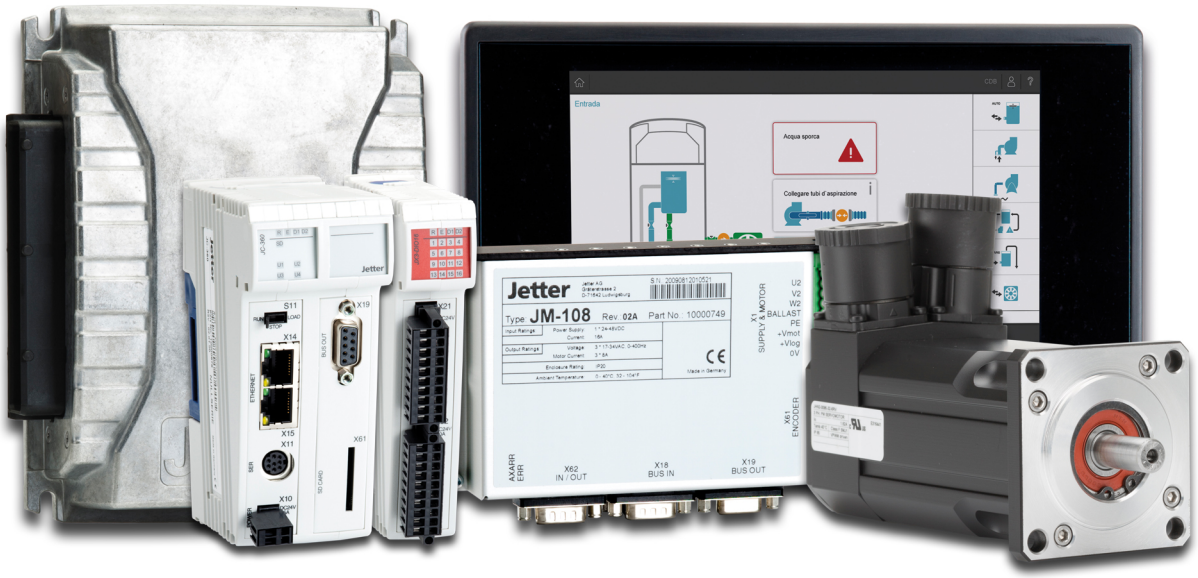
Jetter automation



JC-310-JM

Version Update from V. 1.22 to V. 1.28

We automate your success.

Revision 1.00
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-310-JM_1.28.0.00.os	OS file for JC-310-JM 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-310-JM with version 1.28, JetSym 5.3.0 or higher is required.

JC-310-JM version update - Overview

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:		
Task processing (system command 170/171)	✓	
Taskcontinue	✓	
Memory management	✓	
New features	✓	
Ethernet system bus:		
NetConsistency	✓	
Communication:		
STX debug server	✓	✓
JetIP server	✓	
NetConsistency	✓	
File system:		
Rename		✓
User-programmable IP interface:		
Buffer management	✓	
Sending/receiving data	✓	✓
System:		
Start delay	✓	
IP configuration	✓	
Error indication	✓	

2 Enhancements

Introduction

Jetter AG are continuously striving to add new features and functions to the controller JC-310-JM. 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

Taskcontinue launches an application task

Introduction	The instructions `Taskcontinue` and `TaskContinueById()` continue an application task, which has been stopped by `Taskbreak` or `TaskBreakById()`.
Obsolete function	A task which has not been running yet since program launch (no `Autorun` attribute), cannot be started by `Taskcontinue` or `TaskContinueById()`.
New function	A task which has not been running yet since program launch, can be started by `Taskcontinue` or `TaskContinueById()`.
Why this change was made	As regards the range of functions of this feature, it is to be adjusted to the other controller series.

New function: FileEnd()

Introduction	This is the first OS version of the controller JC-310-JM 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-310-JM 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-310-JM 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-310-JM 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)
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 Startup delay

Introduction

The device provides a register to which a delay time can be written.

Application

The boot process of the device is delayed by the entered delay time.

Contents

Setting the startup delay

Introduction

If other devices connected to the bus have got a longer startup time, the boot process must be delayed.

Set delay time

To set the delay time, proceed as follows:

Step	Action
1	Switch on the device.
2	Enter the password. For this, write value 1112502132 (0x424f6f74) to R 202970.
3	Enter the desired delay time in steps of 100 ms into register 202971.

Result: The next boot process will be delayed by the set startup delay time before initializing the JX2 and JX3 system bus.

R 202970

Password register

Enter 1112502132 (0x424F6F74) into this register. Then enter the desired value into the startup delay time register. Now, the controller sets the value of this register to 0.

Register properties	
Value	1112502132 (0x424F6F74)

R 202971

Startup delay time

Write into this register the delay time in multiples of 100 milliseconds.

Register properties	
Values	0 (OFF) ... 3,000 (300 seconds)
Value after reset	As described above (remanent)

Procedure

- The controller only executes start delay, when switch S11 is in *RUN* position.
- Start delay is terminated by leaving the *RUN* position.

Display

- LED **D1** flashing slowly during the first half of the start delay time (approx. 1 Hz).
- LED **D1** flashing fast during the second half of the start delay time (approx. 4 Hz).

2.3 Jetter Ethernet system bus

Introduction

The Jetter Ethernet system bus is based on TCP, UDP/IP and can therefore be used along with other TCP, UDP/IP protocols.

It has been designed for data exchange between the following devices via standard Ethernet.

- Programming device
- Controllers
- Bus node
- Communication modules

Data interchange

The Jetter Ethernet system bus makes a difference between the cyclic and acyclic data interchange between communication participants. Both kinds of data interchange can be executed simultaneously within a network.

Data exchange	Properties
Cyclic	■ **Architecture:** Publish/subscribe ■ **Nodes:** Controllers, bus nodes and communication modules ■ **Access:** Automatically by OS ■ **Access time:** Fast, deterministic ■ **Data:** Registers, inputs/outputs ■ **Configuration:** Hardware Manager in JetSym ■ **Reach:** Subnet
Acyclic	■ **Architecture:** Client/server ■ **Client:** PC and controllers ■ **Server:** PC, controllers, bus nodes and communication modules ■ **Data:** E.g. registers, inputs/outputs, STX variables, application program ■ **Access:** PC or application program ■ **Access time:** Depending on the reaction time of the server ■ **Configuration:** Only when using network registers ■ **Reach:** International

Minimum requirements

The device is operated in a system consisting of various components by Jetter AG. In order to ensure proper interaction of these components, the operating system used and the programming tool JetSym must have at least the release numbers listed below.

Component	As of version
JC-310-JM	1.22.0.00
JC-340	1.22.0.00
JC-350	1.22.0.00
JC-360	1.22.0.00
JC-360MC	1.22.0.00
JC-365	1.26.0.00
JC-365MC	1.26.0.00
JC-440(MC)	1.02.0.00
JC-940MC	1.06.0.20
JC-945MC	1.01.0.00
JX3-BN-ETH	1.18.0.02
JX3-COM-EIPA	1.01.0.00
JX3-COM-PND	1.03.0.06
JM-200-ETH	1.22.0.00
JetSym	5.1.2

Contents

2.3.1 NetConsistency function

Target

The goal of NetConsistency is automated comparison of actual system properties with the set system properties of network nodes. If the actual system properties are not in accordance with the set system properties, the respective issues are automatically replaced within the system by the set system properties.

Application

The user can take the following actions by applying NetConsistency:

- Exchanging a defective system component, a network node by simply adjusting it to the new system component within an engineered plant.
  The JetControl, which is the NetConsistency master, automatically configures the new system component by all kinds of information given in the former system component.
- Easily updating an already existing plant:
  Download of the new system properties to the NetConsistency master JetControl, is required. JetControl automatically recognizes the difference between the former and the actual system configuration. It assigns the new system properties to the respective places.

System properties

Possible system properties are:

- Network parameters (IP address, port number, subnet mask, default gateway)
- Parameter data
- Configuration data

Configuration data

The JetSym Hardware Manager generates the configuration and parameter data. The Hardware Manager transfers the data to JetControl through the feature **Compare program/Download.**

The NetConsistency master

The NetConsistency feature supplies a NetConsistency master defined in the system. Only a JetControl can be a NetConsistency master.

Prerequisites

There are the following prerequisites for using NetConsistency:

- JetSym as of V 5.1.0
- At least one NetConsistency master:

Product	As of version
JC-940MC	1.05.0.08 1.06.6.01 (testing os)
JC-945MC	1.01.0.00
JC-440(MC)	1.02.0.00
JC-340, JC-350	1.23.0.04
JC-310-JM	1.28.0.00
JC-360(MC)	1.28.0.00
JC-365(MC)	1.28.0.00

- NetConsistency slaves: Min. 1, max. 64

Produkt	As of version
JC-310-JM	1.22.0.00
JM-200-ETH	1.22.0.00
Ethernet axis JM-xxx (JM-2xx-OEM)	2.07.0.37
Ethernet axis MC-JM-xxx (JM-2xx-OEM)	2.07.0.37
JX3-BN-ETH	1.18.0.02
JX3-COM-EIPA	1.01.0.00
JX3-COM-PND	1.03.0.06

Contents

NetConsistency function

Restrictions

- NetConsistency is only available for the Jetter Ethernet system bus.
- The network nodes have to be connected to the same subnet.
- Only if JetIPScan is active, NetConsistency will be executed. JetIPScan is active, if bit 2 of R 202962 is set.
- JetControl executes NetConsistency only once at booting the JetControl, which is the master of NetConsistency.

Function

The NetConsistency feature in its actual version comprises the following system properties:

- Network parameters
  - IP address
  - Subnet mask
  - Default gateway
- Parameter data
- Configuration data

Network parameters

For this, NetConsistency uses JetIPScan. One of the JetIPScan features is to assign network parameters to bus nodes via GNN.

The controller assigns the network parameters to those bus nodes which you have configured in Hardware Manager.

The controller assigns the IP address to those bus nodes which you have configured in Hardware Manager.

As subnet mask, the controller assigns its own subnet mask to the bus node.

As default gateway, the controller assigns its own IP address or its own default gateway to the bus node:

Product	Assigned default gateway
JC-940MC and JC-945MC, if only ETH1 has been configured	Default gateway of the controller
JC-940MC and JC-945MC, if ETH2 and/or ETH3 have been configured	IP address of ETH1 of the controller
JC-340, JC-350 and JC-440	Default gateway of the controller

Parameter and configuration data

For this, NetConsistency uses FTP. Via FTP, parameter and configuration files are transferred to the bus nodes.

The controller stores the parameter and configuration files of all bus nodes in a backup directory. For each bus node, the backup directory holds a folder called *NetNode* plus the GNN attached to the name.

Example: File system of the controller: */SysConfig/Backup/NetNode002*

JetSym transfers all parameter and configuration files to the backup directory of the controller by comparison and download.

For uploading the parameter and configuration files, the addressed bus nodes are rebooted after file transfer. Bus nodes **without** parameter and/or configuration files are not rebooted.

The following products having got parameter and configuration files are rebooted:

Product	As of version
JX3-BN-ETH	V. 1.18.0.02
JX3-COM-EIPA	V. 1.01.0.00
JX3-COM-PND	V. 1.03.0.06

The following products **not** having got parameter and configuration files are **not** rebooted:

Product	As of version
JC-310-JM	V. 1.22.0.00
JM-200-ETH	V. 1.22.0.00
Ethernet axis JM-xxx (JM-2xx-OEM)	V 2.07.0.37
Ethernet axis MC-JM-xxx (JM-2xx-OEM)	V 2.07.0.37

System launch of the bus nodes without non-volatile storage of the IP address

At system launch, the bus nodes use the GNN set via their own DIP switch sliders 1 to 8. This applies, until the network parameters configured in Hardware Manager via JetControl - which is the NetConsisteny master - are assigned to the bus node.

Non-volatile storage via NetConsistency of the network parameters assigned last is not implemented.

We recommend the following: When configuring the bus nodes in Hardware Manager, use the GNN as least significant byte of the IP address.

There is **no** non-volatile storage for the IP addresses of the following products:

Product	As of version
JC-310-JM	V. 1.22.0.00
JM-200-ETH	V. 1.22.0.00
Ethernet axis JM-xxx (JM-2xx-OEM)	V 2.07.0.37
Ethernet axis MC-JM-xxx (JM-2xx-OEM)	V 2.07.0.37
JX3-COM-EIPA	V. 1.01.0.00
JX3-COM-PND	V. 1.03.0.06

System launch of the bus nodes with non-volatile storage of the IP address

The network parameters assigned by NetConsistency are saved to the non-volatile store in the **config.ini** file of the bus nodes, if the DIP switch sliders 9 through 12 of the JX3-BN-ETH are in the position listed below.

DIP switch	Position
9	ON
10	OFF
11	OFF
12	OFF

The GNN of the bus nodes are configured via DIP switch sliders 1 through 8. The coding is binary, which means that, for example, switch 3 in position ON means GNN = 4.

At system launch, the bus nodes apply the network parameters which are stored in **/System/config.ini**. Immediately after this, the network parameters configured in Hardware Manager via JetControl - which is the NetConsistency master - are assigned to the bus nodes. If NetConsistency has already assigned the network parameters configured in Hardware Manager to the bus nodes, these bus nodes already use these for system launch.

The bus nodes store the assigned network parameters in the file **/System/config.ini** in the file system. In this case, the already existing file **/System/config.ini** is overwritten.

The DIP switches of the bus nodes set the GNN. This is for identifying the bus nodes within the system, so the network parameters configured in Hardware Manager can be assigned.

There is non-volatile storage for the IP addresses of the following products:

Product	As of version
JX3-BN-ETH	V. 1.18.0.02
JX3-COM-EIPA	V 1.05.0.02 (Beta-OS)
JX3-COM-PND	V 1.05.0.02 (Beta-OS)

Assigning the network parameters dependent on the GNN and transfer of the parameter and configuration data

Introduction

Via JetIPScan, NetConsistency sets the network parameters automatically for the following devices and automatically transfers the parameter and configuration data.

Via JetIPScan, NetConsistency sets the network parameters automatically for the following devices:

- Ethernet axes JM-xxx (JM-2xx-OEM, JM-200-ETH, JC-310-JM)
- Ethernet axes MC-xxx (JM-2xx-OEM, JM-200-ETH, JC-310-JM)
- JX3-BN-ETH
- JX3-COM-EIPA
- JX3-COM-PND

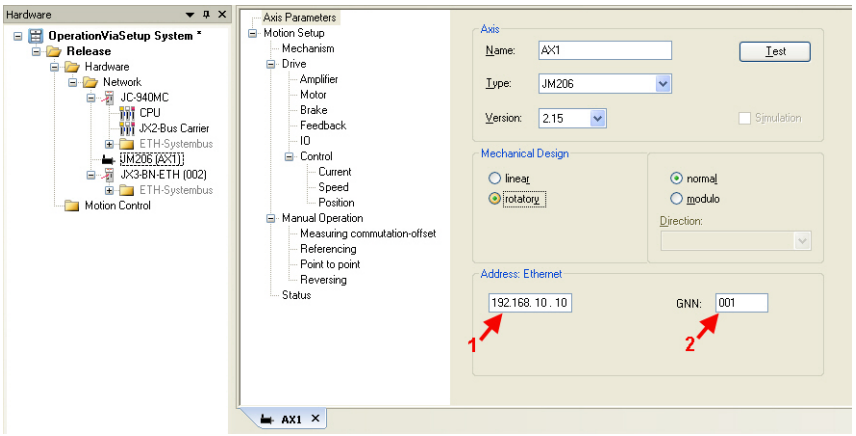
NetConsistency automatically transfers via FTP the parameter and configuration data to the following devices:

- JX3-BN-ETH
- JX3-COM-EIPA
- JX3-COM-PND

Automatically means that when exchanging a network node, you **only** have to take over the GNN (Global Node Number) which has got the same function as the settings of the DIP switch belonging to the former network node.

Any further settings are transmitted to the network node by the JetControl. Via JetIPScan, NetConsistency assigns the network parameters as set in Hardware Manager for the respective network nodes and transfers via FTP the parameter and configuration data as set in Hardware Manager for the respective network nodes.

Assigning the IP address and the GNN to the JM-200 with option -ETH



Step	Action
1	Set the GNN at the DIP switch (DIP switch sliders 1 through 8) of the MC-JM-xxx or JM-xxx.
2	Start JetSym.
3	Select the device MC-JM-xxx or JM-xxx in Hardware Manager.
4	Select the tab **Axis Parameters.**
5	As an address for **Ethernet Networks (1),** enter the IP address. **A special hint:** Use the GNN as least significant byte of the IP address.
6	As **GNN (2),** enter the Global Node Number of the device. The number has to match the settings of the DIP switch at the device.

Result: IP address and GNN have been assigned to the device.

Setting the DIP switch at the MC-JM-xxx or JM-xxx

The MC-JM-xxx or JM-xxx uses the settings of the DIP switch sliders 1 through 8 as GNN. The coding is binary.

Illustrations

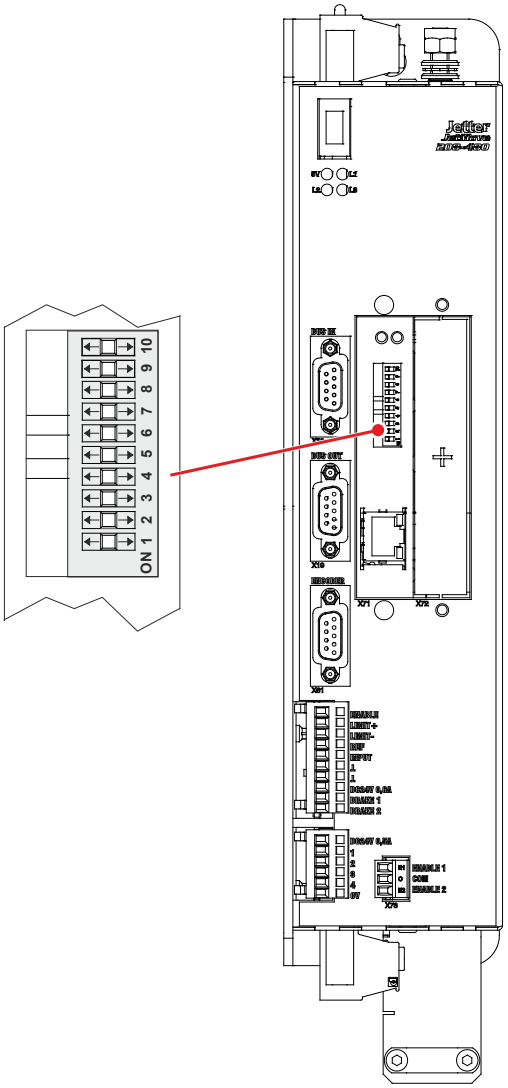
GNN = 4: Switch 3 is set to ON. All other DIP switch sliders are set to OFF.

GNN = 5: DIP switch sliders 1 and 3 are set to ON. All other DIP switch sliders are set to OFF.

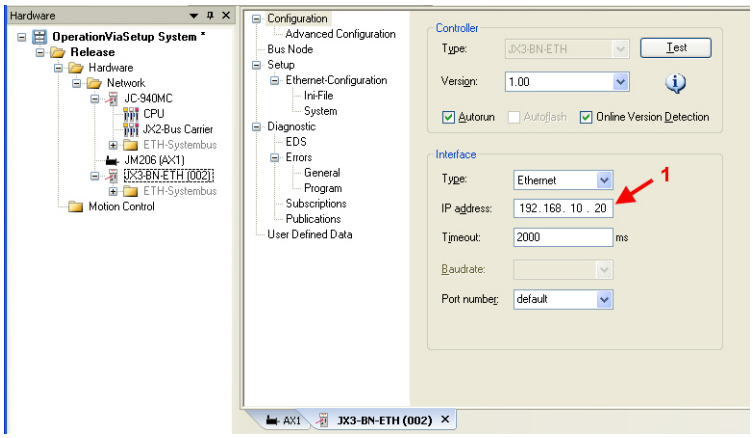
GNN = 8: Switch 4 is set to ON. All other DIP switch sliders are set to OFF.

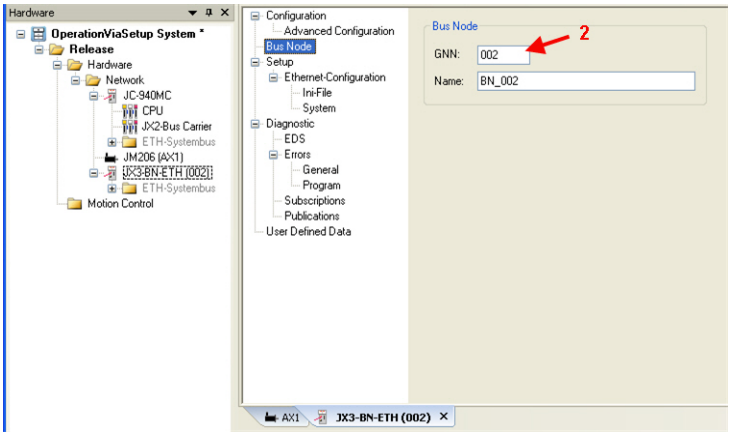
Position of the DIP switch sliders at the MC-JM-xxx or JM-xxx

If at the digital servo amplifier an Ethernet port is integrated, there is a 10-pin DIP switch available. The illustration below shows the position of the DIP switch sliders.



Assigning the IP address and the GNN to the JX3-BN-ETH and JX3-COM-EIPA/-PND





Step	Action
1	Set the GNN at the DIP switch (DIP switch sliders 1 through 8) of the JX3-BN-ETH or JX3-COM-EIPA/-PND.
2	Set the GNN operating mode at the DIP switch (DIP switch sliders 9 through 12) of the JX3-BN-ETH or JX3-COM-EIPA/-PND.
3	Start JetSym.
4	Select the device JX3-BN-ETH or JX3-COM-EIPA/-PND in Hardware Manager.
5	Select the tab **Configuration.**
6	As **IP Address (1),** enter the IP address.
7	Select the tab **Bus Node.**
8	As **GNN (2),** enter the Global Node Number of the device. The number has to match the settings of the DIP switch at the device.

Result: IP address and GNN have been assigned to the device.

Setting the DIP switch sliders at the JX3-BN-ETH and JX3-COM-EIPA/-PND

The settings of DIP switch sliders 9 through 12 activate remanent storage of the assigned network parameters in the **config.ini file**.

Set DIP switch slider 9 to ON and DIP switch sliders 10 through 12 to OFF.

The settings of DIP switch sliders 1 through 8 are for configuring the IP address. The coding is binary.

Illustrations

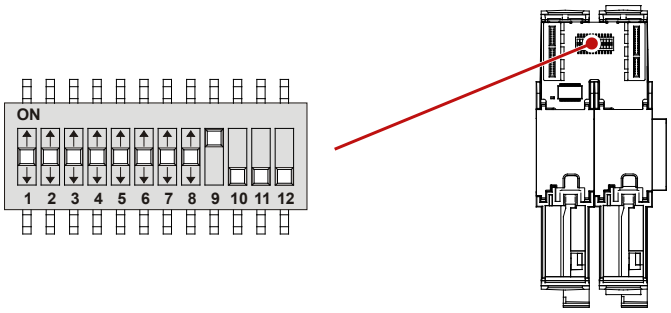
GNN = 4: Switch 3 is set to ON. All other DIP switch sliders are set to OFF.

GNN = 5: DIP switch sliders 1 and 3 are set to ON. All other DIP switch sliders are set to OFF.

GNN = 8: Switch 4 is set to ON. All other DIP switch sliders are set to OFF.

Position of the DIP switch sliders at the JX3-BN-ETH and JX3-COM-EIPA/-PND

The illustration below shows the position of the DIP switch sliders.



Network topology

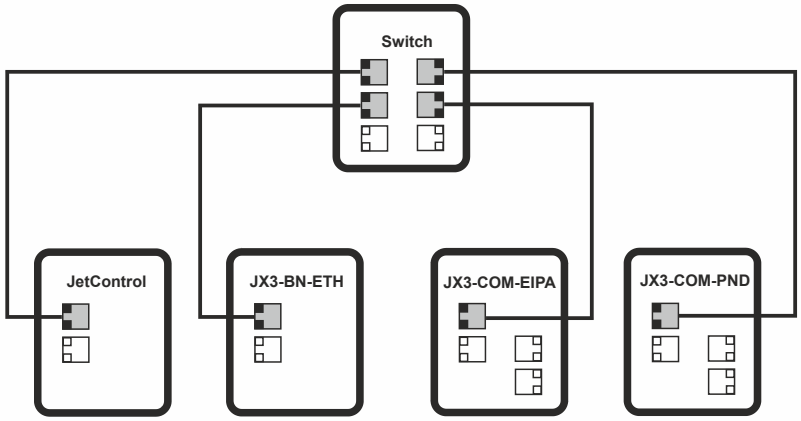
If NetConsistency is applied, the network nodes **must** be arranged in star-shaped topology, see figure *star-shaped topology*.

Background

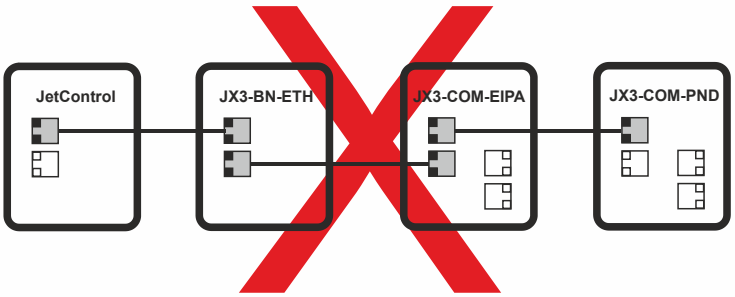
After transferring the parameter and configuration files via FTP, the controller reboots the respective bus nodes.

This means that the Ethernet switch placed on the bus nodes is rebooted and can therefore not forward any more network frames. For line topology - see figure *Line topology*, this means that, for example, JX3-BN-ETH, being the first bus node in the line topology of JetControl is given the command to reboot. Yet, JX3-COM-EIPA being the next bus node in the line cannot receive another reboot command given by the controller, because the Ethernet switch of the first bus node, JX3-BN-ETH, is rebooting at that moment. For this reason, using line topology together with NetConsistency is not permitted.

Star-shaped topology



Line-shaped topology



Compare program/Download

When you have set all parameters in Hardware-Manager, transfer the settings to the system parameters via **Compare program/Download**.

This is done by the following instruction in Hardware Manager:

- Compare program/Download (right mouse button on **release**)

Assigned network parameters

At system launch, the controller assigns the following network parameters to the connected network nodes:

- IP address
- Subnet mask
- Default gateway

IP address

The controller assigns the IP address as set in Hardware Manager.

Subnet mask

The controller assigns its own subnet mask.

Default gateway

The assigned default gateway depends on the controller type:

Product	Assigned default gateway	
JC-340, JC-350, JC-440MC	Default gateway of the controller	
JC-940MC	**If ...**	**... then ...**
	... neither with ETH2 nor with ETH3 network parameters have been configured,	... the controller assigns the default gateway of ETH1.
	... with ETH2 or with ETH3 network parameters have been configured,	... the controller assigns the IP address of ETH1 as the default gateway.
JC-945MC	**If ...**	**... then ...**
	... with ETH3 no network parameters have been configured,	... the controller assigns the default gateway of ETH1.
	... with ETH3 network parameters have been configured,	... the controller assigns the IP address of ETH1 as the default gateway.

Then, the controller transfers the parameter and configuration files to the network nodes and reboots the network node concerned.

Activating and deactivating JetIPScan in JetControl

Introduction

You have to enable JetIPScan by making an entry into the system command register. The settings are remanent.

Enable JetIPScan

To enable JetIPScan, proceed as follows:

Step	Action
1	Switch the device ON.
2	Write value 1112502132 (0x424f6f74) to password register 202960.
3	Enter value 331 into system command register 202961.
⇨	Bit 2 of register 202962 is set and JetIPScan is enabled.

Disable JetIPScan

To disable JetIPScan, proceed as follows:

Step	Action
1	Switch the device ON.
2	Write value 1112502132 (0x424f6f74) to password register 202960.
3	Enter value 330 into system command register 202961.
⇨	Bit 2 of register 202962 is cleared and JetIPScan is disabled.

Program run at system launch

Program run at system launch

The following table shows the program run at system launch:

Step	Description
1	When booting, the network nodes of non-volatile IP address storage take the IP address from the file **config.ini**.
2	When booting the JetControl, each network node is assigned a network configuration (IP address, subnet mask, gateway address) via JetIPScan while the NetConsistency function is executed. Files for network nodes holding parameter and/or configuration files are transferred via FTP. For uploading the new parameter and configuration data, the addressed network nodes are rebooted after file transfer.
3	After the booting the JetControl, and thus, after executing the NetConsistency function, the network nodes can be addressed via the network configurations as set in Hardware Manager. The parameter and configuration data of Hardware Manager are stored to the network nodes.

Program run at NetConsistency

NetConsistency passes the following states of the JetControl boot process:

Step	Description
1	The basic driver is initialized.
2	An instance is initialized.
3	The functions of NetConsistency are executed.

Register description - NetConsistency basic driver

Registers - Overview

Register	Description
470000 ... 470008	Cookie
470009	Version number
470010	Status
470011	Command
470020	Maximum possible amount of instances
470021	Number of instances ready for operation
470030 ... 470035	Restrictions
470040 ... 470157	Locating faults

R 470000 ... R 470008

Cookie

This register shows the beginning of the NetConsistency registers. This way, orientation is simplified.

Module register properties	
Type of access	Read
Value after reset	NetConsistency
Data type	RegString

R 470009

Version of NetConsistency

R 470009 shows the version of NetConsistency.

Module register properties	
Values	IP#0.00.0.00 ... IP#9.99.9.99
Type of access	Read
Value after reset	Version of NetConsistency

R 470010

Status register

R 470010 shows the status of the NetConsistency basic driver.

Meaning of the individual bits		
Bit 0	**Error**	
	0 =	No error
	1 =	Error

Bit 2	Status of initialization
	0 = Basic driver not initialized
	1 = Basic driver initialized

Module register properties	
Type of access	Read
Value after reset	0x00000004

R 470011

Command register

The value is 0, as there are no commands.

R 470020

Maximum possible number of instances

R 470020 shows the maximum possible number of NetConsistency instances. The actual value is always 1.

Module register properties	
Values	1
Type of access	Read
Value after reset	1

R 470021

Number of instances ready for operation

R 470021 shows the number of NetConsistency instances.

Module register properties	
Values	0 ... 1
Type of access	Read
Value after reset	1

R 470030

Maximum number of error messages for the logger

R 470030 sets the maximum number of error messages which are transferred to the logger by NetConsistency.

Module register properties	
Values	10
Type of access	Read
Value after reset	10

R 470031

Number of error messages transmitted to the logger

R 470031 displays the number of error messages transmitted to the logger by NetConsistency.

Module register properties	
Values	0 ... 10
Type of access	Read

R 470032

Maximum number of warnings for the logger

R 470032 sets the maximum number of warnings forwarded to the logger by NetConsistency.

Module register properties	
Values	10
Type of access	Read
Value after reset	10

R 470033

Number of warnings forwarded to the logger

R 470033 displays the number of warnings transmitted to the logger by NetConsistency.

Module register properties	
Values	0 ... 10
Type of access	Read

R 470034

Maximum possible number of error history entries

R 470034 defines the maximum possible number of error history entries.

Module register properties	
Values	10
Type of access	Read
Value after reset	10

R 470035

Number of entries in the error history

R 470035 displays the number of error messages entered into the error history by NetConsistency.

Module register properties	
Values	0 ... 30
Type of access	Read

R 470040

Error numbers

R 470040 shows the error numbers.

Error name	Error number
NoError	0
GroupFunction	-1
GroupCStandard	-2
GroupJetterFileSystem	-3
GroupJetterLogger	-4
GroupJetterOS	-5
GroupJetterParserXml	-6
GroupJetterPcom	-7
GroupUtility	-8
GroupJetIpScan	-9
Api	-100
Manager	-110
ManagerInit	-111
ManagerDeinit	-112
ManagerMultipleInit	-113
Instance	-120
InstanceInit	-121

Error name	Error number
InstanceDeinit	-122
StateMachine	-140
StateMachineInit	-141
StateMachineDeinit	-142
Error	-150
ErrorInit	-151
ErrorDeinit	-152
Warning	-160
WarningInit	-161
WarningDeinit	-162
Register	-170
RegisterInit	-171
RegisterDeinit	-172
Xml	-180
XmlInit	-181
XmlDeinit	-182
XmlInvalidGnn	-183
XmlInvalidIpAddress	-184
XmlTagNetConsistencyAttrVersion	-185
XmlTagNetNodesAttrCount	-186
XmlTagNetNodeAttrName	-187
XmlTagNetNodeAttrType	-188
XmlTagNetNodeAttrGnn	-189
XmlTagPcomAttrName	-190
XmlTagPcomAttrCommand	-191
XmlTagPcomAttrModuleId	-192
XmlTagPcomAttrTypeId	-193
XmlTagIpAddress	-194
XmlTagJetIPAttrPort	-195
XmlTagJx3SystembusAttrCrcEdsModuleCount	-196
XmlTagFilesAttrCount	-197
XmlTagFilesAttrCrc	-198
XmlTagFileAttrCrc	-199
XmlTagFileAttrPath	-200
XmlTagFileAttrName	-201
JetModuleReadReg	-300
JetModuleWriteReg	-301

Error name	Error number
Utility	-310
JetIPScan	-320
JetIPScanInit	-321
JetIPScanDeinit	-322
Processing	-330
ProcessingInit	-331
ProcessingDeinit	-332

Module register properties	
Values	-2^{16} ... 0
Type of access	Read

R 470041

Time of the error in milliseconds

R 470041 displays the time of the error in milliseconds. When JetControl has been activated for 50 days, an overflow occurs.

Module register properties	
Values	0 ... 2^{32} ms = 0 ... 50 days
Type of access	Read

R 470042

Instance, at which the error occurred

R 470042 displays the instance, at which the error occurred. In fact, only one instance is possible.

Module register properties	
Values	0: First instance
Type of access	Read

R 470043

Number of error parameters

R 470043 shows the number of error parameters.

Module register properties	
Values	0 ... 5
Type of access	Read

R 470044 **Error parameter 1**

R 470044 shows error parameter 1. The value is only valid, if R 470043 ≥ 1.

Module register properties	
Values	0 ... 2^{32}
Type of access	Read

R 470045 **Error parameter 2**

R 470045 shows error parameter 2. The value is only valid, if R 470043 ≥ 2.

Module register properties	
Values	0 ... 2^{32}
Type of access	Read

R 470046 **Error parameter 3**

R 470046 shows error parameter 3. The value is only valid, if R 470043 ≥ 3.

Module register properties	
Values	0 ... 2^{32}
Type of access	Read

R 470047 **Error parameter 4**

R 470047 shows error parameter 4. The value is only valid, if R 470043 ≥ 4.

Module register properties	
Values	0 ... 2^{32}
Type of access	Read

R 470048 **Error parameter 5**

R 470048 shows error parameter 5. The value is only valid, if R 470043 = 5.

Module register properties	
Values	0 ... 2^{32}
Type of access	Read

R 470049

Number of characters of the error message

R 470049 shows the number of characters of the error message. The error message has been stored to registers 470050 ... 470157.

Module register properties	
Values	0 ... 300
Type of access	Read

R 470050 ... R 470157

Text of the error message

These registers contain the text of the error message.

Module register properties	
Type of access	Read
Value after reset	""
Data type	RegString

Register description of the NetConsistency instance

Register overview

Register	Description
471010	Status
471011	Command

R 471010

Status register

R 470010 shows the status of the first NetConsistency instance.

Meaning of the individual bits		
Bit 0	**Error**	
	0 =	No error
	1 =	Error
Bit 2	**Status of initialization**	
	0 =	The first instance has not been initialized
	1 =	The first instance has been initialized
Bit 3	**Status of execution**	
	0 =	No execution
	1 =	Execution in process

Module register properties	
Type of access	Read
Value after reset	0x00000004

R 471011

Command register

The value is 0, as there are no commands.

Error handling at NetConsistency

Possibilities of error output

There are the following possibilities of error output:

- Via the logger of NetConsistency and JetIPScan
- Via the enhanced error register R 200009
- Via error number register R 200051 of JetIPScan
- Via error number register R 200061 of NetConsistency

R 200009

Enhanced error register

R 200009 is a bit-coded register.

Meaning of the individual bits		
Bit 12	**Error message by JetIPScan**	
	0 =	No error
	1 =	JetIPScan has reported an error. The error number is contained in R 200051.
Bit 16	**Error message by NetConsistency**	
	0 =	No error
	1 =	NetConsistency has reported an error. The error number is contained in R 200061 and R 470040.

Module register properties	
Type of access	Read

R 200051 **Error numbers of JetIPScan**

R 200051 shows the error numbers of JetIPScan. The content of this register is identical with JetIPScan MR 13.

Module register properties

Values	0	No error or warning
	5	The user has terminated the function
	1001	The first received response does not match response 2 and 3 (see MR 101x)
	1002	The second received response does not match response 1 and 3 (see MR 102x)
	1003	The third received response does not match response 1 and 2 (see MR 103x)
	-1	All 3 responses are dissimilar (see MR 100x)
	-2	The IP settings of at least one node could not be changed (see MR 140x)
	-3	The JetIPScan function has been invoked, although it is active already
	-10	The length of the set value list is < 1 or > 255, or the pointer to the list is invalid
	-11	A GNN of the set value list is < 1 or > 255, or it is a multiple GNN
	-20 ... -40	Internal error
	-1001 ... -1199	The node has reported the wrong CtrlID or CtrlIDopt (see MR 110x)
	-2001 ... -2199	The node has not called (see MR 120x)
	-3001 ... -3199	Several nodes of the same GNN have called (see MR 130x)
Type of access	Read	

R 200061 **Error numbers of NetConsistency**

R 200061 shows the error numbers of NetConsistency, see R 470040.

Related topics

- **Register description - NetConsistency basic driver** (see page 36)
- **Register description - JetIPScan** (see page 47)

2.3.2 JetIPScan - Register description

Introduction

This chapter describes the registers from which the status information of the JetIPScan feature can be read out. You can use these registers for debugging or diagnostics. Further features, such as, for example, checking the network configuration, cannot be triggered this way.

Contents

Register numbers

Introduction

Status information is 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
520000	520000 ... 522999

Determining the register number

In this chapter, only the last four figures of a register number are specified. e.g. MR 1499. Add to this module register number the basic register number of the corresponding device to determine the complete register number, for example 521499.

Registers - Overview

Register	Description
MR 0 ... MR 13	Global status
MR 1000 ... MR 1499	Warnings and errors
MR 2000 ... MR 2399	SET and ACTUAL configurations

Global status - Register description

Introduction

The current I/O size can be read from this register.

MR 0

State of the total

In MR 0. the controller signals a summary of status messages in bit-coded mode.

Meaning of the individual bits	
Bit 0	**Function enable** This bit corresponds to bit 2 of the system status register 202962. 0 = JetIPScan client - OFF 1 = JetIPScan client - ON
Bit 1	**Collective error message** 1 = Reg 13 contains value 0

Module register properties	
Type of access	Read
Value after reset	Bit 0: Depends on release status. Bit 1: 0

MR 10

State of execution

Corresponds to the feedback value *State*.

Module register properties

Values	0	The function is not active. Function terminated.
	1	Waiting for response from network nodes
	2	Send an inquiry frame
	3	Check the replies sent by the nodes
	4	Write the configurations of the nodes
Type of access	Read	

MR 11 **Number of cycles**

Corresponds to the feedback value *Count*.

Module register properties		
Values	0 ... 3	Number of cycles
Type of access	Read	

MR 12 **Number of changes**

Corresponds to the feedback value *Changed*.

Module register properties		
Values	0 ... 199	Number of changed network nodes
Type of access	Read	

MR 13 **Result of the function**

Corresponds to the feedback value *Result* and the register content of the global error number 2000051. This register indicates the value of the latest error or warning. Values greater than zero indicate warnings. Values smaller than zero are error messages.

Module register properties		
Values	0	No error or warning
	5	The user has terminated the function
	1001	The first received response does not match response 2 and 3 (see MR 101x)
	1002	The second received response does not match response 1 and 3 (see MR 102x)
	1003	The third received response does not match response 1 and 2 (see MR 103x)
	-1	All 3 responses are dissimilar (see MR 100x)
	-2	The IP settings of at least one node could not be changed (see MR 140x)
	-3	The JetIPScan function has been invoked, although it is active already
	-10	The length of the set value list is < 1 or > 255, or the pointer to the list is invalid
	-11	A GNN of the set value list is < 1 or > 255, or it is a multiple GNN
Values	-20 ... -40	Internal error

Module register properties

	-1001 ... -1199	The node has reported the wrong CtrlID or CtrlIDopt (see MR 110x)
	-2001 ... -2199	The node has not called (see MR 120x)
	-3001 ... -3199	Several nodes of the same GNN have called (see MR 130x)
Type of access	Read	

Warnings and errors - Register description

Introduction

Detailed diagnostics of the warnings and errors which have occurred can be carried out by means of these registers.

If, during checking and setting the IP address of all nodes a warning or an error occurs, the controller sets the corresponding bit in the registers described below. In this case, the bit corresponds to the GNN of the node.

The GNN of the node and the bit number relate as follows:

Bit number = GNN - 1

As a register contains 32 bit, individual groups of 7 subsequent registers each are created (see table).

Register bit	GNN
Register.0	1
Register.31	32
(Register + 1).0	33
(Register + 1).31	64
(Register + 2).0	65
(Register + 2).31	96
(Register + 3).0	97
(Register + 3).31	128
(Register + 4).0	129
(Register + 4).31	160
(Register + 5).0	161
(Register + 5).31	192
(Register + 6).0	193
(Register + 6).6	199

MR 1000 ... 1006

All 3 responses are dissimilar

The controller scans the network configuration three times and compares the three replies. If all three replies are dissimilar, the controller sets the respective bit in these registers.

Meaning of the individual bits	
Bit = 0	No error
Bit = 1	Error

Module register properties	
Bit number	GNN - 1

Type of access	Read

MR 1010 ... 1016

Reply no. 1 is not the same as replies 2 and 3

The controller scans the network configuration three times and compares the three replies. If replies 2 and 3 are the same, yet reply 1 is different, the controller sets the respective bit in these registers.

Meaning of the individual bits	
Bit = 0	No warning
Bit = 1	Warning

Module register properties	
Bit number	GNN - 1
Type of access	Read

MR 1020 ... 1026

Reply no. 2 is not the same as replies 2 and 3

The controller scans the network configuration three times and compares the three replies. If replies 1 and 3 are the same, yet reply 2 is different, the controller sets the respective bit in these registers.

Meaning of the individual bits	
Bit = 0	No warning
Bit = 1	Warning

Module register properties	
Bit number	GNN - 1
Type of access	Read

MR 1030 ... 1036

Reply no. 3 is not the same as replies 2 and 3

The controller scans the network configuration three times and compares the three replies. If replies 1 and 2 are the same, yet reply 3 is different, the controller sets the respective bit in these registers.

Meaning of the individual bits	
Bit = 0	No warning
Bit = 1	Warning

Module register properties	
Bit number	GNN - 1
Type of access	Read

MR 1100 ... 1106

Wrong CtrlID or CtrlIDopt

A node having got the required GNN has called, yet, the CtrlID or CTRLIDopt do not agree with it.

Meaning of the individual bits	
Bit = 0	No error
Bit = 1	Error

Module register properties	
Bit number	GNN - 1
Type of access	Read

MR 1200 ... 1206

The node has not called

The node having got the required GNN has not called.

Meaning of the individual bits	
Bit = 0	No error
Bit = 1	Error

Module register properties	
Bit number	GNN - 1
Type of access	Read

MR 1300 ... 1306

Multiple call

Several nodes using the same GNN have called. Yet, each node must have a unique GNN.

Meaning of the individual bits	
Bit = 0	No error
Bit = 1	Error

Module register properties	
Bit number	GNN - 1
Type of access	Read

MR 1400 ... 1406

The IP settings could not be changed

When the IP settings of a node have been changed, the controller checks whether the node has taken over these changes.

If the node has not taken over these changes, the controller sets the respective bit in these registers.

Meaning of the individual bits	
Bit = 0	No error
Bit = 1	Error

Module register properties	
Bit number	GNN - 1
Type of access	Read

Configuration - Register description

Introduction

These registers can be used to check the SET configuration and the three received ACTUAL configurations When you have entered the GNN in MR 2000, the controller transfers the values to the 4 register arrays.

MR 2000

GNN

Enter the GNN here.

Module register properties	
Values	1 ... 199
Value after reset	1

MR 2010 ... 2015

SET configuration

These registers let you read the default SET configuration.

Register	Command line parameter
2010	NodeID (GNN)
2011	CtrlID
2012	CtrlIDopt
2013	IpAddr
2014	IpMask
2015	Gateway

MR 2110 ... 2123

ACTUAL configuration 1

These registers let you read the first received ACTUAL configuration.

Register	Command line parameter
2110	NodeID (GNN)
2111	CtrlID
2112	CtrlIDopt
2113	IpAddr
2114	IpMask
2115	Gateway
2120	Quantity
2121	MAC address high
2122	MAC address low

Register	Command line parameter
2123	Sent IP address

MR 2210 ... 2223

ACTUAL configuration 2

These registers let you read the second received ACTUAL configuration.

Register	Command line parameter
2210	NodeID (GNN)
2211	CtrlID
2212	CtrlIDopt
2213	IpAddr
2214	IpMask
2215	Gateway
2220	Quantity
2221	MAC address high
2222	MAC address low
2223	Sent IP address

MR 2310 ... 2323

ACTUAL configuration 3

These registers let you read the third received ACTUAL configuration.

Register	Command line parameter
2310	NodeID (GNN)
2311	CtrlID
2312	CtrlIDopt
2313	IpAddr
2314	IpMask
2315	Gateway
2320	Quantity
2321	MAC address high
2322	MAC address low
2323	Sent IP address

2.4 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

2.4.1 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 62)
- **Terminating a connection** (see page 71)
- **Sending data** (see page 66)
- **Receiving data** (see page 68)

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 71)
- **Sending data** (see page 66)
- **Receiving data** (see page 68)
- **Initializing the user-programmable IP interface** (see page 61)

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 61)
- **Establishing a connection** (see page 62)
- **Terminating a connection** (see page 71)
- **Receiving data** (see page 68)

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 61)
- **Establishing a connection** (see page 62)
- **Terminating a connection** (see page 71)
- **Sending data** (see page 66)

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 62)
- **Sending data** (see page 66)
- **Receiving data** (see page 68)
- **Initializing the user-programmable IP interface** (see page 61)

2.4.2 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	

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.