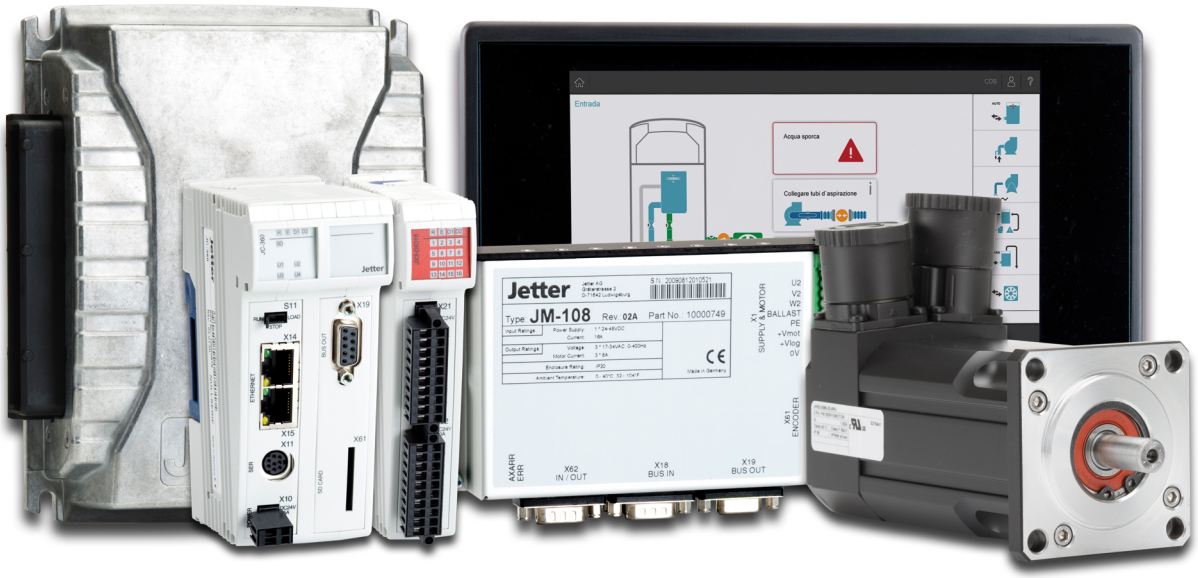
Jetter automation



JC-940MC

Version Update from V. 1.05 to V. 1.10

We automate your success.

Revision 1.01
September 2017 / Printed in Germany

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

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

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

Table of Contents

1 Introduction 4

Operating system update 5
JC-940MC Version Update - Overview 6
Interdependencies of OS versions in moving CAN axes using MC 7

2 Enhancements 8

2.1 Various new features and modifications 9
Debugging registers for STX variables 10
New functions NetBitSetReg() and NetBitClearReg() 12
New function: FileCopy() 13
Debugging of properties in the setup pane 14
Nested tasklocks 15
System start without DNS 16
New function: DirLister 17
New function: FileEnd() 18
New functions: BitSetReg() and BitClearReg() 19
STX memory utilization 20
Additional JetIP/TCP server connections 21
Enhanced error register 200009 22
NetConsistency copies configuration files 23
Flash disk - Properties 24
Additional supported USB port 25
Additional PCI slots 26
More slave modules on JX2 system bus supported 27
2.2 DNS client 28
Register overview 29
Register description 30
2.3 User-programmable IP interface 33
2.3.1 Programming 35
Initializing the user-programmable IP interface 36
Establishing a connection 37
Sending data 41
Receiving data 43
Terminating a connection 46
2.3.2 Registers 47
Register numbers 48
Registers - Description 49

3 Fixed software bugs 52

Exceptions get lost 53
No download and debugging at partial download 54
User-Programmable IP Interface - Illegal connection handle 55
Long key names caused a crash 56
In renaming files names with the maximum number of characters caused the controller to crash 57
Input values of a device on the network was frozen 58
JetSym oscilloscope displayed incorrect values of Float registers 59
Timer overflow at TimerStart/TimerEnd 60

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

Topic	Page
Operating system update	5
JC-940MC Version Update - Overview	6
Interdependencies of OS versions in moving CAN axes using MC	7

Operating system update

OS file for operating system update

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

OS file	Description
JC-940MC_1.10.0.00.os	OS file for JC-940MC with version 1.10

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-940MC with version 1.10, JetSym 5.3.0 or higher is required.

JC-940MC Version Update - Overview

V. 1.01

The following table gives an overview of newly added features and fixed software bugs in OS version 1.01:

Feature	New	Fixed
Local I/O modules:		
Added	✓	

V. 1.10

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

Feature	New	Fixed
Application program:		
Memory management	✓	
New functions	✓	
Communication:		
JetIP server	✓	
NetConsistency	✓	
File system:		
Memory	✓	
USB flash drive	✓	
Rename		✓
User-programmable IP interface:		
Buffer management	✓	
Sending/receiving data	✓	✓
System:		
Error indication	✓	
Additional PCI slots	✓	

Interdependencies of OS versions in moving CAN axes using MC

Interdependencies of OS versions

JC-940MC, JX6-SB-I, and JM-2xx must have the following OS versions if it is intended to move CAN axes by means of Motion Control (MC).

Device	Minimum OS version
JC-940MC	1.10
JX6-SB-I	2.30.0.09
JM-2xx	2.15.0.08

2 Enhancements

Introduction

Jetter AG are continuously striving to add new features and functions to the controller JC-940MC. 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

Topic	Page
Various new features and modifications	9
DNS client	28
User-programmable IP interface	33

2.1 Various new features and modifications

Introduction

This chapter covers the new features and modifications

Contents

Topic	Page
Debugging registers for STX variables	10
New functions NetBitSetReg() and NetBitClearReg()	12
New function: FileCopy()	13
Debugging of properties in the setup pane	14
Nested tasklocks	15
System start without DNS	16
New function: DirLister	17
New function: FileEnd()	18
New functions: BitSetReg() and BitClearReg()	19
STX memory utilization	20
Additional JetIP/TCP server connections	21
Enhanced error register 200009	22
NetConsistency copies configuration files	23
Flash disk - Properties	24
Additional supported USB port	25
Additional PCI slots	26
More slave modules on JX2 system bus supported	27

Debugging registers for STX variables

Obsolete function

For remote access to controller variables, JetSym had to be applied so far. The project going with the running program had to be active. Only then access in setup or monitor mode would be successful.

New function

By means of two registers and an appropriate application program, certain controller variables can be accessed.

Reason for this change

The main purpose of applying these registers is supporting plant diagnostics, if there is neither any PC with JetSym available, nor is there any remote maintenance possible.

R 210091

Variable address

This register is for setting the address of a variable. The address of a variable can be determined by the map file of the project or within the JetSym editor.

Register properties	
Values	0 ... end of the variable range
Value after reset	0

R 210093

Memory address

This register is for displaying the absolute memory address of the variable.

Register properties	
Values	0 ... 4,294,967,296
Value after reset	Base address

Example

```
Var
    nDebugVar1:Int;
    pnData: Pointer To Int;
    nValue: Int;
End_Var;

Const
    cVar1Addr = &nDebugVar1;
End_Const;

Task tMain Autorun
    ...
    pnData := Regs[210093];
    nValue := @pnData;
    ...
End_Task
```

Determining the memory address in the JetSym editor

Step	Action
1	In the source code, assign constants to the variable addresses.
2	Compile the program
3	Place the mouse cursor on the constant.

Result: JetSym lets you read out the value of the constant. Enter the value into register 210091.

Retrieving the memory address from the map file

The map file is assigned the name, the address, and the length of the variable:

```
Memory:
nDebugVar1;12;4
```

Enter the variable address taken from the map file (here: 12) into register 210091.

Accessing the variable value

Step	Action
1	Enter the variable address into R 210091.
2	Read the memory address out of R 210093 and assign it to a pointer variable.
3	Read out the variable value via pointer variable.

New functions NetBitSetReg() and NetBitClearReg()

Introduction

This is the first OS version of the controller JC-940MC that supports the STX functions `NetBitSetReg()` and `NetBitClearReg()`.

Prerequisites

JetSym programming environment version 5.2 or higher must be installed to be able to use these functions.

Declaration

```
Function NetBitSetReg(IPAddr,DestRegNumber,BitMask,Ref
NewVal,Interf,IPPort:=DEFAULTIPPORT,Res:=0);
Function NetBitClearReg(IPAddr,DestRegNumber,BitMask,Ref
NewVal,Interf,IPPort:=DEFAULTIPPORT,Res:=0);
```

Reference

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

New function: FileCopy()

Introduction	This is the first OS version of the controller JC-940MC that supports the STX function `FileCopy()`.
Prerequisites	JetSym programming environment version 5.2 or higher must be installed to be able to use these functions.
Declaration	`Function FileCopy(Const Ref strSrcName:String, Const Ref strDstName:String):Int`
Reference	For a detailed description of this command and its application refer to JetSym online help.

Debugging of properties in the setup pane

Introduction	As of this OS version, debugging of properties in the setup pane is supported by JetSym.
Prerequisites	To display properties in the setup pane, JetSym programming environment version 5.2 or higher must be installed.
Reference	For a detailed description of object properties and their application, refer to the JetSym online help.

Nested tasklocks

Obsolete function	If an application program had issued the `Tasklock` command to block task switch, issuing the `Taskunlock` command once would enable task switch again.
New function	If an application program has issued the `Tasklock` command more than once to block task switch, task switch is not enabled before issuing the `Taskunlock` command as often as having issued the `Tasklock` command.

System start without DNS

Introduction

When specifying IP addresses of target systems, you can also use names. Then the controller translates these names into IP addresses. Configuring the e-mail client may serve as an example for this. Making an entry in a configuration file or the Domain Name System is used to assign names to their corresponding IP address.

Obsolete function

During the boot process the controller reads the IP address of the DNS server out of the configuration memory. This means that the IP address of the DNS server must be known and must not be changed at that moment.

New function

The controller resolves the name each time an e-mail is sent. The IP address of the DNS server can be changed during runtime via register.

Reason for this change

The IP address of the DNS server is to be changeable during runtime.

R 104534

IP address of the DNS server during runtime

This register contains the IP address of the DNS server being valid during runtime of the controller. The value written to this register cannot be stored to a remanent memory.

Register properties	
Values	Valid IP address

New function: DirLister

Introduction

This is the first OS version of the controller JC-940MC 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 function: FileEnd()

Introduction	This is the first OS version of the controller JC-940MC 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 functions: BitSetReg() and BitClearReg()

Introduction	This is the first OS version of the controller JC-940MC 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-940MC 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.

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)

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.

Flash disk - Properties

Capacity

The following disk space is available to the user:

Parameter	Value
Flash disk capacity up to OS V. 1.06	8 MB
Flash disk capacity as of OS V. 1.07	32 MB

Important note!

When you update to version 1.07, the user area of the flash disk will be formatted.

Therefore, backup all files to a separate storage medium beforehand.

Properties

The internal flash disk drive has got the following properties:

- Up to 7 directory levels and 1 file level are allowed.
- Differentiation between upper and lower case.
- Directory and file names with a length of up to 63 characters are possible.
- All characters except "/" and ".." are permitted for directory and file names
- User/access administration for a maximum number of 31 locks and 33 users.

Additional supported USB port

Obsolete technical data

Parameter	Description
Number of supported USB flash drives	1
File system folder	/USB1

New technical data

Parameter	Description
Number of supported USB flash drives	2
File system directories	/USB1 /USB2

Why this change was made

Up to two USB flash drives can be plugged in at the same time.

Allocation of USB flash drives

USB flash drives plugged into the USB ports are allocated to the directories in the file system as described below:

One USB flash drive:

If only one USB flash drive is plugged in, it is automatically allocated to /USB1, irrespective of the port it is actually plugged in.

Two USB flash drives are plugged in while the controller is running:

Order	Description
USB flash drive plugged in first	/USB1
USB flash drive plugged in second	/USB2

Two USB flash drives are already plugged in when the controller is energized

The allocation depends on the order in which the operating system of the controller activates the USB ports.

Order	Description
1	X63
2	X64
3	X61
4	X62

One of two USB flash drives is removed:

The remaining USB flash drive remains allocated to its directory.

Additional PCI slots

Obsolete technical data

Parameter	Description
Number of PCI slots	1

New technical data

Parameter	Description
Number of PCI slots	3

Why this change was made

To support a greater number of local submodules.

More slave modules on JX2 system bus supported

Obsolete technical data

Parameter	Description
Number of supported slave modules on the JX2 system bus of a JX6-SB-I	8

New technical data

Parameter	Description
Number of supported slave modules on the JX2 system bus of a JX6-SB-I	16

Why this change was made

To support a greater number of Slave modules on the JX2 system bus of a JX6-SB-I module.

2.2 DNS client

Introduction

This chapter describes the registers of the DNS client and the DNS cache.

Contents

Topic	Page
Register overview	29
Register description	30

Register overview

Introduction

You can access the data of the DNS client via the following register that have been written into.

Register overview

Register	Description
101x03	IP address of the DNS server taken from the IP configuration
104534	IP address of the DNS server during runtime
510000	Status
510001	Command
510002	IP address of DNS server
510003	Error code
510009	Number of entries in the DNS cache
510010	Timeout
510011	IP address
510012 ... 510029	Name

Register description

R 101x03

IP address of the DNS server taken from the IP configuration

For a description of these registers, refer to the chapter on IP configuration.

R 104534

IP address of the DNS server during runtime

This register contains the IP address of the DNS server being valid during runtime of the controller. The value written to this register cannot be stored to a remanent memory.

Register properties	
Values	Valid IP address

R 510000

Status

In this register, the controller signals a summary of status messages in bit-coded mode.

Meaning of the individual bits		
Bit 0	**File /etc/hosts**	
	0 =	File not read / The file does not exist
	1 =	The file has been read and the entries have been stored in the cache
Bit 1	**Access to the DNS server**	
	0 =	No access
	1 =	Access has taken place
Bit 2	**Running state**	
	0 =	Access to the DNS server is not running
	1 =	Access to the DNS server is running
Bit 3	**Error**	
	0 =	No error while accessing a DNS server
	1 =	Error while accessing a DNS server

Register properties	
Access	Read

R 510001

Command

Controls the access to the DNS cache.

Register properties		
Values	1	Pick the first entry in the cache
	2	Pick the next entry in the cache
	3	Clear the entry from the cache

R 510002

IP address of DNS server

Displays the current IP address of the DNS server.

Register properties	
Values	Valid IP address
Type of access	Read access

R 510003

Error code

If bit 3 of register 510000 is set, this register specifies the error code.

Register properties		
Values	-1	Error at accessing the DNS server, e.g. no reply
	-2	Invalid reply of the DNS server
	-3	No IP address could be retrieved from the response
Type of access	Read access	

R 510009

Number of entries in the DNS cache

It specifies the number of entries in the cache of the DNS clients.

Register properties		
Values	0 ...	The amount of entries is limited by the available memory.
Type of access	Read access	

R 510010

Timeout

Displays the time-out of the cache entry picked by command 1 or 2.

Register properties		
Values	0	The entry is never cleared automatically.
Type of access	Read access	

R 510011

IP address

Display the IP address of the cache entry picked by command 1 or 2.

Register properties	
Values	Valid IP address
Type of access	Read access

R 510012 ... R 510029

Name

Displays the name of the cache entry picked by command 1 or 2 in the register string format.

Register properties	
Values	String of 31 characters max.
Type of access	Read access

2.3 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

Contents

Topic	Page
Programming	35
Registers	47

2.3.1 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

Topic **Page**
Initializing the user-programmable IP interface 36
Establishing a connection 37
Sending data 41
Receiving data 43
Terminating a connection 46

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 37)
- **Terminating a connection** (see page 46)
- **Sending data** (see page 41)
- **Receiving data** (see page 43)

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 46)
- **Sending data** (see page 41)
- **Receiving data** (see page 43)
- **Initializing the user-programmable IP interface** (see page 36)

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 36)
- **Establishing a connection** (see page 37)
- **Terminating a connection** (see page 46)
- **Receiving data** (see page 43)

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 36)
- **Establishing a connection** (see page 37)
- **Terminating a connection** (see page 46)
- **Sending data** (see page 41)

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 37)
- **Sending data** (see page 41)
- **Receiving data** (see page 43)
- **Initializing the user-programmable IP interface** (see page 36)

2.3.2 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

Topic	Page
Register numbers	48
Registers - Description	49

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

Topic	Page
Exceptions get lost	53
No download and debugging at partial download	54
User-Programmable IP Interface - Illegal connection handle	55
Long key names caused a crash	56
In renaming files names with the maximum number of characters caused the controller to crash	57
Input values of a device on the network was frozen	58
JetSym oscilloscope displayed incorrect values of Float registers	59
Timer overflow at TimerStart/TimerEnd	60

Exceptions get lost

Symptoms

Exceptions, which are thrown out of the application program by the command `ThrowException()` are overwritten and thus get lost.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC	< 1.22.0.00 < 1.22.0.00 < 1.06.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

There is no remedy to be applied to the releases concerned.

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	1.22.0.00 1.22.0.00 1.06.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

No download and debugging at partial download

Symptoms

If you select in JetSym under project properties the option "Partial Download", download with following debugging will not work.

The controller does not stop the program at the predefined breakpoint but will pass it.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	JC-340/350 JC-360/365 (MC) JC-940MC	< 1.22.0.00 < 1.22.0.00 < 1.06.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

Remedy/workaround

There is no remedy to be applied to the releases concerned.

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	1.22.0.00 1.22.0.00 1.06.0.00
Hardware revision	Not relevant	
Configuration or operating mode	Not relevant	

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	

Timer overflow at TimerStart/TimerEnd

Error description

When a timer, that was started using the function `TimerStart()`, elapsed, the function `TimerEnd()` returned `True`. If the controller was not switched off for several days, function `TimerEnd()` returned the value `False`, even though the timer has not been restarted.

Affected versions/revisions

The following versions/revisions are affected by this bug:

OS version	< 1.10.0.0
Hardware revision	Not relevant
Configuration or operating mode	Not relevant

Remedy/workaround

There is no remedy for the affected versions/revisions.

Fixed versions/revisions

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

OS version	1.10.0.0
Hardware revision	Not relevant
Configuration or operating mode	Not relevant

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.