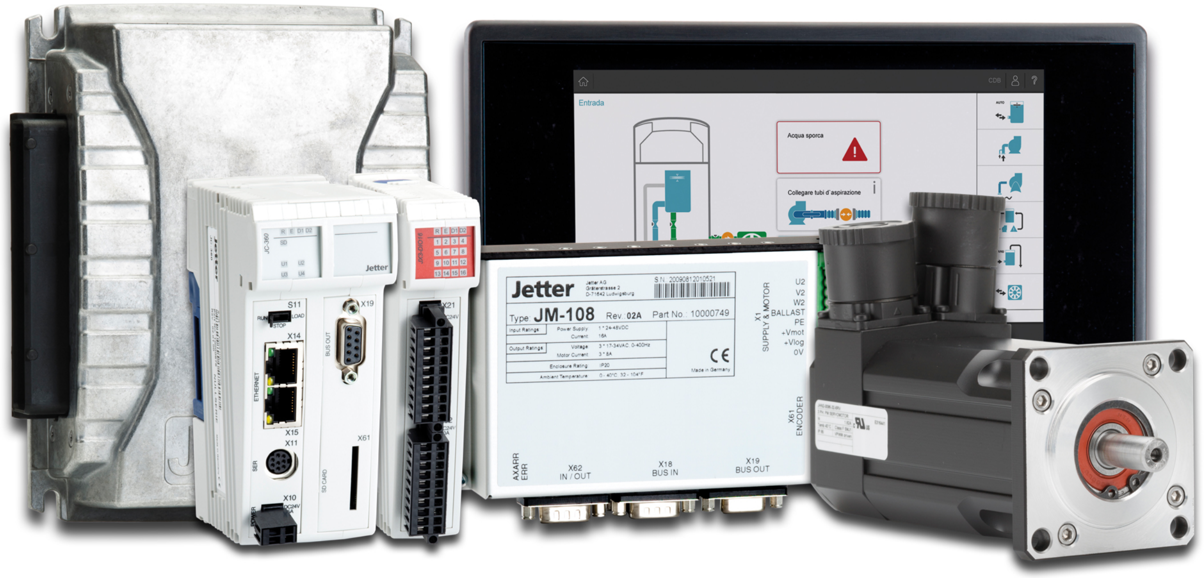
Jetter automation



JC-350

Version Update from V. 1.24 to V. 1.28

We automate your success.

Revision 1.02

September 2017 / Printed in Germany

Table of Contents

1 Introduction

Introduction

This chapter shows the history of OS versions.

OS update - Why?

An OS update lets you enhance the functionality of your device by

- adding new functions
- fixing software bugs
- installing an OS of a specific version after its release

Contents

Operating system update

OS file for operating system update

For an OS update, you will need the following file:

OS file	Description
JC-350_1.28.0.00.os	OS file for JC-350 with version 1.28

Downloading an operating system

You can download operating systems from the Jetter AG **homepage https://www.jetter.de/en/downloads.html**. There, the OS files for download can be found in the respective product category.

OS update by means of JetSym

To update the OS, proceed as follows:

Step	Action
1	Download the OS file from www.jetter.de.
2	Establish a connection between PC and controller.
3	In JetSym: Select menu item "Build -> Update OS" or Click on the button "OS Update" in the CPU window of the Hardware Manager.
4	Select the OS file.
5	Start the OS update by clicking OK.
6	**Result:** Following Power OFF/Power ON the new OS is launched.

Minimum requirements

For programming a JC-350 with version 1.28, JetSym 5.3.0 or higher is required.

JC-350 version update - Overview

V. 1.04

The following table gives an overview of newly added features and fixed software bugs in OS version 1.04:

Function	New	Fixed
JX2 system bus:		
Register overlaying for digital inputs/outputs	✓	
Support of JX-SIO modules and third-party CANopen® devices	✓	
JX3 system bus:		
Register overlaying for digital inputs/outputs	✓	
System bus special registers for status and control	✓	
OS update:		
Via FTP: On completion notification the OS has actually been stored.		✓
Updating a JX2 slave module while registers are being accessed blocks communication.		✓
Application program:		
Task switch could fail to happen.		✓
Error signal in case of invalid file **\App\start.ini**		✓
Display commands:		
Redirection to JX2-SER1 works only if JX2-PRN1 has been configured, too.		✓

V. 1.05

The following table gives an overview of newly added features and fixed software bugs in OS version 1.05:

Function	New	Fixed
JX2 system bus: V1.05.0.00		
AS interface gateway BWU1821 is supported	✓	
Frequency inverter 8200 vector is supported.	✓	
JetMove 1xx is not detected during boot process.		✓
Automatic baud rate recognition does not work reliably for some of the baud rates and configurations of IP67 modules.		✓
Repetition counter does not work when polling I/O modules		✓
AutoCopy function:		
Automatic copying of controller data		
Application program:	✓	
Pending cyclic tasks are started immediately after Taskunlock.	✓	
For function pow(x,y) a floating point number can be entered as exponent	✓	

Function	New	Fixed
Cyclic tasks can be debugged	✓	
Length of project and program names > 39 characters		✓
Restart of an elapsed timer		✓
The value returned by DateTimeDecode() was always 1 day short of the actual day.		✓
DateTimeEncode and -IsValid might return the value TRUE irrespective of an invalid date		✓
Application registers:		
The register type can be set up without having to start the application program	✓	
Displays and HMIs:		
A floating point value can be used as default for UserInput	✓	
The default value for UserInput is not displayed correctly		✓
It is not possible to enter LED register numbers		✓

V. 1.08

The following table gives an overview of newly added features and fixed software bugs in OS version 1.08:

Function	New	Fixed
System configuration:		
System rights for configuration file	✓	
JX2 system bus: V1.11.0.00		
Timeout after CAN PRIM message		✓
Registers of LJX7-CSL modules		✓
Write access to analog outputs of CANopen® modules		✓
State of digital inputs when the controller is powered on		✓
Digital outputs on JX-SIO or CANopen® modules		✓
Input/output 64 on JX-SIO or CANopen® modules		✓
User-programmable CAN Interface		✓
Application program:		
NetCopyList functions	✓	
StrCopy()		✓
Crash in the case of "invalid" application program		✓
NetCopyVarFromReg()		✓
JX3 system bus:		
Module registers for digital I/Os	✓	
Displays and HMIs:		
UserInput()		✓

V. 1.09

The following table gives an overview of newly added features and fixed software bugs in OS version 1.09:

Function	New	Fixed
System:		
System command register	✓	
JX2 system bus: V1.13.0.00		
Status change of inputs on JX2-ID8		✓
Status change of fast inputs		✓
Application program:		
FTP client	✓	
Axis instructions		✓
Taskrestart in the case of Delay()		✓
Crash in the case of missing library		✓
Floating-point number registers in data files		✓
NetCopyVarToReg with floating-point number registers		✓
JX3 system bus:		
Dummy modules	✓	
AutoCopy:		
FTP commands	✓	
Serial interface:		
Initialization after booting		✓

V. 1.10

The following table gives an overview of newly added features and fixed software bugs in OS version 1.10:

Function	New	Fixed
System:		
LED registers		✓
SD memory card		✓
JX2 system bus: V1.17.0.00		
Further modules	✓	
CAN PRIM	✓	
Application program:		
Task instructions using variable parameters	✓	
UserInput()		✓
NetCopyListSend()		✓
Task state register		✓

Function	New	Fixed
Real-time clock:		
Additional register for milliseconds	✓	
User-programmable IP interface:		
More connections	✓	

V. 1.12

The following table gives an overview of newly added features and fixed software bugs in OS version 1.12:

Function	New	Fixed
System:		
System command register	✓	
JX2 system bus: V1.21.0.00		
Initialization		✓
CAN PRIM		✓
CANopen® SYNC interval		✓
CANopen® application registers		✓
CANopen® type "String"		✓
Set CANopen® output		✓
CANopen® version number		✓
Wago 750		✓
JX3 system bus:		
Register accesses		✓
Application program:		
Program Control	✓	✓
Structures are assigned.	✓	
Sorting data	✓	
Variables are displayed in JetSym		✓
HTTP server:		
New file type	✓	
Serial interface:		
Error detection		✓

V. 1.14

The following table gives an overview of newly added features and fixed software bugs in OS version 1.14:

Function	New	Fixed
JX2 system bus: V1.22.0.00		
Operating system update		✓
Application program:		
New instructions	✓	

V. 1.16

The following table gives an overview of newly added features and fixed software bugs in OS version 1.16:

Function	New	Fixed
JX2 system bus: V1.23.0.00		
CANopen® registers		✓
Application program:		
New data types	✓	
New features	✓	
Memory protection	✓	
Cyclic tasks		✓
NetCopyVarToReg		✓
Cycle time register		✓

V. 1.18

The following table gives an overview of newly added features and fixed software bugs in OS version 1.18:

Feature	New	Fixed
Application program:		
Debugging	✓	
Memory protection	✓	✓
Partial download		✓
StrFormat()		✓
Exceptions		✓
Ethernet system bus:		
Enhanced diagnostic functions	✓	
Module support		✓
JX3 system bus:		
Register	✓	
Initialization		✓
OS update		✓

Feature	New	Fixed
STX debug server:		
TCP connection management	✓	

V. 1.22

The following table gives an overview of newly added features and fixed software bugs in OS version 1.22:

Feature	New	Fixed
Application program:		
Debugging	✓	
New functions	✓	
Tasklock	✓	
Initializing of variables	✓	
Partial download		✓
Exceptions		✓
Ethernet system bus:		
IP address setting	✓	
JetSync blockage	✓	
JX2 system bus:		
Error counter/error bits	✓	
CAN PRIM	✓	
Error indication		✓
Register for overlaying of outputs		✓
DNS client:		
IP address of DNS server	✓	
Diagnostics	✓	

V. 1.24

The following table gives an overview of newly added features and fixed software bugs in OS version 1.24:

Function	New	Fixed
Application program:		
Task processing (system command 170/171)	✓	
Taskcontinue	✓	
Ethernet system bus:		
NetConsistency	✓	
System:		
Start delay	✓	
IP configuration	✓	

V 1.28

The following table gives an overview of newly added features and fixed software bugs in OS version 1.28:

Feature	New	Fixed
Application program:		
Memory management	✓	
New features	✓	
Communication:		
STX debug server	✓	✓
JetIP server	✓	
ARP	✓	
NetConsistency	✓	
File system:		
SD memory card	✓	
Rename		✓
User-programmable IP interface:		
Buffer management	✓	
Sending/receiving	✓	✓
System:		
Error indication	✓	
Access to I/Os	✓	

2 Enhancements

Introduction

Jetter AG are continuously striving to add new features and functions to the controller JC-350. By updating your OS you are given the possibility to enhance the functionality of your controller. To do so, you need the following:

- an OS file
- the software tool JetSym
- a connection between PC and controller

Contents

2.1 Various new features and modifications

Introduction

This chapter covers the new features and modifications

Contents

IP address notification

Feature

The controller JC-350 notifies other devices on the network of its current IP address. To this end, it sends once a special Ethernet telegram (“Gratuitous ARP”) on the following occasions:

- during the boot up process as soon as the IP settings are active;
- during runtime when IP settings have changed.

Advantages

The resulting advantages are:

- Other devices on the network and infrastructure devices (such as switches) can immediately contact the “new” device.
- This feature reduces the time spent to establish actual communication with the controller.

New function: FileEnd()

Introduction	This is the first OS version of the controller JC-350 that supports the STX function `FileEnd()`.
Prerequisites	JetSym programming environment version 5.3 or higher must be installed to be able to use these functions.
Declaration	`Function FileEnd(Ref F:File):Int`
Reference	For a detailed description of this command and its application refer to JetSym online help.

New function: DirLister

Introduction

This is the first OS version of the controller JC-350 that supports the STX function DirLister.

Prerequisites

JetSym programming environment version 5.3 or higher must be installed to be able to use these functions.

Declaration

```
Type
DirKind : Enum (Files = 0, Folders = 1);
End_Type

Function DirListPath(enKind: DirKind, Const Ref strPath : String,
Const Ref strFilter : String := '*.*') : Bool;
Function DirListGetEntry(Ref strEntry : String) : Bool;
Function DirListClose();
```

Reference

For more information on these functions refer to JetSym online help.

New functions: BitSetReg() and BitClearReg()

Introduction	This is the first OS version of the controller JC-350 that supports the STX functions `BitSetReg()` and `BitClearReg()`.
Prerequisites	JetSym programming environment version 5.3 or higher must be installed to be able to use these functions.

Declaration	`Function BitSetReg(RegNr:Int, BitNr:Int);`
	`Function BitClearReg(RegNr:Int, BitNr:Int);`

Reference	For more information on these functions refer to JetSym online help.

STX memory utilization

Introduction

This is the first OS version of the controller JC-350 that lets you display in JetSym STX the memory usage by the application program.

Prerequisites

For displaying the memory usage, version 5.3.1 or higher of the programming tool JetSym must be installed.

Registers - Overview

The registers listed below let you read out the physical memory (in bytes) used by the application program. The readings are displayed as a graphic in the CPU window of JetSym Hardware Manager.

Register	Description
R 211010	Total memory: Total
R 211011	Total memory: Used
R 211012	Total memory: Free
R 211013	System memory: Total
R 211014	System memory: Used
R 211015	System memory: Free
R 211016	Application memory: Total
R 211017	Application memory: Used
R 211018	Application memory: Free
R 211019	Used memory: Program
R 211020	Used memory: Data
R 211021	Used memory: Constants
R 211022	Used memory: Stack
R 211023	Used memory: JIT compiler
R 211024	Used memory: System

Reference

For a detailed description on how to display the STX memory usage, please refer to the JetSym online help.

Formatting SD cards using FAT32

Functions supported so far

So far, JC-350 supported the following SD card functions:

- Read/write access to FAT16 SD cards.
- Read/write access to FAT32 SD cards.
- Formatting SD cards using FAT16.

New function

Now, JC-350 supports the following additional SD card functions:

- Formatting SD cards using FAT32.

Formatting

To format an SD Card in the controller, proceed as follows:

Step	Action
1	Enter the value 1179923250 (0x46543332) into register 202936.
2	Disconnect the controller from the power supply.
3	Energize the controller. **Result**: During the boot process the SD card is formatted in FAT32 format.

Task switch on I/O access is now enabled

Obsolete function

When the controller was powered up, Task switch on I/O access was disabled by default.

New function

When the controller is powered up, Task switch on I/O access is enabled by default.

Activating this function

System command register 202961 lets you enable/disable Task switch on I/O access.

- To disable Task switch on I/O access, use system command 160.
- To enable Task switch on I/O access, use system command 161.

Bit 0 of system status register 202962 shows the state of this function.

- If bit 0 is set (= 1), this function is enabled.
- If bit 0 is not set (= 0), this function is disabled.

Additional JetIP/TCP server connections

Obsolete technical data

Parameter	Description
Number of connections	4

New technical data

Parameter	Description
Number of connections	8

Why this change was made:

Enabling multiple connections to be open at the same time.

Enhanced error register 200009

R 200009

Error bits added in this version are highlighted in gray:

Meaning of the individual bits	
Bit 3	Error in file "ModConfig.da"
Bit 5	Fatal internal error of the unit executing the application program
Bit 10	Error message from a device on the Jetter Ethernet system bus
Bit 12	Error message from JetIPScan
Bit 16	Error message from NetConsistency
Bit 20	Internal error of the OS’s memory management unit
Bit 21	Internal error of the application program’s memory management unit
Bit 22	At booting up the system logger is active (register 209700 = 213)
Bit 24	IP address conflict detected

Module register properties	
Type of access	Read access
Value after reset	0

NetConsistency copies configuration files

Function supported so far	NetConsistency checked the IP settings of the configured devices and set them if necessary.
New function	In addition, NetConsistency copies the configuration and parameter files of the configured devices on the network and reboots them.
Restriction	The network topology must be star-shaped.

2.2 User-programmable IP interface

The user-programmable IP interface

The user-programmable IP interface allows to send or receive any data via Ethernet interface on the device using TCP/IP or UDP/IP. When using this feature, data processing is completely carried out by the application program.

Applications

The user-programmable IP interface allows the programmer to carry out data exchange via Ethernet connections which do not use standard protocols, such as FTP, HTTP, JetIP or Modbus/TCP. The following applications are possible:

- Server
- Client
- TCP/IP
- UDP/IP

Required programmer's skills

To be able to program user-programmable IP interfaces the following knowledge of data exchange via IP networks is required:

- IP addressing (e.g. IP address, port number, subnet mask)
- TCP (e.g. connection establishment/termination, data stream, data backup)
- UDP (e.g. datagram)

Restrictions

For communication via user-programmable IP interface, the programmer must not use any ports which are already used by the operating system. Therefore, do not use the following ports:

Protocol	Port number	Default value	User
TCP	Depending on the FTP client	20	FTP server (data)
TCP	21		FTP server (controller)
TCP	23		System logger
TCP	80		HTTP server
TCP	From the file /EMAIL/email.ini	25, 110	E-mail client
TCP	502		Modbus/TCP server
TCP, UDP	1024 - 2047		Various
TCP, UDP	IP configuration	50000, 50001	JetIP
TCP	IP configuration	52000	Debug server

2 Enhancements

Contents

2.2.1 Programming

Introduction

The user-programmable IP interface is used to carry out data exchange between application program and network client via TCP/IP or UDP/IP connections. For this purpose, function calls are used. These function calls are included in the programming language of the device. To program this feature, proceed as follows:

Step	Action
1	Initializing the user-programmable IP interface
2	Open connections
3	Transfer data
4	Terminate the connections

Technical specifications

Technical data of the user-programmable IP interface:

Function	Description
Number of connections	20
Maximum data size	4,000 bytes
Number of receive buffers per connection	4

Restrictions

While the device is processing one of the functions of the user-programmable IP interface, tasks having called the functions should not be stopped through `TaskBreak` or restarted through `TaskRestart`.

Failure to do so could result in the following errors:

- Connections do not open
- Data loss during sending or receiving
- Connections remain open unintentionally
- Connections are closed unintentionally

Table of contents

Initializing the user-programmable IP interface

Introduction

This function must be initialized each time the application program is launched.

Function declaration

```
Function ConnectionInitialize():Int;
```

Return value

The following return value is possible:

Return value	
0	Always

How to use this function

The function is used and its return value assigned to a variable for further utilization in the following way:

```
Result := ConnectionInitialize();
```

Operating principle

The device processes this function in the following steps:

Step	Description
1	The device closes all open connections of the user-programmable IP interface.
2	The device initializes all OS-internal data structures of the user-programmable IP interface.

Related topics

- **Establishing a connection** (see page 29)
- **Terminating a connection** (see page 38)
- **Sending data** (see page 33)
- **Receiving data** (see page 35)

Establishing a connection

Introduction

Before data can be sent or received, a connection has to be established. Here, the following criteria have to be discerned:

- Which transaction log (TCP or UDP) has to be used?
- Is it a client or a server that has to be installed?

Function declaration

```
Function ConnectionCreate(ClientServerType:Int,
                          IPType:Int,
                          IPAddr:Int,
                          IPPort:Int,
                          Timeout:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment

`ClientServerType`	Client = 1 = CONNTYPE_CLIENT
	Server = 2 = CONNTYPE_SERVER	

`IPType`	UDP/IP = 1 = IPTYPE_UDP TCP/IP = 2 = IPTYPE_TCP	
`IPAddr`	Valid IP address	Required only for TCP/IP client
`IPPort`	Valid IP port number	Will be ignored for UDP/IP client
`Timeout`	0 ... 1,073,741,824 [ms]	0 = infinitely

Return value

If the return value was positive, the connection could be established. If the returned value was negative, an error occurred and the connection could not be established.

Return value	
> 0	A positive return value must be stored in a variable. It must be made available as a handle at activating the functions `Send data`, `Receive data`, and `Terminate connection`.
-1	Error during connection set-up
-2	Internal error
-3	Invalid parameter
-8	Timeout

Using this function with a TCP/IP client

If a client is to establish a TCP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_CLIENT,
                           IPTYPE_TCP,
                           IP#192.168.75.123,
                           46000,
                           T#10s);
```

Functioning principle with a TCP/IP client

The task stops at the program line until the connection is established or the specified timeout has elapsed. This function is processed in the following steps:

Step	Description	
1	The device tries to establish a TCP/IP connection via port 46000 to the network client with IP address 192.168.75.123.	
2	**If ...**	**... then ...**
	the network client has accepted the connection,	the function is terminated and a positive value is returned as handle for further access to the connection.
	the connection could not be established and the timeout of 10 seconds has not elapsed yet,	step 1 is carried out.
	an error has occurred or the timeout has elapsed,	the function is terminated and a negative value is returned.

Using this function with a TCP/IP server

If a server is to establish a TCP/IP connection to a client, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_SERVER,
                           IPTYPE_TCP,
                           0,
                           46000,
                           T#100s);
```

Functioning principle with a TCP/IP server

The task stops at the program line until the connection is established or the specified timeout has elapsed. This function is processed in the following steps:

Step	Description	
1	The device sets up TCP/IP port 46000 for receiving connection requests.	
2	**If ...**	**... then ...**
	the network client has established a connection,	no further connection requests to this port are accepted, the function is terminated and a positive value is returned as handle for further access to the connection.
	the connection could not be established and the timeout of 100 seconds has not elapsed yet,	the system waits for a connection to be established.
	an error has occurred or the timeout has elapsed,	the function is terminated and a negative value is returned.

Using this function with a UDP/IP client

If a client is to establish a UDP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_CLIENT,
                           IPTYPE_UDP,
                           0,
                           0,
                           0);
```

Functioning principle with a UDP/IP client

UDP is a connectionless communication mode. For this reason, the device opens only one communication channel for sending data to a network client. This function is processed in the following steps:

Step	Description	
1	The device sets up a UDP/IP communication channel for sending data.	
2	**If ...**	**... then ...**
	no error has occurred,	the function is terminated and a positive value is returned as handle for further access to the connection.
	an error has occurred,	the function is terminated and a negative value is returned.

Using this function with a UDP/IP server

If a server is to establish a UDP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_SERVER,
                           IPTYPE_UDP,
                           0,
                           46000,
                           0);
```

Functioning principle with a UDP/IP server

UDP is a connectionless communication mode. For this reason, the device opens only one communication channel for receiving data from a network client. This function is processed in the following steps:

Step	Description	
1	The device sets up a UDP/IP communication channel at port 46000 for receiving data.	
2	**If ...**	**... then ...**
	no error has occurred,	the function is terminated and a positive value is returned as handle for further access to the connection.
	an error has occurred,	the function is terminated and a negative value is returned.

Related topics

- **Terminating a connection** (see page 38)
- **Sending data** (see page 33)
- **Receiving data** (see page 35)
- **Initializing the user-programmable IP interface** (see page 28)

Sending data

Introduction

Data can be sent via a previously established connection.

Function declaration

```
Function ConnectionSendData(IPConnection:Int,
                            IPAddr:Int,
                            IPPort:Int,
                            Const Ref SendData,
                            DataLen:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
`IPConnection`	Handle	Return value of the function `ConnectionCreate()`
`IPAddr`	Valid IP address	Required only for UDP/IP client
`IPPort`	Valid IP port number	Required only for UDP/IP client
`SendData`	address of the data block to be sent	
`DataLen`	1 ... 4,000	Data block length in bytes

Return value

The following return values are possible:

Return value	
0	Data have been sent successfully
-1	Error when sending, e.g. connection interrupted
-3	Invalid handle, e.g. sending via a UDP/IP server

Using this function with a TCP/IP connection

If data are to be sent via a TCP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionSendData(hConnection,
                             0,
                             0,
                             SendBuffer,
                             SendLen);
```

Functioning principle with a TCP/IP connection

When using TCP/IP, data are sent via a previously opened connection. Therefore, specification of the IP address and IP port number is not required anymore and can be ignored in the function.

In the following situations, the task is not processed further after issuing this function call:

- The data have been sent and their reception has been confirmed.
- An error has occurred.

Using this function with a UDP/IP connection

If, with a client, data are to be sent via a UDP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionSendData(hConnection,
                             IP#192.168.75.123,
                             46000,
                             SendBuffer,
                             SendLen);
```

Functioning principle with a UDP/IP connection

With UDP/IP there is no connection between two given network clients. Therefore, with each function call data can be sent to another client or another port. The task will pause at this function call, until the data are sent.

You will not get any acknowledgment of the remote network client having received the data.

UDP/IP-client and -server

A UDP/IP-client connection is for sending data only. The sending port is set by the operating system.

A UDP/IP-server connection is for both sending and receiving data. The port which was specified at opening up the communication is used as sending port.

Related topics

- **Initializing the user-programmable IP interface** (see page 28)
- **Establishing a connection** (see page 29)
- **Terminating a connection** (see page 38)
- **Receiving data** (see page 35)

Receiving data

Introduction

Data can be sent via a previously established TCP/IP connection or via a UDP/IP connection of a server.

Via UDP/IP connection of a client data can not be received, but only sent.

Restrictions

Data packets which are received via network must be fetched by the application program. Per connection, four packets as a maximum are stored temporarily in the operating system of the controller. All further packets are discarded.

Function declaration

```
Function ConnectionReceiveData(IPConnection:Int,
                               Ref IPAddr:Int,
                               Ref IPPort:Int,
                               Ref ReceiveData,
                               DataLen:Int,
                               Timeout:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
`IPConnection`	Handle	Return value of the function `ConnectionCreate()`
`IPAddr`	Address of a variable for saving the IP address of the sender	Required only for UDP/IP server
`IPPort`	Address of a variable for saving the IP port number of the sender	Required only for UDP/IP server
`ReceiveData`	Address of the data block to be received	
`DataLen`	1 ... 4,000	Maximum data block length in bytes
`Timeout`	0 ... 1,073,741,824 [ms]	0 = infinite

Return value

The following return values are possible:

Return value	
> 0	Number of received data bytes
-1	Error when receiving data, e.g. connection interrupted
-3	Invalid handle, e.g. receiving data via a UDP/IP client
-8	Timeout

Using this function with a TCP/IP connection

If data are to be received via a TCP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionReceiveData(hConnection,
                                Dummy,
                                Dummy,
                                ReceiveBuffer,
                                sizeof(ReceiveBuffer),
                                T#10s);
```

Functioning principle with a TCP/IP connection

When using TCP/IP, data are sent via a previously opened connection. Therefore, specification of the IP address and IP port number is not required anymore and can be ignored in the function.

In the following situations, the task is not processed further after issuing this function call:

- The data have been received.
- An error has occurred.

In case of a TCP/IP connection, data are sent as data stream.

The device processes this function in the following steps:

Step	Description	
1	The device waits until data have been received, but no longer than the specified timeout.	
2	**If ...**	**... then ...**
	the timeout has elapsed or the connection has been terminated,	the function is exited and an error message is issued.
	data have been received,	they are copied to the receive buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3.
3	**If ...**	**... then ...**
	more data have been received than could have been copied into the receive buffer,	these are buffered by the device to be fetched by further function calls.
4	The function is exited and the number of data, which have been copied into the receive buffer, is returned.	

Using this function with a UDP/IP server

If, with a server, data are to be received via a UDP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionReceiveData(hConnection,
                                IPAddr,
                                IPPort,
                                ReceiveBuffer,
                                sizeof(ReceiveBuffer),
                                T#10s);
```

Functioning principle with a UDP/IP server

In the following situations, the task is not processed further after issuing this function call:

- All data have been received.
- An error has occurred.

In case of a UDP/IP connection, data are sent as datagram.
The device processes this function in the following steps:

Step	Description	
1	The device waits until all data of a datagram have been received, but no longer than the specified timeout.	
2	**If ...**	**... then ...**
	the timeout has elapsed or the connection has been terminated,	the function is exited and an error message is issued.
	data have been received,	they are copied to the receive buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3.
3	**If ...**	**... then ...**
	... more data have been received than could be copied into the receive buffer - that is, if the sent datagram is too large,	... these data are discarded.
4	The sender's IP address and IP port number are transferred into the variables which are given along with the data.	
5	The function is exited and the number of data, which have been copied into the receive buffer, is returned.	

Related topics

- **Initializing the user-programmable IP interface** (see page 28)
- **Establishing a connection** (see page 29)
- **Terminating a connection** (see page 38)
- **Sending data** (see page 33)

Terminating a connection

Introduction

Clear all connections which are no longer required as the number of concurrently opened connections is limited.

Function declaration

```
Function ConnectionDelete(IPConnection:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
`IPConnection`	Handle	Return value of the function `ConnectionCreate()`

Return value

The following return values are possible:

Return value	
0	Connection terminated and deleted
-1	Invalid handle

How to use this function

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

```
Result := ConnectionDelete(hConnection);
```

Related topics

- **Establishing a connection** (see page 29)
- **Sending data** (see page 33)
- **Receiving data** (see page 35)
- **Initializing the user-programmable IP interface** (see page 28)

2.2.2 Registers

Introduction

This chapter describes the registers of the device which contain the current connection list of the user-programmable IP interface. These registers can be used for debugging or diagnostic purposes. However, they can't be used for other functions such as establishing or terminating a connection.

Contents

Register numbers

Introduction

Data of one connection each are displayed within the registers of a coherent register block. The basic register number of this block is dependent on the controller.

Register numbers

Basic register number	Register numbers
350000	350000 ... 350007

Determining the register number

In this chapter only the last figure of a register number is specified, for example MR 1. To calculate the complete register number, add the basic register number of the corresponding device to this figure, e.g. 350000.

Registers - Overview

Register	Description
MR 0	Selecting a connection
MR 1	Type of connection
MR 2	Transport protocol
MR 3	IP address
MR 4	IP port number
MR 5	State
MR 6	Number of sent bytes
MR 7	Number of received bytes
MR 8	Number of discarded bytes
MR 9	Number of discarded packets

Registers - Description

Introduction

The operating system manages the established connections in a list. Module register MR 0 *Selection of a connection* is used to copy connection details into other registers of a register block.

MR 0

Selecting a connection

Connections are selected by writing values to this register. This register is used to display whether the following registers contain usage data.

Module register properties		
Reading values	0	Connection exists
	-1	Connection does not exist

Module register properties		
Writing values	0	Address the first connection in the list
	> 0	Address the next connection in the list
	< 0	Address the previous connection in the list

MR 1

Type of connection

The value in this register shows whether the connection is a client or a server connection.

Module register properties		
Values	1	Client
	2	Server

MR 2

Transport protocol

The value in this register shows whether TCP or UDP is used as transport protocol.

Module register properties		
Values	1	UDP
	2	TCP

MR 3

IP address

The value in this register shows the configured IP address.

Module register properties	
Values	0.0.0.0 ... 255.255.255.255

MR 4

IP port number

The value in this register shows the configured IP port number.

Module register properties	
Values	0 ... 65.535

MR 5

Indication

The value in this register shows status the connection is currently in.

Module register properties		
Values	0	Connection terminated
	1	Connection is being established
	2	Connection is established
	3	TCP/IP server: Waiting for connection request from client
	4	Internal usage

MR 6

Number of sent bytes

The value in this register shows the number of data bytes sent via the given connection. Since this is a signed 32-bit register and the sent bytes are added each time, the number range may be exceeded from the positive maximum value to the negative maximum value.

Module register properties	
Values	-2.147.483.648 ... 2.147.483.647

MR 7

Number of received bytes

The value in this register shows the number of data bytes received via the given connection. Since this is a signed 32-bit register and the received bytes are added each time, the number range may be exceeded from the positive maximum value to the negative maximum value.

Module register properties	
Values	-2.147.483.648 ... 2.147.483.647

MR 8

Number of discarded bytes

The value in this register indicates the data bytes which could not be received, because the application program had not taken the cached data bytes.

Module register properties	
Values	0 ... 2.147.483.647

MR 9

Number of discarded packets

The value in this register indicates the data packets which could not be received, because the application program had not taken the cached data packages.

Module register properties	
Values	0 ... 2.147.483.647

3 Fixed software bugs

Introduction

This chapter describes the software bugs which have been fixed in the new OS version.

Contents

User-Programmable IP Interface - Illegal connection handle

Error description

If in the case of `ConnectionReceiveData()`, `ConnectionSendData()` or `ConnectionDelete()` an illegal connection handle is specified, the controller crashes.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JCM-350	< 1.28.0.00 < 1.28.0.00 < 1.10.0.00 < 1.28.0.00 < 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

Check the connection handle for validity before using it.

Fixed versions/revisions

Starting from the following versions/revisions this bug has been fixed:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JCM-350	1.28.0.00 1.28.0.00 1.10.0.00 1.28.0.00 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Long key names caused a crash

Error description	If for several locks/keys long names were entered in the file **/System/keys.ini**, the controller crashes.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JM-200-ETH JCM-350	< 1.28.0.00 < 1.28.0.00 < 1.10.0.00 < 1.28.0.00 < 1.28.0.00 < 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

Make sure that the length (number of characters) of all names in the file **/System/keys.ini** does not exceed 224 characters.

Fixed versions/revisions

Starting from the following versions/revisions this bug has been fixed:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JM-200-ETH JCM-350	1.28.0.00 1.28.0.00 1.10.0.00 1.28.0.00 1.28.0.00 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

In renaming files names with the maximum number of characters caused the controller to crash

Error description

If a file was renamed and the new name had the maximum length of 63 characters, the controller crashed.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JM-200-ETH JCM-350	< 1.28.0.00 < 1.28.0.00 < 1.10.0.00 < 1.28.0.00 < 1.28.0.00 < 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

Make sure that the length of the new name does not exceed 62 characters.

Fixed versions/revisions

Starting from the following versions/revisions this bug has been fixed:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JM-200-ETH JCM-350	1.28.0.00 1.28.0.00 1.10.0.00 1.28.0.00 1.28.0.00 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Input values of a device on the network was frozen

Error description

If several devices on the network communicated with the controller via Publish/subscribe, it could happen that input values of one of the devices froze. The outputs on this device still worked. Inputs and outputs to other devices also continued to work. No errors were signaled. This problem could be solved by relaunching this subscriber on the controller.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JCM-350	< 1.28.0.00 < 1.28.0.00 < 1.10.0.00 < 1.28.0.00 < 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

Connect on each device a not assigned output to a not assigned input. Toggle this output in the application program. Check whether the input follows the output state. When the input stops following the output, relaunch the subscriber on the controller.

Fixed versions/revisions

Starting from the following versions/revisions this bug has been fixed:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC JC-310-JM JCM-350	1.28.0.00 1.28.0.00 1.10.0.00 1.28.0.00 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

JetSym oscilloscope displayed incorrect values of Float registers

Error description

During oscilloscope recording sessions in JetSym incorrect values were displayed for Float registers.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC	< 1.28.0.00 < 1.28.0.00 < 1.10.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

Use only Integer registers in JetSym oscilloscope.

Fixed versions/revisions

Starting from the following versions/revisions this bug has been fixed:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC	1.28.0.00 1.28.0.00 1.10.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

IP configuration: Inconsistency in registers

Error description

When changes were made to IP settings in registers 101200 through 101202, the new values were immediately displayed in registers 104531 through 104533. However, they did not take effect immediately.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-310-JM JCM-350	< 1.28.0.00 < 1.28.0.00 < 1.28.0.00 < 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

There is no remedy or workaround for affected versions/revisions.

Fixed versions/revisions

Starting from the following versions/revisions this bug has been fixed:

OS version	JC-340/350 JC-360/365 (MC) JC-310-JM JCM-350	1.28.0.00 1.28.0.00 1.28.0.00 1.20.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Jetter
automation

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg | Germany

Phone +49 7141 2550-0
Fax +49 7141 2550-425
info@jetter.de
www.jetter.de

We automate your success.