsblock gibt das Protokoll und die Anzahl der Kommunikationseinheiten an. Eine Kommunikationseinheit spezifiziert die Registerblöcke, die übertragen werden sollen und die Adresse des Kommunikationspartners. Bis zu 10 solcher Kommunikationseinheiten können definiert werden. Dabei können mehrere unterschiedliche Registerblöcke mit einem Kommunikationspartner ausgetauscht werden.

Die Funktion kann nur bei nicht laufenden RemoteScan aufgerufen werden. Eine Umkonfiguration im laufenden Betrieb ist nicht möglich

3.2.1.1 Beschreibungsblock

Registeroffset	Beschreibung	
0	Protokoll	1 = JetWay 3 = JetIP 5 = Modbus/TCP
1	Anzahl nachfolgender Kommunikationseinheiten	1 .. 10
Kommunikationseinheit 1		
2	Adresse	
3	Portnummer	Modbus/TCP: 502
4	Update Rate	10 .. 65535 ms
5	Anzahl Ausgangsregister	Modbus/TCP: 0 .. 125
6	Ausgangs-Quellregisternummer	lokal Modbus/TCP: 66000 .. 66999
7	Ausgangs-Zielregisternummer	Remote
8	Anzahl Eingangsregister	Modbus/TCP: 0 .. 125
9	Eingangs-Quellregisternummer	Remote
10	Eingangs-Zielregisternummer	lokal Modbus/TCP: 66000 .. 66999
11	Nummer des ersten Registers des Statusregisterblocks	
12	Timeout	in Millisekunden

Bei Modulen ohne Eingangs- oder Ausgangsregister ist die entsprechende Anzahl auf 0 zu setzen.
Sind sowohl Ein- als auch Ausgänge konfiguriert, so werden zunächst die Ausgänge gesendet und danach die Eingänge gelesen.

3.2.1.2 Statusregisterblock

Im Beschreibungsblock einer jeden Kommunikationseinheit ist die Nummer des ersten Registers des aus 3 aufeinander folgenden Registern bestehenden Statusblocks anzugeben, in welchem, bei laufendem RemoteScan, die Fehlermeldungen dieser Kommunikationseinheit abgelegt werden.
Der Statusblock hat folgenden Aufbau:

<table>
<tr><th>Registeroffset</th><th>Bedeutung</th><th></th></tr>
<tr><td>0</td><td>Status
(bitcodiert)</td><td>
<table>
<tr><th>Bit-Nr</th><th>Bedeutung</th><th></th></tr>
<tr><td>0</td><td>Scan läuft</td><td>wird nach jedem Update Zyklus gesetzt</td></tr>
<tr><td>1</td><td>Fehler aufgetreten</td><td>wird bei jedem Auftreten eines Fehlers gesetzt</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Fehlerkennung</td><td>Es ist immer die Kennung des letzten aufgetretenen Fehlers zu lesen.
<table>
<tr><th>Registerinhalt</th><th>Bedeutung</th></tr>
<tr><td>0</td><td>kein Fehler</td></tr>
<tr><td>< 0</td><td>applikationsspezifischer Fehler Modbus/TCP:
<table>
<tr><th>Kennung</th><th>Bedeutung</th></tr>
<tr><td>-1</td><td>Fehler im Netzwerk-Treiber</td></tr>
<tr><td>-2</td><td>Fehler in der Verbindungsverwaltung</td></tr>
<tr><td>-3</td><td>Fehler beim Senden der Ausgangsregister</td></tr>
<tr><td>-4</td><td>Fehler beim Lesen der Eingangsregister</td></tr>
<tr><td>-5</td><td>Exception Rückmeldung</td></tr>
<tr><td>-6</td><td>Fehler beim Empfang der Rückmeldung</td></tr>
<tr><td>-7</td><td>Falsche 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>Fehler beimLesen/Schreiben der lokalen Register</td></tr>
<tr><td>103 / 104</td><td>Fehler in der unterlagerten Kommunikationsschicht</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>Fehleranzahl</td><td>wird bei jedem Auftreten eines Fehlers inkrementiert</td></tr>
</table>

Anmerkung: Es ist sinnvoll vor dem Starten des RemoteScan die Inhalte der Statusregisterblöcke mit 0 zu initialisieren.

3.2.1.3 Fehlermeldungen

Nach Ausführung der Funktion kann im Ergebnisregister die Rückmeldung gelesen werden.

Registerinhalt	Bedeutung
> 0	Anzahl der konfigurierten Kommunikationseinheiten
-1	nicht unterstütztes Protokoll
-2	eingestellte Anzahl Kommunikationseinheiten > 10
-3	ungültige Adresse oder Portnummer
-4	ungültige Registernummer
-10	RemoteScan läuft bereits

3.2.2 RemoteScan starten

Die Spezialfunktion 81 wird aufgerufen um einen – mittels Spezialfunktion 80 konfigurierten – RemoteScan zu starten.

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

<Source Register Number> Gibt die Nummer des Parameter-Registers an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

Dieser Funktion werden keine Parameter übergeben. Der Inhalt des Registers, auf das <Source Register Number> zeigt, ist somit unerheblich.
Die Funktion liefert stets den Ergebniswert 0 zurück.

3.2.3 RemoteScan stoppen

Mittels der Spezialfunktion 82 wird ein laufender RemoteScan gestoppt und alle eventuell geöffneten Kommunikationsverbindungen geschlossen.

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

<Source Register Number> Gibt die Nummer des Parameter-Registers an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

Dieser Funktion werden keine Parameter übergeben. Der Inhalt des Registers, auf das <Source Register Number> zeigt, ist somit unerheblich.
Die Funktion liefert stets den Ergebniswert 0 zurück.

Achtung: Die Ausführung dieser Funktion kann – abhängig von der Konfiguration – relativ lange dauern, da gewartet wird bis alle gerade laufenden Übertragungen abgeschlossen sind.

3.3 Register

RemoteScan Protokoll Register	
JC 647	**JC 24x**
63020	**2965**
Funktion	**Beschreibung**
Lesen	Protokoll 1 = JetWay 3 = JetIP 5 = Modbus/TCP
Schreiben	nicht möglich
Wertebereich	32 Bit
Wert nach Reset	0

RemoteScan Anzahl Einheiten	
JC 647	**JC 24x**
63021	**2966**
Funktion	**Beschreibung**
Lesen	Anzahl Kommunikationeinheiten
Schreiben	nicht möglich
Wertebereich	32 Bit
Wert nach Reset	0

RemoteScan Aktiv	
JC 647	**JC 24x**
63022	**2967**
Funktion	**Beschreibung**
Lesen	0 = nicht aktiv 1 = aktiv (läuft)
Schreiben	nicht möglich
Wertebereich	
Wert nach Reset	0

3.4 Beschreibung

Für die Verwendung des Kommunikationsprotokolls ‚Modbus/TCP' wird keine Lizenzdatei benötigt.
Dieses Protokoll ist ab der Version 3.04 verfügbar.

3.5 Register

Das bitcodierte Register 63827 wurde um 2 Bits erweitert

Register 63827: Webfunktionen		
Funktion	**Beschreibung**	
Lesen	Bit-Nummer	Bedeutung
	0	0 = FTP-Server nicht verfügbar 1 = FTP-Server verfügbar
	1	0 = HTTP-Server nicht verfügbar 1 = HTTP-Server verfügbar
	2	0 = E-Mail Funktion nicht verfügbar 1 = E-Mail Funktion verfügbar
	3	0 = Datendatei Funktion nicht verfügbar 1 = Datendatei Funktion verfügbar
	4	0 = kein Modbus/TCP 1 = Modbus/TCP lizenziert
	5	0 = Modbus/TCP Server nicht verfügbar 1 = Modbus/TCP Server gestartet
Schreiben	nicht möglich	
Wertebereich	0 .. 255	
Wert nach Reset	abhängig von der Initialisierung	

4 Server

4.1 Beschreibung

Nach dem Start des Modbus/TCP Servers kann durch einen externen Client auf Register, Eingänge und Ausgänge zugegriffen werden. Dabei können gleichzeitig 4 Verbindungen geöffnet sein.

4.2 Unterstützte Befehle

4.2.1 Registerzugriffe

Da über Modbus/TCP nur Register mit 16 Bit Breite übertragen werden, kann auf die höherwertigen 16 Bit der JetControl-Register nicht zugegriffen werden. Beim Empfang von Registerwerten erfolgt keine Vorzeichenerweiterung auf 32 Bit.

4.2.2 Class 0

4.2.2.1 read multiple registers (fc 3)

Registerblock lesen.
Startregisternummer entspricht der Registernummer innerhalb des JC-647.

4.2.2.2 write multiple registers (fc 16)

Registerblock schreiben.
Startregisternummer entspricht der Registernummer innerhalb des JC-647.

4.2.3 Class 1

4.2.3.1 read coils (fc 1)

Ausgänge lesen.
Die Ausgangsnummer ist in JC-647-interner Nummerierung zu übergeben.

JetSym - Anwendernummerierung	JC-647 – interner Nummerierung
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
Usw.	usw.

4.2.3.2 read input discretes (fc 2)

Eingänge lesen.
Die Eingangsnummer ist in JC-647-interner Nummerierung zu übergeben.

JetSym - Anwendernummerierung	JC-647 – interner Nummerierung
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
Usw.	usw.

4.2.3.3 read input registers (fc 4)

Eingänge in 16 Bit Worten zusammengefasst lesen.

Eingänge	Registernummer
101 .. 116	0
201 .. 216	2
301 .. 316	4
Usw.	usw.

4.2.3.4 write coil (fc 5)

Ein-/Ausschalten eines einzelnen Ausgangs.
Die Ausgangsnummer ist in JC-647-interner Nummerierung zu übergeben.

JetSym - Anwendernummerierung	JC-647 – interner Nummerierung
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
Usw.	usw.

4.2.3.5 write single register (fc 6)

Beschreiben der niederwertigen 16 Bit eines JC-647 Registers.

4.2.4 Class 2

4.2.4.1 force multiple coils (fc 15)

Ein-/Ausschalten mehrerer Ausgänge.
Die Ausgangsnummer ist in JC-647-interner Nummerierung zu übergeben.

JetSym - Anwendernummerierung	JC-647 – interner Nummerierung
101 .. 164	0 .. 0x3F
201 .. 264	0x40 .. 0x7F
Usw.	usw.

4.2.4.2 read / write registers

Schreiben und gleichzeitiges Lesen von Registern.
Startregisternummer entspricht der Registernummer innerhalb des JC-647. Es werden zunächst die angeforderten Register gelesen danach die übertragenen Register beschrieben.

5 Client

5.1 Allgemein

Der Modbus/TCP Client im JetControl 647 unterstützt Class 0 Conformance (siehe 4.2.2). Das heißt, dass die Befehle zum Lesen und Schreiben von mehreren Registern genutzt werden. In einem Telegramm können bis zu 125 Register mit 16 Bit Breite übertragen werden. Beim Senden von 32 Bit Registern werden lediglich die niederwertigen 16 Bit übertragen. Beim Zuweisen von empfangenen Registerwerten auf die JetControl-internen 32 Bit Register wird keine Vorzeichenerweiterung durchgeführt.
Als Protokoll-ID wird ‚0' verwendet, als Unit-ID eine ‚1'. Die Zuordnung der gesendeten und empfangenen Telegramme erfolgt über die Transaction-ID.
Es können gleichzeitig Verbindungen zu 11 unterschiedlichen Servern geöffnet sein.

5.2 RemoteScan

Hierüber werden zyklisch die in den 16 Bit-Registern 66000 bis 66999 zusammengefassten Ein- und Ausgänge von 14001 bis 19999 von und zu den konfigurierten Servern übertragen. Zu jedem Server (IP-Adresse und Port) wird eine Verbindung aufgebaut, unabhängig davon wie viele Kommunikationseinheit auf diesen Server konfiguriert sind.
Sind mehrere Kommunikationseinheiten auf einen Server konfiguriert, so werden die Zugriffe serialisiert, da die Server in der Regel kein „command pipelining" unterstützen. Sind mehrere Server konfiguriert, so wird parallel mit ihnen kommuniziert.

5.2.1 Register – E/A-Überlagerung

JC 24x		JC 647	
Register	Ein- / Ausgänge	Register	Ein- / Ausgänge
8000	20001 – 20016	66000	14001 – 14016
8001	20017 – 20032	66001	14017 – 14032
8002	20033 – 20048	66002	14033 – 14048
Usw.	usw.	Usw.	usw.
8999		66375	19999
		66998	Keine Zuordnung
		66999	Keine Zuordnung

Da es sich bei den Register und ihren überlagerten Ein- und Ausgängen lediglich um Speicherzellen im RAM handelt, jedoch keine direkte Abbildung auf Hardware stattfindet, ist nicht festgelegt, ob ein Register Eingänge oder Ausgänge enthält. Erst bei der Konfiguration in den Kommunikationseinheiten findet eine Zuordnung statt.

Formel:

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

Z.b:
IONo 14033 ergibt Register 66002 BitNo 0

5.3 Spezialfunktionen

Als azyklischer Übertragungskanal zu einem Modbus/TCP Server können die Spezialfunktionen 65 (Register lesen) und 66 (Register schreiben) benutzt werden (die Funktionen sind unabhängig vom RemoteScan verfügbar).
Während eine dieser beiden Spezialfunktionen ausgeführt wird, werden parallele Aufrufe dieser Funktionen in anderen Tasks blockiert, bis die Funktion beendet ist.
Die Funktionen öffnen eine Verbindung zu dem angegebenen Server, übertragen die gewünschten Daten und schließen die Verbindung wieder. Bei einer bereits durch den RemoteScan bestehenden Verbindung wird diese benutzt und der Verbindungsauf- und abbau entfällt.
Achtung: Während eine dieser Funktionen ausgeführt wird, sollte kein TaskBreak oder TaskRestart auf diese Task oder ein Programm-Neustart mittels JetSym durchgeführt werden, da hierbei die Verbindung geöffnet bleibt und somit unter Umständen keine weiteren Übertragungen stattfinden können.

5.3.1 Register lesen

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

<Source Register Number> Gibt die Nummer des ersten Registers eines Beschreibungsblocks an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

Beschreibungsblock		
Registeroffset	Bedeutung	
0	IP-Adresse	
1	Portnummer	502
2	Timeout	in Millisekunden
3	Quellregisternummer	remote
4	Zielregisternummer	lokal
5	Anzahl Register	1 .. 125

Ergebnis	Bedeutung
0	kein Fehler
-1 oder -2	Fehler beim Verbindungsaufbau
-4	Fehler bei Datenübertragung
-5	Fehlermeldung vom Server
-8	Timeout
-10	Modbus/TCP nicht lizenziert

5.3.2 Register schreiben

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

<Source Register Number> Gibt die Nummer des ersten Registers eines Beschreibungsblocks an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

Beschreibungsblock		
Registeroffset	Bedeutung	
0	IP-Adresse	
1	Portnummer	502
2	Timeout	in Millisekunden
3	Quellregisternummer	lokal
4	Zielregisternummer	remote
5	Anzahl Register	1 .. 125

Ergebnis	Bedeutung
0	kein Fehler
-1 oder -2	Fehler beim Verbindungsaufbau
-3	Fehler bei Datenübertragung
-5	Fehlermeldung vom Server
-8	Timeout
-10	Modbus/TCP nicht lizenziert

5.3.3 ST Beispiel Programm

```
// ********************************************************************************
// *** 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 sFehler
        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$');
        GOTO sFehler
        end_if;
        when nRemoteScanCfgResult > 0 continue;
        nRemoteScanCfgResult := 22;          // set it to a value != zero
        SYSTEMFUNCTION(cStartRemoteScan,             &stRemoteScanCfgHdr,
&nRemoteScanCfgResult);
        if NOT (nRemoteScanCfgResult = 0)
        then
        // Result 0
        DISPLAY_TEXT(0, zPosNFText,'Modbus E003$');
        GOTO sFehler
        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 sFehler:
goto sFehler;
end_task
```

6 Spezialfunktion 60 und 61 Modbus CRC

6.1 Modbus RTU CRC-Prüfsumme

Zur Generierung und Überprüfung der Modbus RTU CRC-Prüfsumme dienen die Spezialfunktionen 60 und 61.
Es wird davon ausgegangen, dass die Zeichen des Modus-Telegramms nacheinander in Registern abgelegt sind.
Achtung: abweichend zu den anderen Spezialfunktionen, bei denen der erste Funktionsparameter ein Register mit den Eingangsdaten und der zweite Parameter ein Register für die Ergebnisdaten der Funktion spezifiziert, gibt bei den Modbus CRC Routinen der erste Parameter den Beginn, der zweite Parameter das Ende des Modbus-Telegramms an.
Der Aufruf der Spezialfunktionen erfolgt mit direkter oder indirekter Registerübergabe, z.B.:

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

6.1.1 Funktion 60: Berechnung der Prüfsumme

Funktionsweise Diese Spezialfunktion berechnet aus einem Telegramm eine zwei Byte grosse Prüfsumme und fügt die zwei Byte an das Ende des Telegrammes an.
Parameter 1 Nummer des Registers mit dem ersten Datum des Modbus Protokolls.
Parameter 2 Nummer des Registers mit dem letzten Datum des Modbus Protokolls ohne die zwei Bytes für die CRC-Prüfsumme.
mögliche Fehler – die Nummer des letzten Registers ist kleiner als die Nummer des ersten Registers;
– jedes Register darf nur in den untersten 8 Bit Nutzdaten enthalten.
Ergebnis im Fehlerfall undefiniert.
Rechenzeit ca. 68 µs bei einer Datenlänge von 100 Registern.

6.1.2 Funktion 61: Überprüfung der Prüfsumme

Funktionsweise Diese Spezialfunktion überprüft aus einem Telegramm die Prüfsumme und fügt das Ergebnis an das Ende des Telegrammes an.
Parameter 1 Nummer des Registers mit dem ersten Datum des Modbus Protokolls.
Parameter 2 Nummer des Registers mit dem letzten Datum des Modbus Protokolls mit den zwei Bytes für die CRC-Prüfsumme.
mögliche Fehler – die Nummer des letzten Registers ist kleiner als die Nummer des ersten Registers;
– jedes Register darf nur in den untersten 8 Bit Nutzdaten enthalten.
Ergebnis im Fehlerfall undefiniert.
Rechenzeit ca. 68 µs bei einer Datenlänge von 100 Registern.

6.1.3 Beispielprogramm

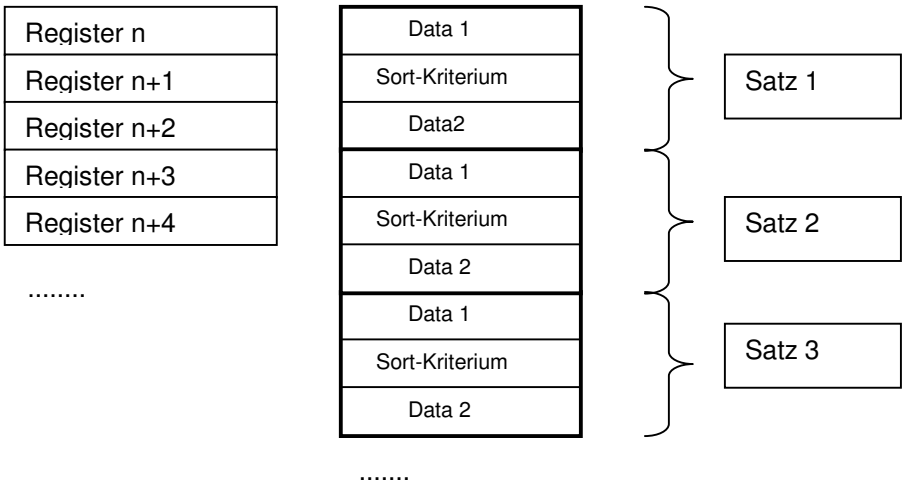
Im folgenden Beispiel wird mit der Spezialfunktion 61 CRC-Prüfsumme eines empfangenen Modbus RTU Telegramms überprüft. Bei richtiger Prüfsumme wird eine 1, bei nicht richtiger Prüfsumme eine 0 an das empfangene Telegramm angefügt.

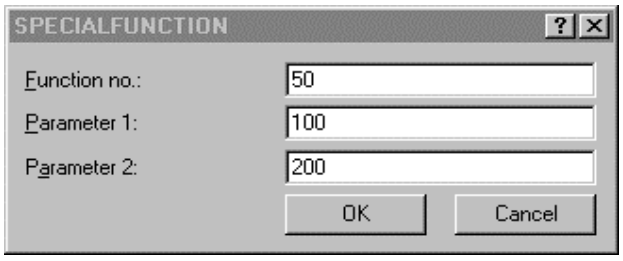
```
REGISTER_LOAD (100, 0x02)            // Slave Adresse
REGISTER_LOAD (101, 0x03)            // Funktionscode
REGISTER_LOAD (102, 0x00)            // Startnummer
REGISTER_LOAD (103, 0x20)            // Startnummer
REGISTER_LOAD (104, 0x00)            // Anzahl
REGISTER_LOAD (105, 0x04)            // Anzahl
REGISTER_LOAD (106, 0x45)            // CRC-Prüfsumme
REGISTER_LOAD (107, 0xF0)            // CRC-Prüfsumme
SPECIALFUNCTION (61, 100, 107)       // CRC-Prüfsumme überprüfen
// das Ergebnis steht in Register 108
IF REG 108 = 1
THEN ...                             // CRC-Prüfsumme korrekt
ELSE ...                             // CRC-Prüfsumme nicht korrekt
```

7 Spezialfunktion 50 Sortieren von Daten

7.1 Einführung

Um Daten in der Steuerung zu sortieren musste bisher der Sortieralgorithmus in Sympas bzw. ST geschrieben werden. Das Sortieren hatte den Vorteil, dass jede Source Zeile bekannt war, jedoch den Nachteil, dass die Sortier-Performance nicht optimal war.
Um für den Anwender eine größtmögliche Flexibilität zu eröffnen wird der Sortieralgorithmus und die Daten voneinander getrennt. Der Sortieralgorithmus ist im Betriebssystem der Steuerung abgelegt. Die zu sortierenden Daten werden mit dem Parameter 1 indirekt Adressiert. Indirekt deshalb, weil der Wert des Parameters 2 ein Register angibt, wo der Descriptor abgelegt ist. Der Descriptor enthält ein Register (Offset 0) welches auf die Daten zeigt. Weiter die Art und Weise des Sortierens usw. siehe Input Descriptor (Parameter 1) Der Descriptor ist notwendig, da die Spezialfunktionen nur 2 Übergabeparameter haben. Der Parameter 2 gibt eine Registeradresse an wo ein Fehlercode und die Verarbeitungszeit abgelegt wird. Durch den Aufruf der Spezialfunktion 50 können die über den Parameter 1 angegebenen Daten sortiert werden.





SPECIALFUNCTION (50, 100, 200)

Der Parameter 1 gibt den Zeiger auf den Input Descriptor an.
Der Parameter 2 gibt den Zeiger auf das Ergebnis an.

7.1.1 Input Descriptor (Parameter 1)

Der Parameter 1 gibt an wo der Descriptor im Speicher der Steuerung abgelegt wird. Dieser Descriptor muß vor Aufruf der Funktion mit sinnvollen Werten initialisiert werden.

Offset	Parameter	Wertebereich	Beschreibung
0	Daten Start Register	0-20479	Start Register, welches die zu Sortierenden Daten enthält
1	Anzahl Datensätze	2-1000	Anzahl Datensätze. Max. 1000
2	Datenlänge	2-1000	Anzahl Register je Datensatz
3	Sortier Element	0-999	Element innerhalb eines Datensatzes, welches als Sortier Kriterium dient
4	Modus	0-1	Gibt an wie sortiert werden soll, aufsteigend, absteigend... Bitposition 0: gelöscht: Aufsteigend; gesetzt: Absteigend 1: frei 2: frei 3: frei
5	Frei	Nicht benützt	Für spätere Erweiterungen vorgesehen
6	Frei	Nicht benützt	Für spätere Erweiterungen vorgesehen
7	Frei	Nicht benützt	Für spätere Erweiterungen vorgesehen
8	Frei	Nicht benützt	Für spätere Erweiterungen vorgesehen
9	Frei	Nicht benützt	Für spätere Erweiterungen vorgesehen

7.1.2 Rückgabewerte (Parameter 2)

Offset	Parameter	Beschreibung
0	**Fehlercode**	**Wie unter Fehlercode (Parameter 2) Offset 0 beschrieben**
1	**Ausführungszeit in [us]**	**Berechnungszeit für das sortieren der Daten**
2	**Frei**	**Für spätere Erweiterungen**
3	**Frei**	**Für spätere Erweiterungen**

7.1.3 Fehlercode (Parameter 2) Offset 0

Der Parameter 2 Offset 0 enthält nach dem Aufruf der Funktion den Rückgabewert.

Rückgabewert	Bedeutung
0	**OK, kein Fehler aufgetreten**
1000	**Start Register > 20479**
2000	**Ziel Register > 20479**
3000	**Anzahl Datensätze > 240**
4000	**Datenlänge > 1000**
5000	**Sortier Modus > 256**
6000	**Sortier Element > (Datenlänge-1)** **Sortier Element liegt ausserhalb vom gültigen Bereich.**

7.1.4 Ausführungszeit (Parameter 2) Offset 1

In diesem Register wird die verbrauchte Zeit für das Sortieren abgelegt.

7.2 Beispiel Datenanordnung

Parameter 1 der Spezialfunktion 50 = 100
Parameter 2 der Spezialfunktion 50 = 200

Register	Offset	Parameter/Beschreibung	Wert
100	0	Daten Start Register	400
101	1	Anzahl Datensätze	3
102	2	Anzahl Elemente je Datensatz	4
103	3	Master Element_3 wird benützt	2
104	4	Modus	0
105	5	Frei	
106	6	Frei	
107	7	Frei	
108	8	Frei	

109	9	Frei	

Register	Beschreibung
400	**DatenSatz[0].Element_1**
401	**DatenSatz[0].Element_2**
402	**DatenSatz[0].Element_3**
403	**DatenSatz[0].Element_4**
404	**DatenSatz[1].Element_1**
405	**DatenSatz[1].Element_2**
406	**DatenSatz[1].Element_3**
407	**DatenSatz[1].Element_4**
408	**DatenSatz[2].Element_1**
409	**DatenSatz[2].Element_2**
410	**DatenSatz[2].Element_3**
411	**DatenSatz[2].Element_4**

8 Beispielprogramm

```
//*********************************************************************************************
//********************** Specialfunktion 50 Sortieren von Daten direkt/direkt ******
//*********************************************************************************************

 REGISTER_LOAD (100, 3000)                    // Data start Register
 REGISTER_LOAD (101, 4)                       // Anzahl Datensätze
 REGISTER_LOAD (102, 2)                       // Datenlänge
 REGISTER_LOAD (103, 0)                       // Sortier Kriterium: Element hier
0-1
 REGISTER_LOAD (104, 0)                       // Modus: Bit 0  = 0 Register 3000
enthält den kleinsten Wert
                                              // Modus: Bit 0  = 1 Register 3000
enthält den größten Wert
 REGISTER_LOAD (105, 0)                       // on the moment not used
 REGISTER_LOAD (106, 0)                       // on the moment not used
 REGISTER_LOAD (107, 0)                       // on the moment not used
 REGISTER_LOAD (108, 0)                       // on the moment not used
 REGISTER_LOAD (109, 0)                       // on the moment not used

 // Daten generieren
```

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


 // Default Werte beschreiben zur Fehler Erkennung

 REGISTER_LOAD (200, 55555)                    // Offset 0: Return Wert
beschreiben
 REGISTER_LOAD (201, 55555)                    // Offset 1: Sortierzeit in
Microsekunden 100 DatenSätze in ca.
                                               //        4000 Microsekunden
 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, "SortFkt d/d 1")
        GOTO sFehler
ELSE

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


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

9 Startverzögerung

9.1 Beschreibung

Wird der JetControl 647 gestartet (mit 24 V Versorgt), so läuft eine in Register 63892 einstellbare Verzögerungszeit ab, bevor die Module am Systembus initialisiert werden und bevor das Anwenderprogramm gestartet wird.
Während diese Zeit abläuft, leuchtet nur die rote LED (ERR). Der Wert in Register 63892 legt die Verzögerungszeit in Vielfachen von 100 ms fest. Der Minimalwert ist 0 (Funktion abgeschaltet). Der Maximalwert ist 300 (also 30 Sekunden).

10 Datendateien

10.1 Allgemein

Mit Hilfe einiger Spezialfunktionen ist es möglich aktuelle Registerwerte und Merkerzustände, gesteuert durch das Anwenderprogramm, in eine Datei zu schreiben oder Werte aus einer vorhandenen Datei zu lesen.
Das Dateiformat ist identisch zu den von JetSym erzeugten ‚Data Dump'-Dateien (siehe A.1).
Die Dateinamen setzen sich aus zwei konstanten Teilen und dem Inhalt eines Registers zusammen, so dass die Dateien durch unterschiedliche Registerwerte ausgewählt werden können (siehe 10.2.2).
Geschriebene Dateien werden im ‚root'-Verzeichnis abgelegt. Zu ladende Dateien müssen ebenfalls im ‚root'-Verzeichnis abgelegt sein. Der Zugriff auf die Datendateien erfolgt mit Administrator-Rechten und kann nicht eingeschränkt werden.

10.2 Spezialfunktionen

10.2.1 Implementierung

Da Dateioperationen – besonders bei großen Dateien – relativ lange dauern können, werden, während eine dieser Funktionen ausgeführt wird, die anderen Anwendertasks bearbeitet. Da jedoch immer nur eine Funktion bearbeitet werden kann, werden Tasks, die eine dieser Funktionen aufrufen während eine Dateioperation einer anderen Task gerade läuft, so lange blockiert, bis die gerade laufende Funktion abgeschlossen ist.
Daraus ergibt sich ebenfalls, dass Datenkonsistenz über die zu schreibenden oder zu lesenden Werteblöcke nicht garantiert werden kann. Diese ist gegebenenfalls durch entsprechende Programmierung im Anwenderprogramm herzustellen.
Der Zustand der gerade laufenden Bearbeitung kann über die unten angegebenen Register (siehe 10.3) abgefragt werden.

10.2.2 Dateinamen

Die Dateinamen beginnen stets mit ‚Data_', gefolgt von einem Zahlenwert und der Dateinamenserweiterung ‚da'. Der Zahlenwert zur Unterscheidung verschiedener Dateien wird aus dem Parameterregister der Spezialfunktionen gewonnen.
Beispiele:
Data_123456789.da
Data_0.da

Zur Erinnerung: Auf Groß-/Kleinschreibung achten. Das Dateisystem arbeitet ‚case sensitive'.

10.2.3 Werte abspeichern – Datei erstellen

Mit Hilfe der Spezialfunktion **90** wird eine neue Datendatei angelegt und ein wählbarer Register- oder Merkerblock in diese Datei geschrieben.

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

<Source Register Number> Gibt die Nummer des ersten Registers des Parameterblocks an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

10.2.3.1 Parameterblock

Ab dem Register <Source Register Number> werden die Funktionsparameter angegeben.

Registeroffset	Bedeutung	
0	„Dateinamen“	numerischer Teil des Dateinamens (siehe 10.2.2)
1	Typ	1 = Register 3 = Merker
2	Beginn Datenblock	Nummer des ersten Registers oder Merkers
3	Ende Datenblock	Nummer des letzten Registers oder Merkers

10.2.3.2 Funktionsergebnis

Im Register <Destination Register Number> kann das Funktionsergebnis gelesen werden.

Registerinhalt	Bedeutung
0	kein Fehler
-1	Fehler beim Anlegen der Datei (z.B. Disk voll)
-2	Fehler beim Schreiben der Daten
-4	Fehler beim Schließen der Datei
-6	ungültige Register-/Merkernummer
-10	Datendatei-Funktionen nicht verfügbar (siehe 10.3)
-20	interner Betriebssystemfehler

10.2.4 Werte abspeichern – an Datei anfügen

Mit Hilfe der Spezialfunktion **91** wird ein wählbarer Register- oder Merkerblock an eine bestehende Datei angefügt. Sollte die Datei nicht vorhanden sein, so wird sie neu angelegt.

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

<Source Register Number> Gibt die Nummer des ersten Registers des Parameterblocks an.
<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

10.2.4.1 Parameterblock

Ab dem Register <Source Register Number> werden die Funktionsparameter angegeben.

Registeroffset	Bedeutung	
0	„Dateinamen“	numerischer Teil des Dateinamens (siehe 10.2.2)
1	Typ	1 = Register 3 = Merker
2	Beginn Datenblock	Nummer des ersten Registers oder Merkers
3	Ende Datenblock	Nummer des letzten Registers oder Merkers

10.2.4.2 Funktionsergebnis

Im Register <Destination Register Number> kann das Funktionsergebnis gelesen werden.

Registerinhalt	Bedeutung
0	kein Fehler
-1	Fehler beim Öffnen oder Anlegen der Datei
-2	Fehler beim Schreiben der Daten
-4	Fehler beim Schließen der Datei
-6	ungültige Register-/Merkernummer
-10	Datendatei-Funktionen nicht verfügbar (siehe 10.3)
-20	interner Betriebssystemfehler

10.2.5 Werte aus Datei einlesen

Mit Hilfe der Spezialfunktion **92** werden Registerwerte und Merkerzustände aus einer Datendatei gelesen und die entsprechenden Register, beziehungsweise Merker, damit beschrieben. Die Abarbeitung erfolgt in der durch den Dateiinhalt vorgegebenen Reihenfolge.

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

<Source Register Number> Gibt die Nummer des Registers an, in dem der numerische Teil des Dateinamens enthalten ist.

<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

10.2.5.1 Funktionsergebnis

Im Register <Destination Register Number> kann das Funktionsergebnis gelesen werden.

Registerinhalt	Bedeutung
0	kein Fehler
-1	Fehler beim Öffnen der Datei (z.B. Datei nicht gefunden)
-3	Fehler beim Lesen der Daten
-4	Fehler beim Schließen der Datei
-10	Datendatei-Funktionen nicht verfügbar (siehe 10.3)
-20	interner Betriebssystemfehler

10.2.6 Datei löschen

Mit Hilfe der Spezialfunktion **96** wird eine Datendatei von der Flash-Disk gelöscht.

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

<Source Register Number> Gibt die Nummer des Registers an, in dem der numerische Teil des Dateinamens enthalten ist.

<Destination Register Number> Gibt die Nummer des Ergebnisregisters dieser Funktion an.

10.2.6.1 Funktionsergebnis

Im Register <Destination Register Number> kann das Funktionsergebnis gelesen werden.

Registerinhalt	Bedeutung
0	kein Fehler
-5	Fehler beim Löschen der Datei (z.B. Datei nicht gefunden)
-10	Datendatei-Funktionen nicht verfügbar (siehe 10.3)
-20	interner Betriebssystemfehler

10.3 Register

10.3.1 Verfügbarkeit

Das bitcodierte Register 63827 wurde um 1 Bit erweitert

<table>
<tr><th colspan="3">Register : Webfunktionen</th></tr>
<tr><th colspan="2">JC 647</th><th>JC 24x</th></tr>
<tr><th colspan="2">63827</th><th>2930</th></tr>
<tr><th>Funktion</th><th colspan="2">Beschreibung</th></tr>
<tr><td>Lesen</td><td colspan="2"><table>
<tr><th>Bit-Nummer</th><th>Bedeutung</th></tr>
<tr><td>0</td><td>0 = FTP-Server nicht verfügbar
1 = FTP-Server verfügbar</td></tr>
<tr><td>1</td><td>0 = HTTP-Server nicht verfügbar
1 = HTTP-Server verfügbar</td></tr>
<tr><td>2</td><td>0 = E-Mail Funktion nicht verfügbar
1 = E-Mail Funktion verfügbar</td></tr>
<tr><td>3</td><td>0 = Datendatei Funktion nicht verfügbar
1 = Datendatei Funktion verfügbar</td></tr>
<tr><td>4</td><td>0 = kein Modbus/TCP
1 = Modbus/TCP lizenziert</td></tr>
<tr><td>5</td><td>0 = Modbus/TCP Server nicht verfügbar
1 = Modbus/TCP Server gestartet</td></tr>
</table></td></tr>
<tr><td>Schreiben</td><td colspan="2">nicht möglich</td></tr>
<tr><td>Wertebereich</td><td colspan="2">0 .. 255</td></tr>
<tr><td>Wert nach Reset</td><td colspan="2">abhängig von der Initialisierung</td></tr>
</table>

10.3.2 Statusregister

Der Zustand der Bearbeitung einer Dateioperation und die Nummer der Task, die diese Operation gerade ausführt kann in zwei Registern gelesen werden.

Register : Bearbeitungszustand	
JC 647	**JC 24x**
63835	**2977**
Funktion	**Beschreibung**
Lesen	0: keine Dateioperation in Bearbeitung 1: Bearbeitung an Datei-Modul übergeben 2: Daten werden gelesen/geschrieben 3: Dateioperation abgeschlossen
Schreiben	nicht erlaubt
Wertebereich	0 .. 255
Wert nach Reset	0

Register : Tasknummer	
JC 647	**JC 24x**
63836	**2978**
Funktion	**Beschreibung**
Lesen	Nummer der Task die gerade eine Dateioperation durchführt 0 .. 99: Tasknummer 255: keine Task
Schreiben	nicht erlaubt
Wertebereich	0 .. 255
Wert nach Reset	255

11 Anhang

A.1 Dateiformat

Es handelt sich um eine reine Textdatei, wobei jeder Eintrag in einer neuen Zeile stehen muss und eine Zeile mit ‚carriage return / line feed' abgeschlossen ist. Kommentarzeilen sind möglich.
Als erster Eintrag einer Datendatei muss die Kennung ‚SD1001' stehen.
Datenzeilen beginnen mit einer Variablenkennung. Danach folgt, in der selben Zeile durch Leerzeichen oder Tabulator getrennt, die Variablennummer. Gefolgt, ebenfalls in der selben Zeile durch Leerzeichen oder Tabulator getrennt, der Variablenwert.
Die Kennungen am Zeilenbeginn dürfen nicht eingerückt sein.

Variablenkennung	Variablentyp
FS	Merker
QS	Fließkommazahl-Register
RS	Ganzzahl-Register

Alle Zeilen, die nicht mit einer dieser Variablenkennungen beginnen, werden, mit Ausnahme der ersten Zeile mit der Dateikennung, als Kommentarzeilen interpretiert.

Beispiel:

```
SD1001
; JC-647 DATA FILE - Jetter AG
FS      111     1
dies ist Kommentar
RS      20112   110
  FS    113     1
QS      65024   -3.141593
QS      65025   6.789e-05
```

Die dritte Zeile von unten ist ebenfalls eine Kommentarzeile, da die Variablenkennung (‚FS') nicht am Beginn der Zeile steht.