Jetter automation

User Manual

JC-940MC - Controller

60879946

We automate your success.

Item # 60879946
Revision 1.05.3
November 2014 / Printed in Germany

Address

How to contact us:

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg
Germany

Phone - Switchboard:	+49 7141 2550-0
Phone - Sales:	+49 7141 2550-433
Phone - Technical Hotline:	+49 7141 2550-444
Fax - Sales:	+49 7141 2550-484
E-mail - Sales:	sales@jetter.de
E-mail - Technical Hotline:	hotline@jetter.de

Assignment to product

This User Manual is an integral part of JC-940MC:

Type: ________________

Serial #: ________________

Year of manufacture: ________________

Order #: ________________

To be entered by the customer:

Inventory #: ________________

Place of operation: ________________

Significance of this User Manual

This document is an integral part of the JC-940MC:

- Keep this document in a way that it is always at hand until the JC-940MC will be disposed of.
- Pass this document on if the JC-940MC is sold or loaned/leased out.

In any case you encounter difficulties to clearly understand the contents of this document, please contact Jetter AG.

We would appreciate any suggestions and contributions on your part and would ask you to contact us at the following e-mail address: info@jetter.de. Your feedback will help us produce manuals that are more user-friendly, as well as address your wishes and requirements.

This document contains important information on the following topics:

- Transport
- Mounting
- Installation
- Programming
- Operation
- Maintenance
- Repair

Therefore, you must carefully read, understand and observe this document, and especially the safety instructions.

In the case of missing or inadequate knowledge of this document Jetter AG shall be exempted from any liability. Therefore, the operating company is recommended to obtain the persons' confirmation that they have read and understood this manual in writing.

Table of Contents

1 Safety instructions

Introduction

This chapter informs the user of basic safety instructions. It also warns the user of residual dangers, if there are any. Furthermore, it contains information on EMC.

Contents

Basic safety instructions

Introduction

This device complies with the valid safety regulations and standards. Jetter AG attaches great importance to the safety of the users.

Of course, the user should adhere to the following regulations:

- Relevant accident prevention regulations
- Accepted safety rules
- EC guidelines and other country-specific regulations

Intended conditions of use

Usage according to the intended conditions of use implies operation in accordance with this User Manual.

The controller JC-940MC is used to control machinery, such as conveyors, production machines, and handling machines.

Operate the controller JC-940MC only within the limits and conditions set forth in the technical specifications. The operating voltage of the controller JC-940MC is classified as SELV (Safety Extra Low Voltage). Therefore, the JC-940MC controller is not subject to the EU Low Voltage Directive.

Usage other than intended

The device must not be used in technical systems which to a high degree have to be fail-safe, such as, for example, in ropeways and airplanes.

The JC-940MC is no safety-related part as per Machinery Directive 2006/42/EC. This device is not qualified for safety-relevant applications and must, therefore, NOT be used to protect persons.

If you intend to operate the device at ambient conditions not being in conformity with the permitted operating conditions, please contact Jetter AG beforehand.

Personnel qualification

Depending on the life cycle of the product, the persons involved must possess different qualifications. These qualifications are required to ensure proper handling of the device in the corresponding life cycle.

Product life cycle	Minimum qualification
Transport/storage:	Trained and instructed personnel with knowledge in handling electrostatic sensitive components.
Mounting/installation:	Specialized personnel with training in electrical engineering, such as industrial electronics technician.
Commissioning/ programming:	Trained and instructed experts with profound knowledge of, and experience with, electrical/drive engineering, such as electronics engineer for automation technology.
Operation:	Trained, instructed and assigned personnel with knowledge in operating electronic devices.
Decommissioning/ disposal:	Specialized personnel with training in electrical engineering, such as industrial electronics technician.

Modifications and alterations to the module	**For safety reasons, no modifications and changes to the device and its functions are permitted.** Any modifications to the device not expressly authorized by Jetter AG will result in a loss of any liability claims to Jetter AG. **The original parts are specifically designed for the device. Parts and equipment from other manufacturers have not been tested by Jetter AG and are, therefore, not released by Jetter AG.** The installation of such parts may impair the safety and the proper functioning of the device. Any liability on the part of Jetter AG for any damages resulting from the use of non-original parts and equipment is excluded.
Transport	The JC-940MC contains electrostatically sensitive components which can be damaged if not handled properly. To exclude damages to the JC-940MC during transport it must be shipped in its original packaging or in packaging protecting against electrostatic discharge. ▪ Use an appropriate outer packaging to protect the JC-940MC against impact or shock. ▪ In case of damaged packaging inspect the device for any visible damage. Inform your freight forwarder and Jetter AG.
Storing	When storing the JC-940MC observe the environmental conditions given in the technical specification.
Repair and maintenance	The operator is not allowed to repair the device. The device does not contain any parts that could be repaired by the operator. If the device needs repairing, please send it to Jetter AG.
Disposal	When disposing of devices, the local environmental regulations must be complied with.

Instructions on EMC

Noise immunity of a system

The noise immunity of a system is determined by the weakest component of the system. For this reason, correct wiring and shielding of cables is of paramount importance.

Measures

Measures for increasing EMI in electric plants:

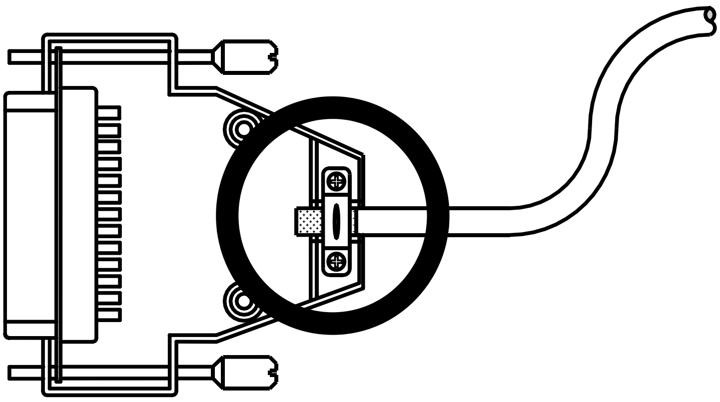
- Also refer to Application Note 016 *EMC-compatible installation of electric cabinets* by Jetter AG.

The following instructions are excerpts from Application Note 016:

- **Physically separate** signal and power lines. Jetter AG recommend spacing greater than 20 cm. Cables and lines should cross each other at an angle of 90°.
- The following line cables must be shielded:
  Analog lines, data lines, motor cables coming from inverter drives (servo output stage, frequency converter), lines between components and interference suppressor filter, if the suppressor filter has not been placed at the component directly.
- Shield cables **at both ends**.
- Unshielded wire ends of shielded cables should be as short as possible.
- The entire shield, **must, in its entire perimeter,** be drawn behind the isolation, and then be clamped under the earthed strain relief **with the greatest possible surface area.**

When male connectors are used:

- Draw the shield, in its entire perimeter, under the shielding clamp of the metallized connector housing (impedance shielding), respectively of the EMC gland bushing, its greatest possible surface area being clamped under a strain relief.
- Only use metallized connectors, e.g. Sub-D with metallized housing. Make sure that the strain relief is directly connected with the housing here as well.



Downloading Application Note 016

You can download Application Note 016 *EMC-Compatible Installation of Electric Cabinets* from the Jetter AG **homepage http://www.jetter.de**. In order to download Application Note 016, browse the following path: *Industrial Automation - Support - Downloads - 07_application_notes*.

2 Product description and design

Introduction

This chapter covers the design of the device, as well as how the order reference is made up including all options.

Contents

JC-940MC - Product description

The controller JC-940MC

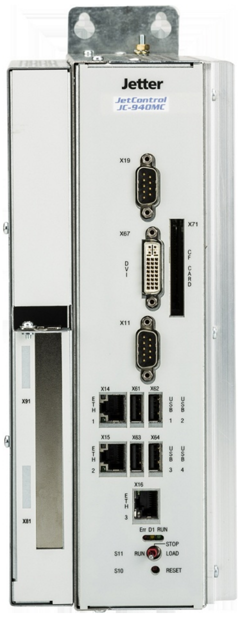
The JetControl 940MC is a high-end compact controller of excellent performance in combination with motion control.

The JC-940MC offers a motion control feature which allows programming axis groups and complex path controls. Special attention was given to straightforward and efficient realization of complex path controls.

The controller can be integrated into a network via its Ethernet ports.

Product features

The features of this product are listed below:



- Point-to-point axes without limitations
- Up to 64 servo axes with path control clock: 2 ms
- 3 Ethernet ports
- 4 USB ports
- Powerful programming language JetSym STX
- Non-volatile registers 120,000
- Program/data memory: 8 MBytes
- Real-time clock
- Integrated Web server/e-mail feature
- Modbus/TCP

Scope of delivery

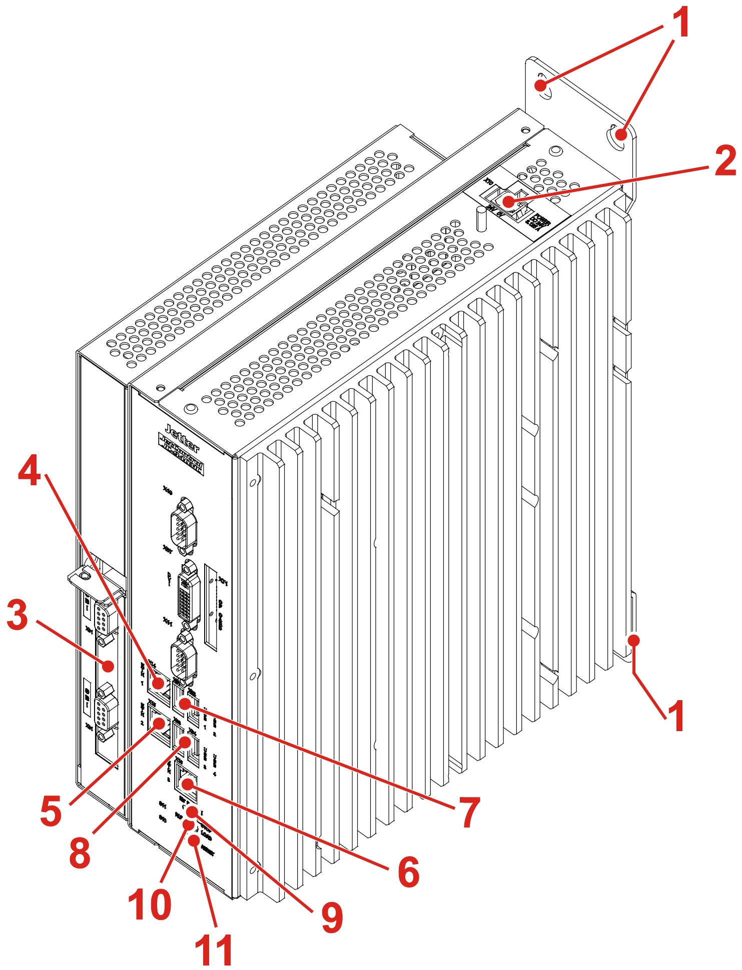
The following items are included in the scope of delivery of the controller JC-940MC:

Item no.	Quantity	Description
10000742	1	The controller JC-940MC
-	1	2-pin connector, spring-cage connection
60874441	1	Installation manual

Parts and interfaces

Parts and interfaces

The controller JC-940MC features the following parts and interfaces:



Number	Part	Description
1	Fastening screw threads	For fixing the controller in the electric cabinet
2	X10	Power supply
3	PCI slot for the module board and JX6 submodules	You can connect two JX2 system busses
4	X14	Ethernet port ETH 1
5	X15	Ethernet port ETH 2
6	X16	Ethernet port ETH 3
7	X61 and X62	USB port 1 and 2
8	X63 and X64	USB port 3 and 4
9	LEDs	Diagnostic and status LEDs

Number	Part	Description
10	S11	RUN - STOP - LOAD mode selector
11	S10	Reset button

Order references/options

Order reference

The order reference consists of the name of the controller JC-940MC and the desired options. Each of the options given below supplements the controller JC-940MC. The order reference reflects only existing options.

JC-940MC	-	A	-	E

Part	Description
JC-940MC	Controller
A	Number of MC-axes (path control): 16, 24 or 64 There is no functional limitation to the number of PtP axes. This limitation is independent of the system bus. JX2 system bus: 16 PtP axes max. per JX6-SB-I Ethernet system bus: 250 PtP axes max.
E01	One PCI slot with two submodules JX6-SB-I
E03	One PCI slot, not assembled

Item no.	Order reference	
10000742	JC-940MC	64 MC axes; no PCI-expansion
10000857	JC-940MC-16	16 MC axes; no PCI-expansion
10000859	JC-940MC-24	24 MC axes; no PCI-expansion

Ordering additional options

Specify your desired options in the order. The controller cannot be equipped with additional features afterwards.

Integrated Web server and e-mail feature

If the controller JC-940MC is equipped with integrated Web server and e-mail feature, it is able to perform the following functions:

- **HTTP server**: The user downloads the homepages into the controller via FTP. They can be accessed with any standard internet browser.
- **SMTP client**: The controller sends e-mails.

Modbus/TCP

The controller JC-940MC can be equipped with Modbus/TCP protocol. The controller can act as both server and client.

List of documentation

Introduction

Various documents and software tools will support the user in engineering, installing and programming the JC-940MC controller. You can download these documents and software tools from the Jetter AG **homepage** **http://www.jetter.de**.

Engineering

The following documents and files support you at engineering the controller:

	Industrial automation catalog
	▪ Product description
	▪ Technical specifications
	Manual on the controller JC-3xx
	▪ The document at hand

Engineering a JX2 station on the JX2 system bus

The following document and the following software tool support you when engineering a JX2 station at the JX2 system bus (JC-9xx):

	JX6-SB - User information
	▪ System bus topology
	▪ JX2 system bus specification
	System bus configurator
	▪ Excel file for designing the JX2 system bus
	▪ SysBus_Configuration_xxx_e.xls (xxx: Version)

Engineering a JX3 station on the JX3 system bus

The following document and the following software tool support you when engineering a JX3 station at the JX3 system bus (JX3-BN-ETH):

	User manual on the bus node JX3-BN-ETH
	▪ Engineering a JX3 station
	▪ Product descriptions of JX3 modules
	System bus configurator
	▪ Excel file for designing the JX3 system bus
	▪ SysBus_Configuration_xxx_e.xls (xxx: Version)

Installation

The following documents support you at installing the controller:

	Installation manual
	It is included in the boxed controller JC-940MC and contains information on:
	▪ Installing the controller
	▪ Terminal assignment
	▪ Specification of terminals
	▪ Diagnostics via LEDs
	Manual on the controller JC-3xx
	▪ The document at hand

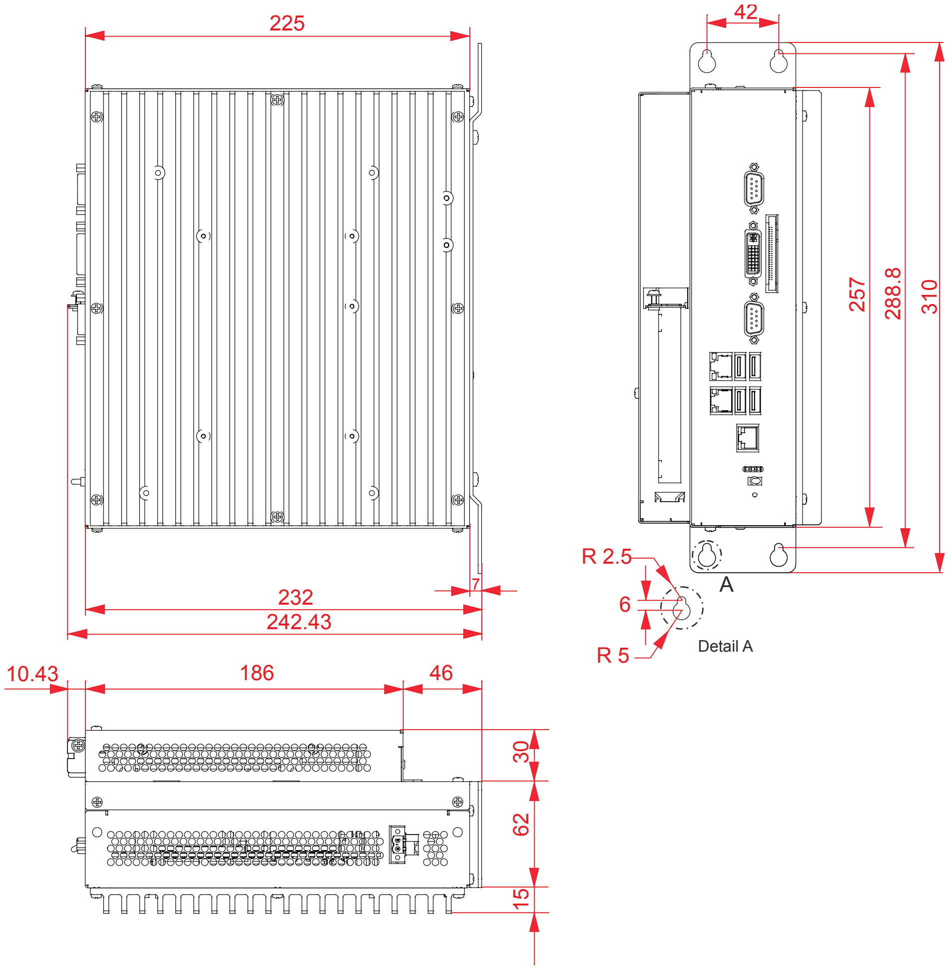
Programming

The following documents and files support you at programming the controller:

	Manual on the controller JC-3xx
	▪ The document at hand
	JX6-SB - User information
	▪ Configuring the JX2-system bus
	▪ Module numbering system
	▪ Diagnostics of modules on the JX2 system bus
	JetSym
	▪ Programming tool

Physical dimensions

Physical dimensions



Minimum clearances	At mounting the controller JC-940MC, a minimum clearance above, below and on the right-hand side (side where the heat sink is positioned) must be maintained. ■ Minimum clearance, above: 100 mm ■ Minimum clearance, below: 100 mm ■ Minimum clearance, right-hand side: 50 mm
Module width	The width of the controller JC-940MC is 77 mm. The corresponding module boards (PCI slots) have a width of 30 mm each. In case of option E, they widen the dimensions of the controller. The width of the controller JC-940MC-E01 is 107 mm, for example.
Mounting orientation	The orientation of the controller JC-940MC is vertical.

3 Identifying the controller

Purpose of this chapter

This chapter is for supporting you in identifying the following information with regard to JC-940MC:

- Determining the hardware revision
- Reading the electronic data sheet EDS. Numerous production-relevant data are stored in the EDS.
- Determining the OS version of the controller and its software components

Prerequisites

To be able to identify the JC-940MC controller, the following prerequisites must be fulfilled:

- The controller is connected to a PC.
- The programming tool JetSym 4.2 or higher is installed on the PC.

Information for hotline requests

If you wish to contact the hotline of Jetter AG in case of a problem, please have the following information on the JC-940MC controller ready:

- Serial number
- OS version number of the controller
- Hardware revision

Contents

3.1 Identification by means of the nameplate

Introduction

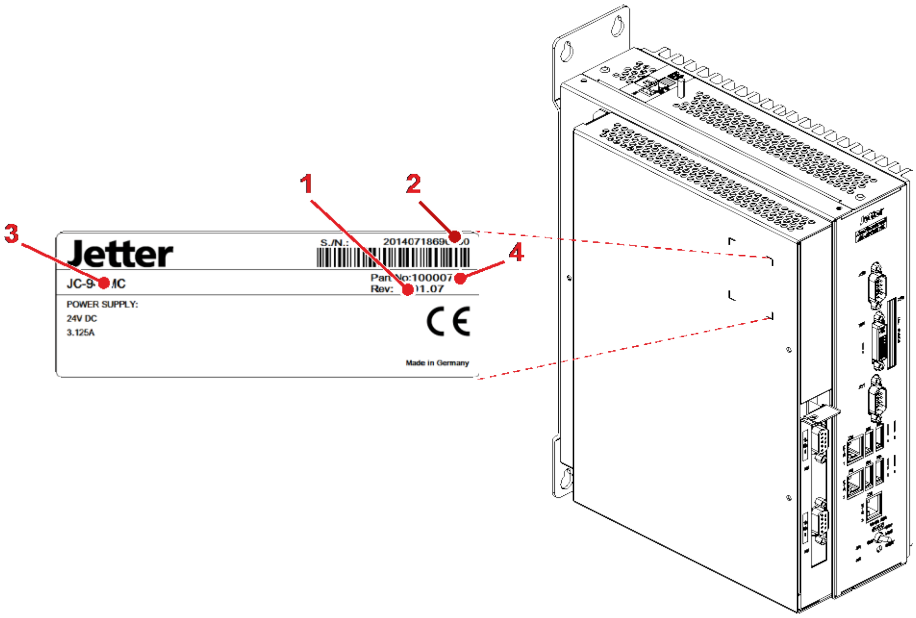
The nameplate is attached to the housing of the JC-940MC and contains details, such as hardware revision number and serial number. If you wish to contact the hotline of Jetter AG in case of a problem, please have this information ready.

Contents

Nameplate

Nameplate

The nameplate of a JC-9xxMC controller contains the following information:



Number	Description
1	Hardware revision
2	Serial number
3	Controller name
4	Item number

3.2 Electronic Data Sheet EDS

Introduction

Each JC-940MC features an Electronic Data Sheet (EDS). Numerous production-relevant data are permanently stored in the EDS. The EDS data can be read out via files in the file system of the JC-940MC or via special registers.

Contents

EDS File "eds.ini"

Introduction

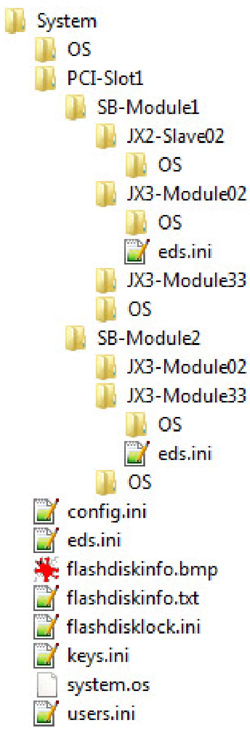
EDS data can be read via the file **eds.ini**.

Properties

- You can access this file through the file system of the controller.
- For an FTP connection, the user needs administrator rights (e.g. user *admin*) or system rights (e.g. user *system*).
- The EDS file of the controller is located in the directory */System*.
- This file allows only read access.
- Formatting the flash disk does not influence the file.

Path to EDS files

The illustration below shows an example of the directory */System* containing the EDS files of the controller:



File structure

The EDS file is a text file the entries of which are grouped into several sections.

Example - Controller

This is an example of an EDS file belonging to a JetControl 940MC:

```
;Jetter AG Electronic Data Sheet

[IDENTIFICATION]
Version = 0
Code = 2304
Name = JC-940MC
```

```
PcbRev = 02
PcbOpt = 00

[PRODUCTION]
Version = 0
SerNum = 10080703010015
Day = 4
Month = 7
Year = 2010
TestNum = 1
TestRev = 01.10.03.50

[FEATURES]
Version = 0
Axes = 16
NumberOfPCISlots = 00
STX = 1
NVRegs = 120000
```

Section [IDENTIFICATION]

The general hardware configuration can be seen from section [IDENTIFICATION].

Name	Example	Description
`Version`	0	Version of this section
`Code`	2304	Module code for JC-940MC
`Name`	JC-940MC	Corresponds to the information on the nameplate
`PcbRev`	02	Hardware revision
`PcbOpt`	00	Hardware option

Section [PRODUCTION]

The serial number and production date can be seen from section [PRODUCTION].

Name	Example	Description
`Version`	0	Version of this section
`SerNum`	10080703010015	Corresponds to the information on the nameplate
`Day`	04	Production date: Day
`Month`	07	Production date: Month
`Year`	2010	Production date: Year
`TestNum`	1	Internal usage
`TestRev`	01.10.03.50	Internal usage

Section [FEATURES]

In the section [FEATURES] special properties of the controller are specified. The OS of the controller will ignore properties of missing entries in the file.

Name	Example	Description
`Version`	0	Version of this section
`Axes`	16	Number of MC-axes
`NumberofPCI slots`	00	Number of PCI slots
`STX`	1	Runtime environment for application program is available
`NVRegs`	120000	Number of remanent registers

Related topics

- **EDS registers** (see page 32)

EDS registers

Introduction

EDS registers let you retrieve entries in the Electronic Data Sheet (EDS).

Register numbers

The basic register number is dependent on the controller. The register number is calculated by adding the number of the module register (MR) to the number of the basic register.

Controller	Basic register number	Register numbers
JC-940MC	100000	100600 ... 100817

EDS registers of a controller

The following table lists the EDS registers of a controller, as well as their connection to the entries in the EDS file **/System/eds.ini**.

Registers	Section in eds.ini	Name in eds.ini	Description
MR 600	IDENTIFICATION	`Version`	Version of this section
MR 601		`Code`	Module code
MR 602 through 612		`Name`	Module name or controller name
MR 613		`PcbRev`	Hardware revision
MR 614		`PcbOpt`	Hardware option
MR 700	PRODUCTION	`Version`	Version of this section
MR 701 through 707		`SerNum`	Serial number
MR 708		`Day`	Production date: Day
MR 709		`Month`	Production date: Month
MR 710		`Year`	Production date: Year
MR 711		`TestNum`	Internal usage
MR 712		`TestRev`	Internal usage
MR 800	FEATURES	`Version`	Version of this section
MR 804		`Switch`	RUN-STOP-LOAD switch
MR 805		`STX`	Runtime environment for the application program
MR 806		`NVRegs`	Number of remanent registers
MR 810		`MotionControl`	MC software
MR 811		`Axes`	Number of MC-axes
MR 812		`Web`	Web server and e-mail client
MR 813		`ModbusTCP`	Modbus/TCP client and server
MR 817		`RTC`	Real-time clock

Related topics

- **EDS File** *eds.ini* (see page 29)

3.3 Version registers

Introduction

The operating system of the JC-940MC controller provides several registers which can be used to read out the revision numbers of the hardware or of the operating system and its components. If you wish to contact the hotline of Jetter AG in case of a problem, please have this revision ready.

Contents

Hardware revisions

Introduction

The controller JC-940MC features special registers, the content of which lets you identify the hardware.

Register overview

The following registers let you read out the hardware revisions:

Registers	Description
200170	Controller type

Version numbers in JetSym setup

The following screenshot shows a JetSym setup window displaying the version registers:

	Name	Number	Content	Type	Comment
1					
2	JCtype	200170			
3					
4					

Related topics

- **Software versions** (see page 36)

Software versions

Introduction

The controller JC-940MC features software with unique version numbers which can be read out via special registers.

Format of software version numbers

The software version numbers of the controller JC-940MC are four-figure values.

1	**.**	**2**	**.**	**3**	**.**	**4**

Element	Description
1	Major or main version number
2	Minor or secondary version number
3	Branch or intermediate version number
4	Build version number

Released version

A released version can be recognized by both Branch and Build having got the value 0.

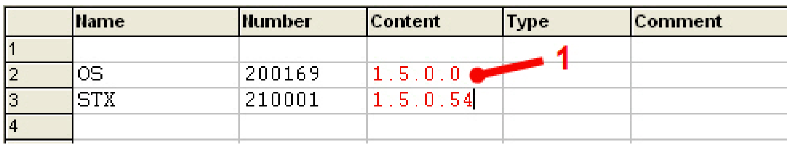
Register overview

The following registers let you read out the software versions:

Registers	Description
200169	Operating system version
210001	Version of the runtime unit for the STX application program

Version numbers in JetSym setup

The following screenshot shows a JetSym setup window displaying version registers. To have the version number displayed in the setup window of JetSym, select the format **IP address**.



Number	Description	Function
1	V 1.05.0.00	OS version number of the controller JetSym displays this information in the title bar of each setup window.

Related topics

- **Hardware revisions** (see page 35)

4 Mounting and installation

Purpose of this chapter

This chapter is for supporting you in mounting and installing the JC-940MC controller as regards the following points:

- Wiring the JC-940MC controller
- Description of the display items
- Description of control elements
- Installation
- Battery replacement

Contents

4.1 Interfaces

Terminal X10

The function of terminal X10 is as follows:

- Power supply for the controller JC-940MC

Jack X14

Other than with jack X15 and X16, the Ethernet system bus from Jetter AG is implemented in jack X14. For this reason, jack X14 is the only port to connect a JX3-BN-ETH or a JetMove-200-ETH to. The function of jack X14 is as follows:

- Ethernet port to a hub, switch or router
- Ethernet port to a PC
- Ethernet port to an HMI by Jetter AG
- Ethernet port to a JX3-BN-ETH or a JetMove-200-ETH
- Ethernet port to any device

Jacks X15 and X16

The function of jacks X15 and X16 is as follows:

- Ethernet port to a hub, switch or router
- Ethernet port to a PC
- Ethernet port to an HMI by Jetter AG
- Ethernet port to any device

Jack X61

The function of jack X61 is as follows:

- USB interface to a USB flash drive
- Copying controller data automatically from or to a USB flash drive via sub-directory USB1 of the controller file system is possible.

Jack X62, X63 and X64

The function of jacks X62, X63 and X64 is as follows:

- USB interface to a USB flash drive

PCI module board (option)

Up to two JX6 expansion modules can be connected to the PCI module board.

Contents

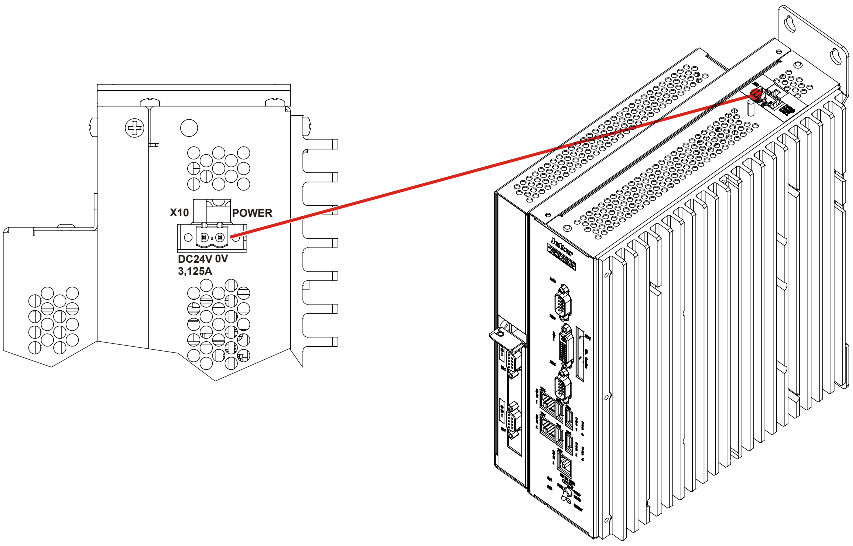
Power supply terminal X10

Terminal interface

X10 lets you connect the following devices:

- Power supply for the controller JC-940MC

Terminal assignment



Terminal point	Description
DC24V 3.125A	Power supply for controller JC-940MC
0V	Reference potential

Technical specifications

Parameter	Description
Rated voltage	DC 24 V
Permissible voltage range	-15 % ... +20 %
Input current	3.125 A max.
Power consumption	75 W max.

Connector for power supply terminal X10

A 2-pin connector is included in the scope of delivery of the controller JC-940MC.

Terminal

Parameter	Description
Technology	Screw terminal
Screwdriver	SZS 0.6 x 3.5
AWG	12 ... 24
Single conductor	0.2 mm^2 ... 2.5 mm^2
Flexible conductor	0.2 mm^2 ... 2.5 mm^2
With wire end ferrule	0.25 mm^2 ... 2.5 mm^2
Wire end ferrule with sleeve	0.25 mm^2 ... 2.5 mm^2

Ethernet port ETH 1, jack X14

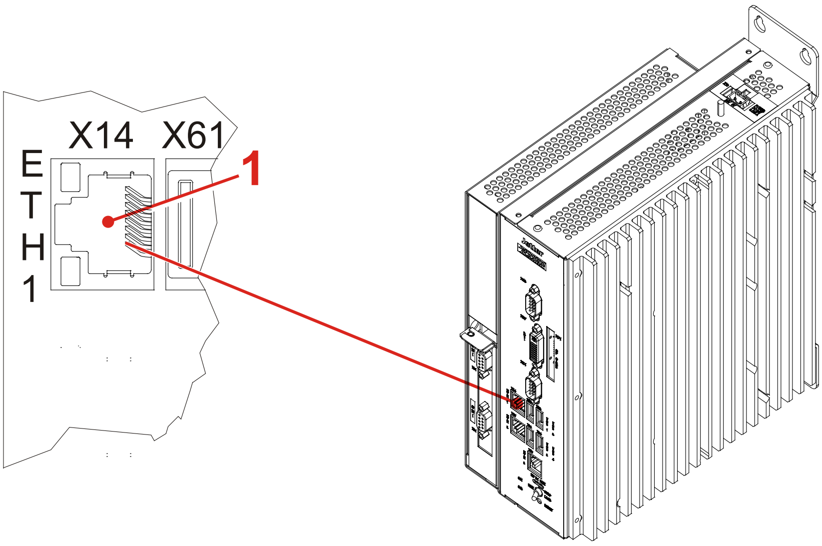
Devices to connect with jack X14

Other than with jack X15 and X16, the Ethernet system bus from Jetter AG is implemented in jack X14.

Jack X14 lets you connect the following devices:

- Ethernet cable to a hub, switch or router
- Ethernet cable to a PC
- Ethernet cable to an HMI by Jetter AG
- Ethernet cable to a JX3-BN-ETH or a JetMove-200-ETH
- Ethernet cable to any device

Position of jack X14



Number	Description
1	Jack X14 - Ethernet port

Technical specifications

Parameter	Description
Connector type	RJ45 Ethernet jack
Number of ports	One port per connector
Bit rate	10 MBit/s, 100 MBit/s (Cat 5e) 1,000 MBit/s (Cat 6)
Auto-crossover	Yes

Cables for jack X14

For connecting devices to jack X14 you can order the following cables:

Item no.	Item
60537500	Patch cable 1:1, 1 m gray Hirose, Cat 5e, shielded
60854512	Patch cable 1:1, 2 m gray Hirose, Cat 5e, shielded
60854514	Patch cable 1:1, 5 m gray Hirose, Cat 5e, shielded
60854515	Patch cable 1:1, 10 m gray Hirose, Cat 5e, shielded

Ethernet port ETH 2, jack X15

Devices to connect with jack X15

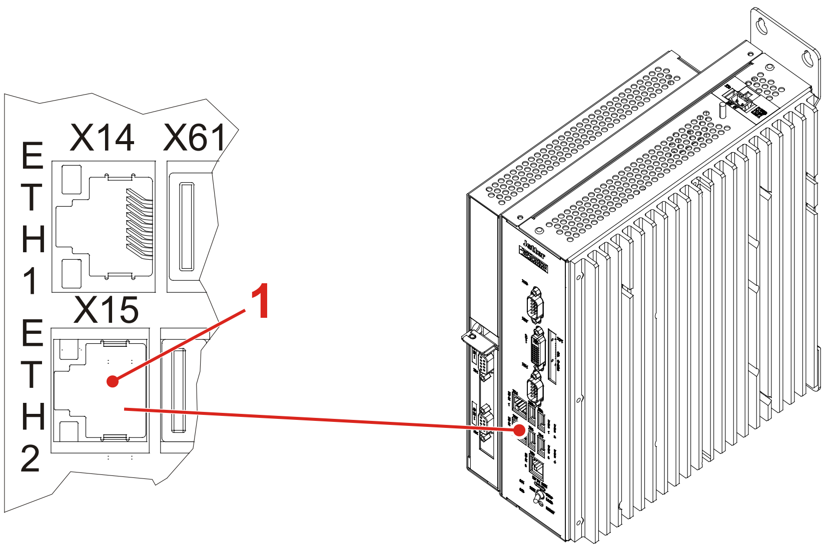
Jack X15 lets you connect the following devices:

- Ethernet cable to a hub, switch or router
- Ethernet cable to a PC
- Ethernet cable to an HMI by Jetter AG
- Ethernet cable to any device

Asynchronous transmission

Port ETH 2 only allows asynchronous transmission of the data packets. Synchronous transmission of the data packets is not allowed. Therefore, reasonable communication via Ethernet with a JX3-BN-ETH or with a JetMove-200-ETH, for example, is not possible.

Position of jack X15



Number	Description
1	Jack X15 - Ethernet port

Technical specifications

Parameter	Description
Connector type	RJ45 Ethernet jack
Number of ports	One port per connector
Bit rate	10 MBit/s, 100 MBit/s (Cat 5e) 1,000 MBit/s (Cat 6)
Auto-crossover	Yes

Cables for jack X15

For connecting devices to jack X15 you can order the following cables individually:

Item no.	Item
60537500	Patch cable 1:1, 1 m gray Hirose, Cat 5e, shielded
60854512	Patch cable 1:1, 2 m gray Hirose, Cat 5e, shielded
60854514	Patch cable 1:1, 5 m gray Hirose, Cat 5e, shielded
60854515	Patch cable 1:1, 10 m gray Hirose, Cat 5e, shielded

Ethernet port ETH 3, jack X16

Devices to connect with jack X16

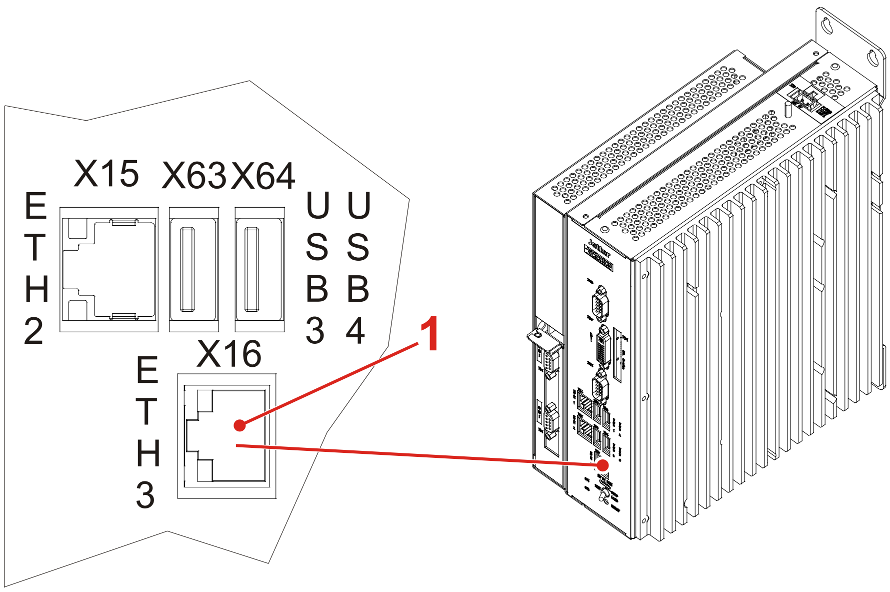
Jack X16 lets you connect the following devices:

- Ethernet cable to a hub, switch or router
- Ethernet cable to a PC
- Ethernet cable to an HMI by Jetter AG
- Ethernet cable to any device

Asynchronous transmission

Port ETH 3 only allows asynchronous transmission of the data packets. Synchronous transmission of the data packets is not allowed. Therefore, reasonable communication via Ethernet with a JX3-BN-ETH or with a JetMove-200-ETH, for example, is not possible.

Position of jack X16



Number	Description
1	Jack X16 - Ethernet port

Technical specifications

Parameter	Description
Connector type	RJ45 Ethernet jack
Number of ports	One port per connector
Bit rate	10 MBit/s, 100 MBit/s (Cat 5e)
Auto-crossover	Yes

Cables for jack X16

For connecting devices to jack X16 you can order the following cables individually:

Item no.	Item
60537500	Patch cable 1:1, 1 m gray Hirose, Cat 5e, shielded
60854512	Patch cable 1:1, 2 m gray Hirose, Cat 5e, shielded
60854514	Patch cable 1:1, 5 m gray Hirose, Cat 5e, shielded
60854515	Patch cable 1:1, 10 m gray Hirose, Cat 5e, shielded

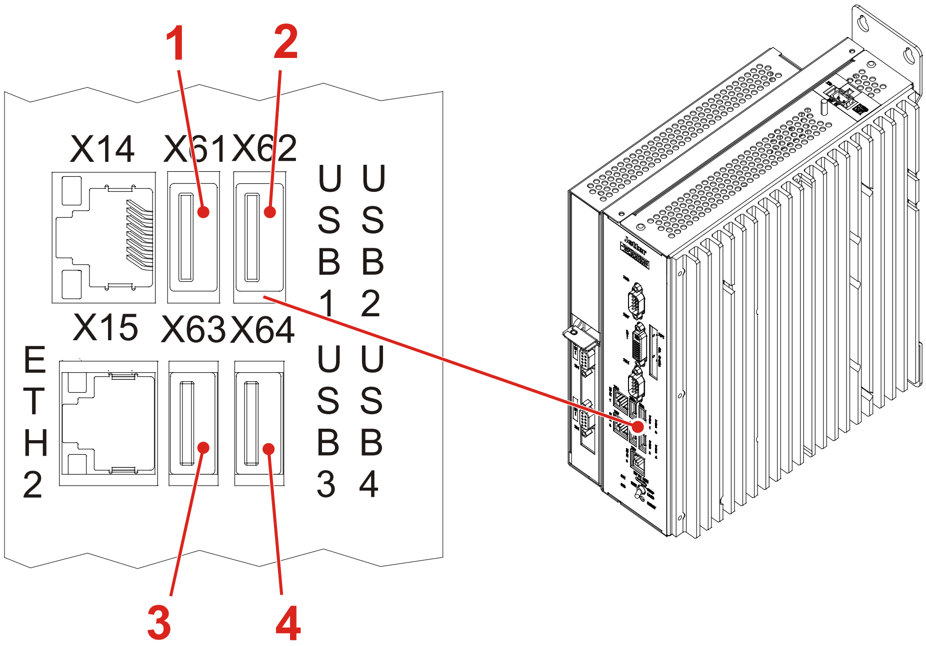
USB port - Jacks X61 to X64

Device to connect with the respective jack

To each of jacks X61 to X64, one USB flash drive can be connected. Only one USB flash drive at a time can be connected.

By means of the AutoCopy function, copying of controller data automatically from or to a USB flash drive via sub-directory USB1 of the file system is possible. For this, the USB flash drive must be connected to jack X61.

Position of jacks X61 and X62



Number	Description
1	Jack X61, USB1 port
2	Jack X62, USB2 port
3	Jack X63, USB3 port
4	Jack X64, USB4 port

Technical specifications

Parameter	Description
Maximum output current	0.5 A
USB type	Type A (host)
Specification	USB 2.0
Maximum permissible cable length	5 m

4.2 JX2 system bus interface - communicating via PCI bus

Introduction

This chapter gives a description of the JX2 system bus interface of the controller JC-940MC - communicating via PCI bus.

Possible number of JX2 system busses

At the PCI bus, up to two JX2 system busses can be connected. The controller JC-940MC is available either without or with one module board. Into each module board, two JX6-SB(-I) cards can be plugged.

Submodules JX6-SB and JX6-SB-I take on converting the PC-internal PCI bus to the JX2 system bus.

Connectable modules

The following modules can be connected to the JX2 system bus of the controller JC-940MC:

- JX2-I/O modules
- JX2 slave modules
- Servo amplifiers JetMove 1xx, JetMove 2xx, and JetMove 6xx
- IP67 modules LioN-S and LJX7-CSL
- Third-party CANopen® modules, e.g. valve terminals

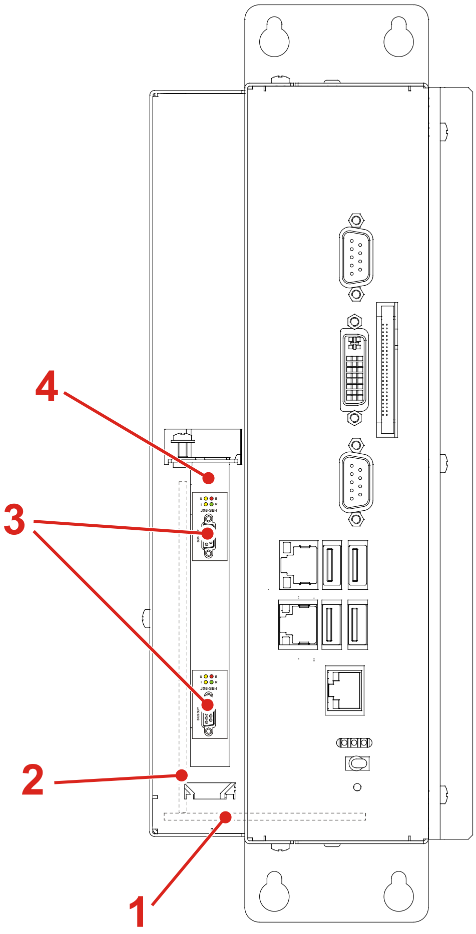
Expandability

To each of the JX2 system busses, up to 31 JX2-I/O modules, 10 Smart-I/O modules, third-party CANopen® modules, as well as 16 JX2 slave modules or JetMoves can be connected.

Contents

Configuration of the controller JC-940MC-E01

Configuration



Number	Part	Description
1	PCI bus	Extended PCI bus
2	Module board	Connection between PCI bus and JX2 system bus
3	BUS OUT	JX2 system bus interface
4	PCI slot	To each PCI slot, two JX2 system busses can be connected

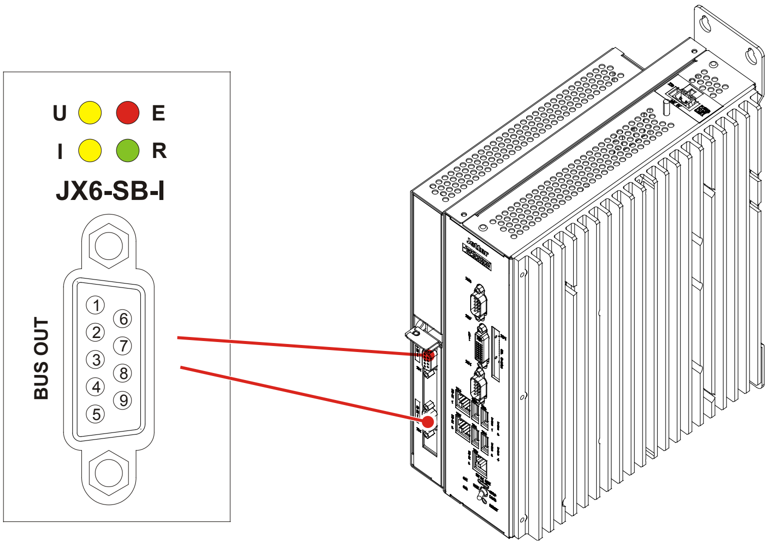
JX2 system bus - Pin assignment

JX2 system bus interface

The Sub-D connector of the submodule JX6-SB(-I) is the interface to the JX2 system bus by Jetter AG. The following modules can be connected to the JX2 system bus:

- JX2-I/O modules
- JX2 slave modules
- Servo amplifiers JetMove 1xx, JetMove 2xx, and JetMove 6xx
- IP67 modules LioN-S and LJX7-CSL
- Third-party CANopen® modules, e.g. valve terminals

Sub-D connector - Pin assignment



Pin	Signal	Description
1	CMODE0	Commissioning
2	CL	Data signal
3	GND	Reference potential
4	CMODE1	Commissioning
5	Unused	
6	Unused	
7	CH	Data signal
8	Unused	
9	Unused	

Technical specifications

Parameter	Description
Type of terminal	Sub-D connector
Number of pins	9
Electrical isolation	None
Baud rates	1,000/500/250/125 kBaud

Suitable cables

For connecting modules to the JX2 system bus you can order the following cables individually:

Item no.	Item
10309001	Cable assy # 530 0.2 m
10309002	Cable assy # 530 0.5 m
10309003	Cable assy # 530 1.0 m
10309004	Cable assy # 530 1.5 m
10309006	Cable assy # 530 2.0 m
10309016	Cable assy # 530 2.5 m
10309015	Cable assy # 530 3.0 m
10309007	Cable assy # 530 4.0 m
10309008	Cable assy # 530 5.0 m

Related topics

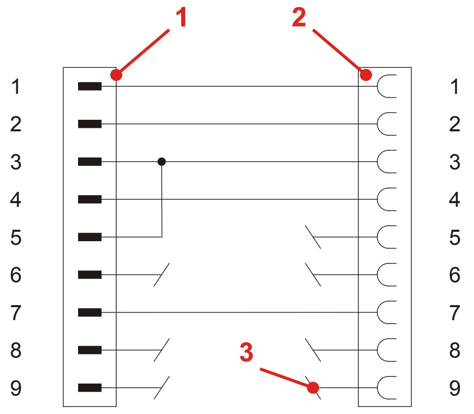
- **JX2 system bus - Cable specification** (see page 52)
- **Line length (in mm) and baud rate** (see page 54)

JX2 system bus cable specification

JX2 system bus cable specification

Parameter	Description
Core cross-sectional area	1,000 kBaud: 0.25 … 0.34 mm^2 500 kBaud: 0.34 … 0.50 mm^2 250 kBaud: 0.34 … 0.60 mm^2 125 kBaud: 0.50 … 0.60 mm^2
Cable capacitance	60 pF/m max.
Resistivity	1,000 kBaud: 70 Ω/km max. 500 kBaud: 60 Ω/km max. 250 kBaud: 60 Ω/km max. 125 kBaud: 60 Ω/km max.
Number of cores	5
Shielding	Complete shielding, no paired shielding
Twisting	Core pairs CL and CH are twisted

Connection diagram



Number	Part	Description
1	Male Sub-D connector, 9-pin	For connection to BUS-OUT
2	Female Sub-D connector, 9-pin	For connection to BUS-IN
3	Not connected	Do not connect these pins

Male Sub-D connector

Pinout of the 9-pin male Sub connector at the JX2 system bus cable:

Pin	Signal name	Description
1	CMODE0	Commissioning
2	CL	Data signal
3	GND	Reference potential
4	CMODE1	Commissioning
5	TERM	Short-circuited with pin 3
7	CH	Data signal

Female Sub-D connector

Pinout of the 9-pin female Sub-D connector to the JX2 system bus cable:

Pin	Signal name	Description
1	CMODE0	Commissioning
2	CL	Data signal
3	GND	Reference potential
4	CMODE1	Commissioning
7	CH	Data signal

Line length and baud rate of the JX2 system bus

Cable lengths

The maximum cable length depends on the baud rate used and the number of expansion modules connected to the bus.

Baud rate	Cable length	Stub length	Total stub length
1,000 kBaud	25 m max.	0.3 m max.	3 m
500 kBaud	100 m max.	1.0 m max.	39 m
250 kBaud	200 m max.	3.0 m max.	78 m
125 kBaud	200 m max.	-	-

Rules for calculating the stub length

When engineering the line length, follow the rules listed below:

- Each non-intelligent JX2-I/O module connected to the system bus reduces the maximum line length by 1.0 m
- Each connected intelligent JX2-I/O slave module reduces the maximum line length by 1.0 m
- Each JetMove reduces the maximum line length by 1.0 m
- Each connected IP67-I/O module reduces the maximum line length by 1.0 m

Baud rate

The baud rate setting depends on the number of modules connected to the JX2 system bus:

JX2-I/O modules JX2 slave modules JetMove	JX-SIO CANopen® modules	1,000 kBaud	500 kBaud	250 kBaud	125 kBaud
x		x	x	x	x
	x	x	x	x	x
x	x	x			x

4.3 Local JX6-I/O submodules connected via PCI bus

Introduction

This chapter describes the local JX6-I/O submodules of the controller JC-940MC connected via PCI bus.

Possible number of local JX6-I/O submodules

At the PCI bus, up to two local JX6-I/O submodules can be connected. The controller JC-940MC is available either without or with one module board. Into each module board, two local JX6-I/O submodules can be plugged.

The module board takes on converting the PC-internal PCI bus to the local JX6-I/O submodule.

Pluggable modules

The following JX6-I/O submodules can be plugged on the PCI module board of the controller JC-940MC:

- JX6-SV1: Combined module (counter, analog output, relay output)
- JX6-IO16CB: Digital inputs/outputs 24 V
- JX6-SB/JX6-SB-I (see chapter *JX2 system bus interface - communicating via PCI bus*)

Expandability

Onto the PCI module board, one or two JX6-I/O submodules can be plugged. The modules specified in the list can be combined at will.

Contents

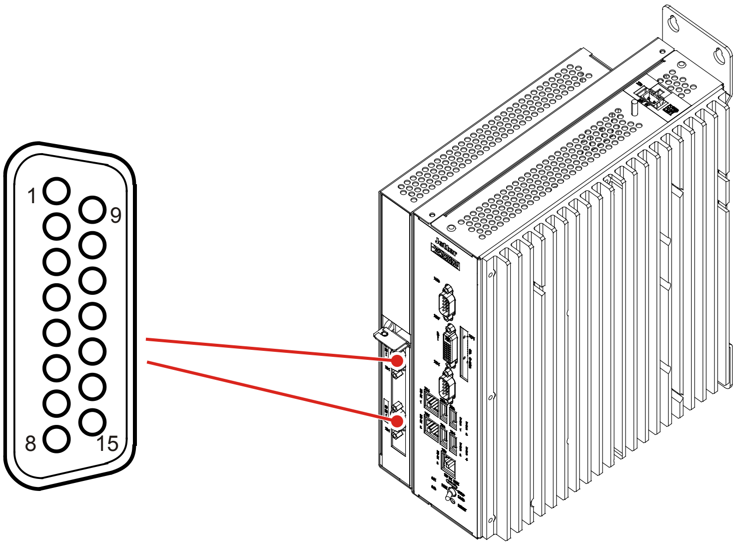
Pinout of the JX6-SV1

Description of the JX6-SV1

The Sub-D connector of the I/O module JX6-SV1 is the interface to various I/O functions. The following interfaces are available on the Sub-D connector:

- 1 galvanically separated relay contact (NOC)
- 1 analog voltage output (-10 V ... +10 V)
- 1 incremental encoder input 5 V differential or
- 1 incremental encoder input 24 V or
- 1 absolute encoder input SSI

Sub-D connector - Pinout



Pin	Signal Incremental encoder 5 V differential incremental encoder	Signal Incremental encoder 24 V	Signal Absolute encoder SSI
1	GND	GND	GND
2	K0+	K0	Unassigned
3	K0-	Unassigned	Unassigned
4	K1+	K1	Data+
5	K1-	Unassigned	Data-
6	K2+	K2	Unassigned
7	K2-	Unassigned	Unassigned
8	Unassigned	Unassigned	Clock-
9	Unassigned	Unassigned	Clock+
10	DC 5 V (50 mA)	Unassigned	Unassigned
11	Interrupt input	Interrupt input	Interrupt input
12	Unassigned	Unassigned	Unassigned

Pin	Signal Incremental encoder 5 V differential incremental encoder	Signal Incremental encoder 24 V	Signal Absolute encoder SSI
13	Analog output	Analog output	Analog output
14	Relay contact 1	Relay contact 1	Relay contact 1
15	Relay contact 2	Relay contact 2	Relay contact 2

Technical specifications

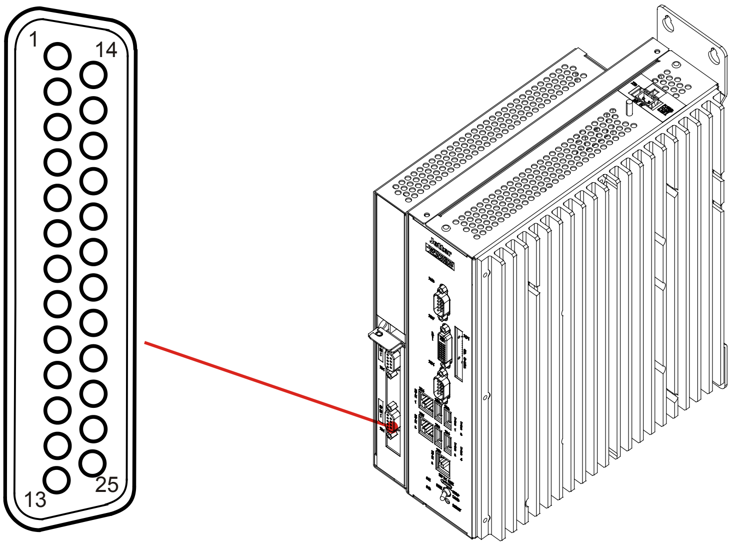
Parameter	Description
Type of terminal	Sub-D connector
Number of pins	15
Electrical isolation	Only at relay contact

Pinout of the JX6-IO16CB

Description of the JX6-IO16CB

The Sub-D connector of the I/O module JX6-IO16CB is the interface to digital inputs and outputs.

Sub-D connector - Pinout



Pin	Signal	Description
1	Output 8	
2	Output 7	
3	DC 24 V	Power supply for the outputs
4	Output 6	
5	Output 5	
6	GND_A	Reference potential of the outputs
7	DC 24 V	Power supply for the outputs
8	Output 4	
9	Output 3	
10	GND_A	Reference potential of the outputs
11	DC 24 V	Power supply for the outputs
12	Output 2	
13	Output 1	
14	Input 8	
15	GND_E	Reference potential of the inputs
16	Input 7	
17	Input 6	
18	Input 5	
19	DC 24 V	Power supply of the outputs

Pin	Signal	Description
20	Input 4	
21	Input 3	
22	Input 2	
23	Input 1	
24	GND_A	Reference potential of the outputs
25	Not assigned	

Technical specifications

Parameter	Description
Type of terminal	Sub-D connector
Number of pins	25
Electrical isolation	Yes
Rated voltage	DC 24 V
Number of inputs	8
Number of outputs	8
Peak current per output	0.5 A

4.4 Indicators and LEDs

LEDs of the controller JC-940MC

The controller JC-940MC features the following LEDs:

- Three LEDs for indicating conditions and errors of the controller
- Two LEDs for indicating the conditions of the Ethernet ports

Contents

LEDs of the controller

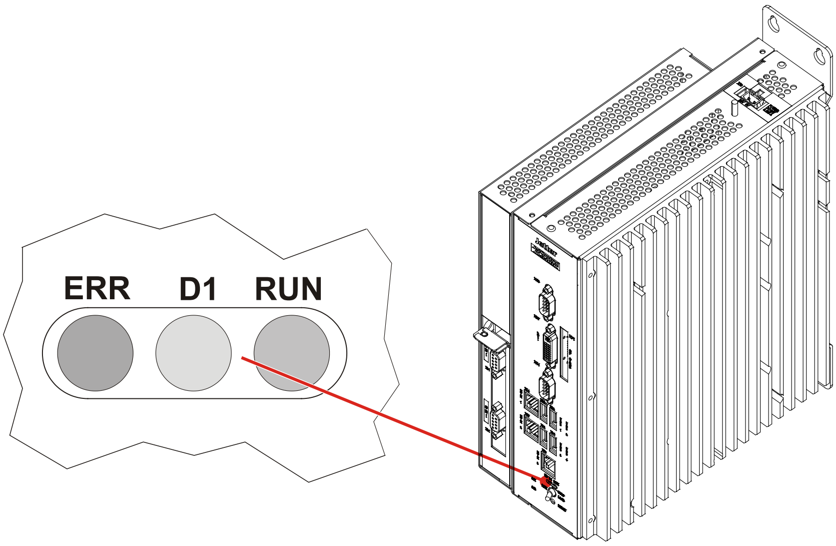
LEDs of the controller

The controller JC-940MC indicates conditions and errors via its LEDs. The LEDs are split into 2 groups:

- State messages of the operating system
- State of the Ethernet port

State LEDs

The state LEDs of the OS are located in the diagnostic and state area at the lower area of the front panel.



OS LED	Color	Description
RUN	green	OS is running
D1	amber	Special conditions
ERR	red	Error

Normal operating condition

In normal operating condition, the OS LEDs of the controller JC-940MC indicate the following:

ERR	D1	RUN	State
OFF	OFF	ON	Normal operating condition ■ Application program is running ■ No error

States of the OS LEDs

The table below shows the possible states of the OS LEDs RUN, ERR, and D1:

LED	State	Description
RUN	OFF	No power supply or failure
	1Hz	Application program is not being executed
	ON	Application program is being executed
ERR	OFF	No error
	ON	Error; refer to error register
D1	OFF	Normal operating condition
	ON	Special conditions

LEDs of the controller during boot process

Normal boot process

If the following requirements are met, the controller goes through its normal boot process:

- Mode selector S11 is in *RUN* position.
- There is a valid OS.
- There is a valid application program.

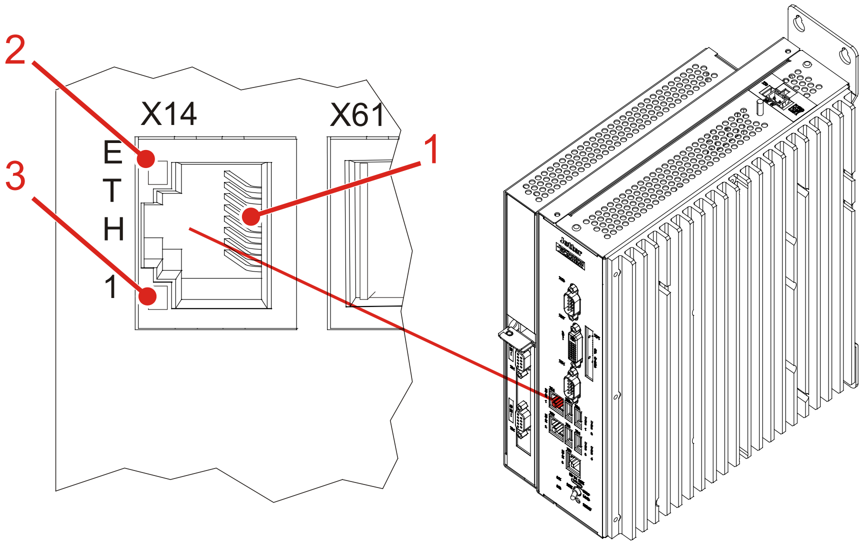
During boot process of the controller, the OS status LEDs then indicate the following:

Step	Description			
1	**ERR**	**D1**	**RUN**	**State**
	ON	OFF	OFF	Reset
2	**ERR**	**D1**	**RUN**	**State**
	OFF	ON	OFF	Initializing the runtime environment of the application program and real-time communication
3	**ERR**	**D1**	**RUN**	**State**
	ON	ON	OFF	Motion control start
4	**ERR**	**D1**	**RUN**	**State**
	ON	ON	ON	Initializing additional functions (Web, Modbus/TCP etc.)
5	**ERR**	**D1**	**RUN**	**State**
	OFF	OFF	ON	Normal condition; the application program is being executed

Status LEDs - Ethernet interface

Status LEDs - Ethernet interface

The state LEDs of the Ethernet interface are located in the immediate vicinity of the RJ45 jack.

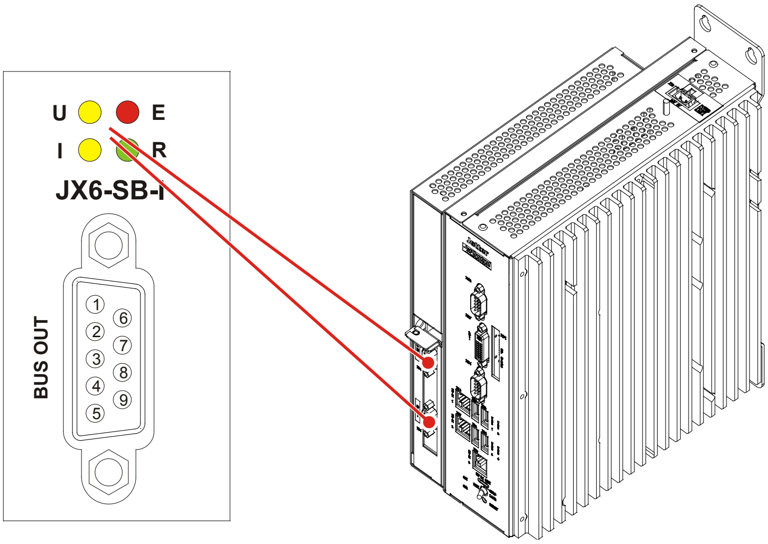


Number	LED color	Description
2	green	LINK: Network connection of 100 MBit/s has been established
	red	LINK: Network connection of 1,000 MBit/s has been established
3	green	ACT: Data transmission

LEDs of the submodule JX6-SB(-I)

LEDs of the submodule JX6-SB(-I)

The submodule JX6-SB(-I) indicates conditions and errors via its LEDs. The LEDs are at the front panel of the JX6-SB(-I).



States after power-up

Immediately after switching on, the LEDs of the submodule JX6-SB(-I) have got the following states:

U	I	E	R	State
OFF	OFF	OFF	1Hz	LED **R** is flashing slowly. The JX6-SB submodule is ready for initialization commands.
OFF	OFF	OFF	2x	LED **R** flashes twice, followed by a long interval. The JX6-SB-I submodule is ready for initialization commands.
OFF	4Hz	4Hz	4Hz	No valid operating system on the JX6-SB(-I) submodule available.

States in master-slave mode

In the master-slave mode, the state-LEDs can enter the following states:

LED	State	Description
R	ON	Successful initializing of the JX2 system bus by means of command 30.
E	ON	Error; see state registers of the JX6-SB(-I) submodule
I	ON	The JX2 system bus is initialized.
U		The controller cyclically reads the input states of all modules connected to the JX2 system bus. After each reading the LED changes its state.

Normal operating condition

In normal operating condition, the LEDs of the submodule JX6-SB(-I) have got the following states:

U	I	E	R	State
	OFF	OFF	ON	Normal operating condition ■ The JX2 system bus has been initialized in the master-slave operating mode. ■ There is no error. ■ The controller cyclically reads the input states of all modules connected to the JX2 system bus.

4.5 Control elements

Control elements of JC-940MC

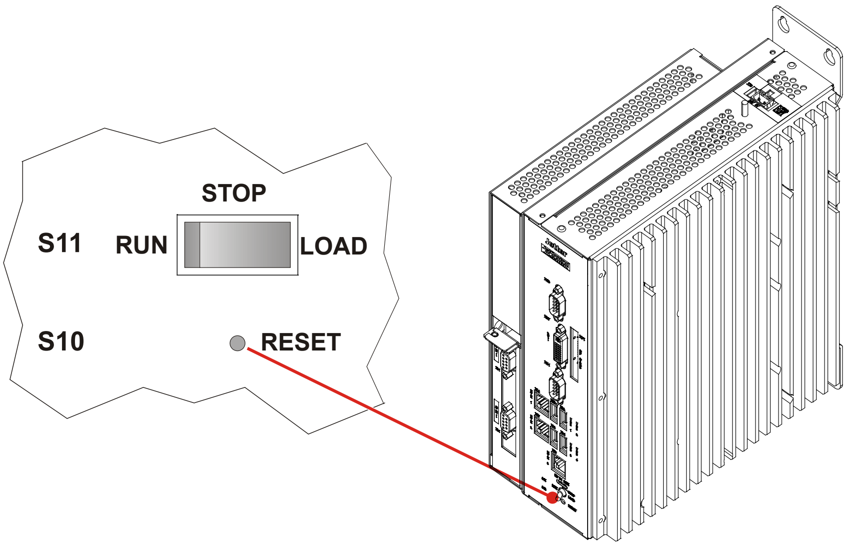
The JC-940MC controller features the following control elements:

- Pushbutton S10 for triggering a reset
- Mode selector S11 with the following positions: RUN, STOP, and LOAD

Contents

Pushbutton S10

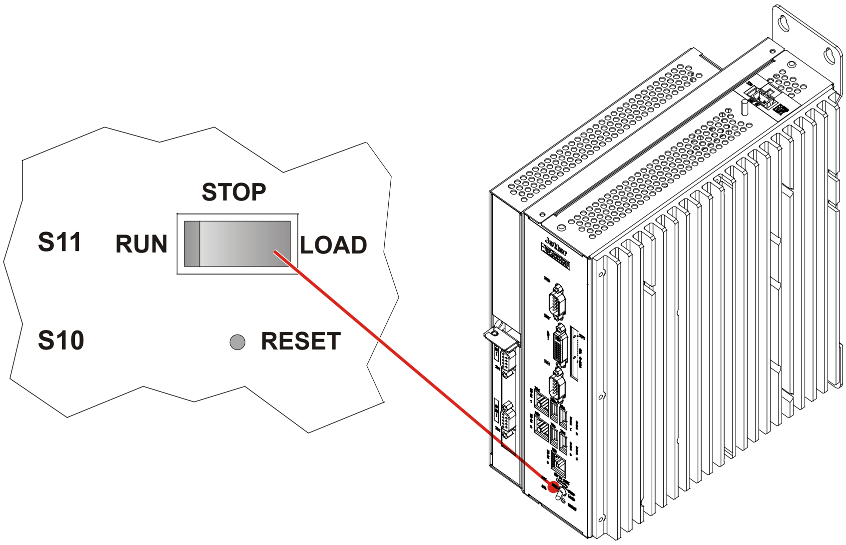
Pushbutton S10



Function of the pushbutton	By pressing this pushbutton a reset of the controller is triggered.
Attention! Loss of data!	By pressing this pushbutton, all changes made to non-volatile registers and flags get lost. All changes made after the latest turning on will be lost.

Mode selector S11

Mode selector S11



Position	Description
RUN	Once the controller JC-940MC is turned on, it launches the application program.
STOP	Once the controller JC-940MC is turned on, the application program will not be launched.
LOAD	Once the controller JC-940MC is turned on, the application program will not be launched.

Functions of the mode selector

The controller JC-940MC checks the position of selector S11 in the following way:

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">Power supply of the controller is at connector X10.</td></tr>
<tr><td rowspan="4">2</td><td colspan="2">The boot loader of the controller checks the position of selector S11.</td></tr>
<tr><th>If ...</th><th>... then ...</th></tr>
<tr><td>... mode selector S11 = RUN or STOP,</td><td>... the OS is launched;
--> proceed with step 3</td></tr>
<tr><td>... mode selector S11 = LOAD,</td><td>... the controller starts the AutoCopy function.</td></tr>
<tr><td rowspan="4">3</td><td colspan="2">The controller checks the position of selector S11.</td></tr>
<tr><th>If ...</th><th>... then ...</th></tr>
<tr><td>... mode selector S11 = RUN,</td><td>... the application program is launched.</td></tr>
<tr><td>... mode selector S11 = STOP,</td><td>... the application program does not start.</td></tr>
<tr><td rowspan="2">4</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>... the position of mode selector S11 is changed once the controller has been turned on,</td><td>... this has no effect on the functioning of the controller.</td></tr>
</table>

4.6 Installing and removing the controller JC-940MC

Introduction

This chapter describes how to install and remove the controller JC-940MC.

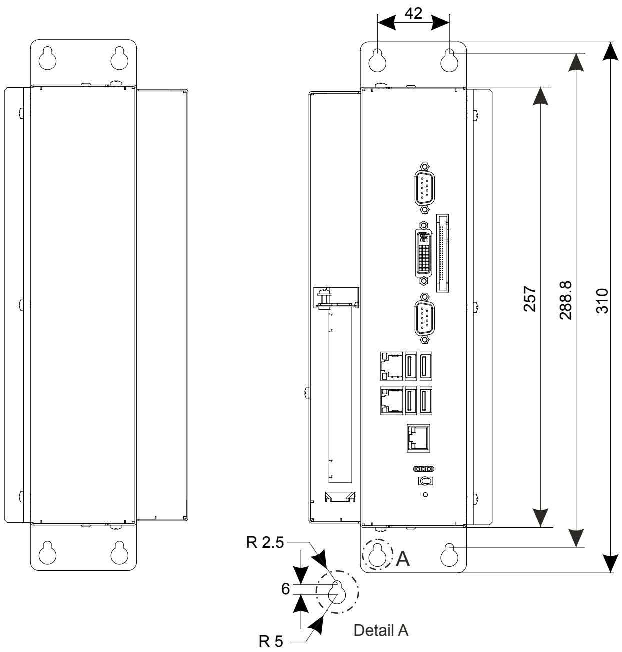
Contents

Installing the controller JC-940MC

Installing

To install the controller JC-940MC, proceed as follows:

Step	Action
1	Mount the controller JC-940MC in vertical direction onto the panel of the control cabinet, as is shown in the illustration below.
2	Make sure that the clearance above and below the controller is 100 mm respectively. On the side of the heat sink, a clearance of at least 50 mm must be kept.
3	Mark on the panel two positions for the fastening screw threads of controller JC-940MC. The dimensions can be taken from the illustration below.
4	Drill the holes and cut the respective threads into the panel.
5	Screw the corresponding fitting bolts into the thread by half of their length.
6	By means of the oblong holes in the rear plate, hang up the controller JC-940MC by the fitting bolts. Then screw them down tightly.



Related topics

- **Removing the controller JC-940MC** (see page 73)

Removing the controller JC-940MC

Removal

To remove the controller JC-940MC, proceed as follows:

Step	Action
1	Remove power from the controller JC-940MC.
2	Unscrew the four fitting bolts by half of their length.
3	Remove the controller JC-940MC from the fitting bolts.

Related topics

- **Installing the controller JC-940MC** (see page 72)

4.7 Battery replacement

Introduction

This chapter describes the technical key data to be considered when replacing batteries on a controller by Jetter AG.

Liability

Jetter AG recommend to return the controller always to Jetter AG to have the batteries replaced. The user is responsible for carrying out a complete data backup (application programs and user registers).

Jetter AG assume no liability for any battery replacement which has not been carried out by Jetter AG.

Notes

- Observe the rules for handling ESD-sensitive components while replacing the battery.
- Polarity reversal of the battery will damage the controller
- The rated voltage of the batteries is 3.0 V +/- 10 %.
- The batteries are integrated, residing in an intermediate socket.

Contents

Removing the battery of the controller JC-940MC

Real-time clock registers

The battery buffers the real-time clock. The lifetime is 10 years. For this reason, you have to remove the battery before this time has expired.

Register areas to be backed up:	None
Real-time clock/BIOS:	Real-time clock, battery-backed
After battery removal:	The real-time clock has to be reset

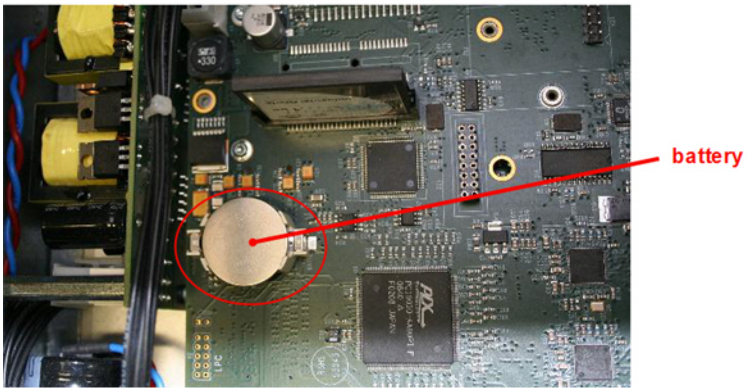
Battery type

The battery type is a B_CR2477N lithium button cell. You can order the battery from Jetter AG under the item number 60876304.

The battery is placed in a socket.

Battery position

The illustration below shows the battery position in the JC-940MC controller.



Battery state

The battery state cannot be checked via software.

4.8 IP configuration

Introduction

This chapter describes the IP configuration for the controller JC-940MC. The following parameters can be set:

- IP address of the controller
- Subnet mask
- IP address of default gateway
- Static route of the IP address
- Static route of the subnet mask
- Static route of the gateway
- IP address of DNS server
- Controller name
- IP port number for the JetSym debugger
- Basic IP port number for communication via JetIP

Required skills of the network configurator

To carry out IP configuration of the controller JC-940MC knowledge of IP networks is required, such as

- IP addressing (IP address, port number, subnet masks etc.)
- FTP (connection setup, data transmission, etc.)

Jetter Ethernet system bus

The Jetter Ethernet system bus (JetIPScan, JetSync, Publish/Subscribe, etc.) and thus synchronous transfer of data packets will further be supported by port ETH 1 (X14) only.

Contents

Factory settings

Introduction

Before the controller JC-940MC is shipped, various parameters are set to a certain value.

The parameters can be changed by the user.

Factory settings

Parameter	Value
ETH 1: IP address	192.168.1.1
ETH 1: Subnet mask	255.255.255.0
ETH 1: IP address of default gateway	0.0.0.0
IP address of DNS server	0.0.0.0
Controller name	JetControl940MC
IP port number for debugger	52000
IP port number for JetIP	50000
Password of the user *admin*	admin
Password of the user *system*	system

Determining the IP address of the controller JC-940MC

Introduction

The program JetIPScan determines the IP address, subnet mask and the IP address of the default gateway of the controller JC-940MC. Yet, this only applies to the Ethernet port ETH 1.

Download of the program JetIPScan

Jetter AG provide the program JetIPScan on their **homepage http://www.jetter.de**. You can find the file **jetipscan_1-08-01.zip** via *Industrie Automation - Support - Downloads - 06_software - 30_sonstiges - jetipscan*.

Contents of the jetipscan_1-08-01.zip

The zip file **jetipscan_1-08-01.zip** contains the following files:

- The program **JetIPScanV1.08_01.exe**
- The help **jetipscan_01_help_en.png**
- A batch file **read_IP_via_JETIPSCAN.bat** to determine the IP address
- A batch file **write_IP_via_JETIPSCAN_10_150.bat** to set IP address 192.168.10.150 for the controller

The batch files start the program JetIPScan

The files are unzipped to the folder **jetipscan_1-08-01**

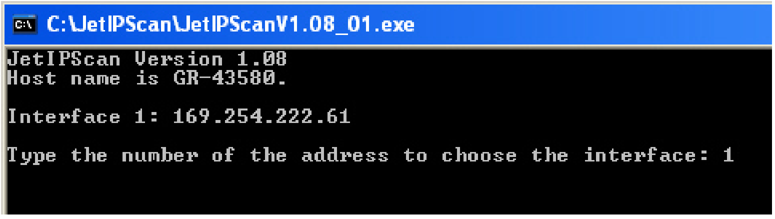
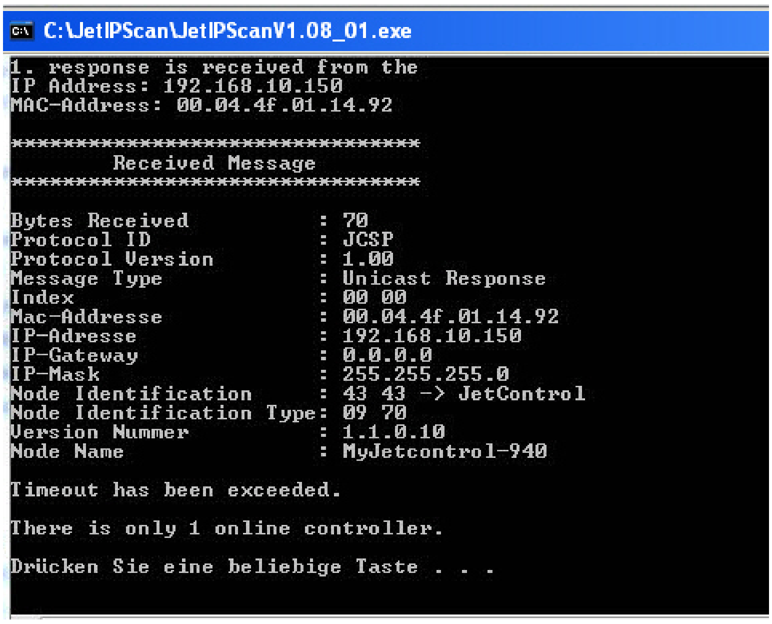
Prerequisites

First, an Ethernet connection between the PC and the controller JC-940MC must be established.

Then launch the JetIPScan program on your PC, for example via the batch file **read_IP_via_JETIPSCAN.bat**.

Determining the IP address

To determine the IP address of the controller JC-940MC, proceed as follows:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Launch the JetIPScan program on your PC.</td></tr>
<tr><td>⇨</td><td>Result: JetIPScan shows all IP addresses, which are presently active on your PC.</td></tr>
<tr><td>2</td><td>Select the interface (IP address) connected with the controller JC-940MC, of which you want to determine the IP address.</td></tr>
<tr><td></td><td>
</td></tr>
<tr><td>⇨</td><td>Result: Among others, JetIPScan shows the IP address of the controller JC-940MC.</td></tr>
<tr><td></td><td>
</td></tr>
</table>

The configuration memory

Introduction

The controller reads the parameters for initializing the IP interface out of the configuration memory during the boot process. The user can access the data stored in the configuration memory in the following ways:

- Registers let you read out and change the application data.

Enabling conditions

The controller reads out data located in the configuration memory only during the boot process. If you make changes to the configuration memory, reboot the controller for these changes to take effect. Only this way these changes take effect.

Default values

Before the controller further processes data from the configuration memory, it checks them for plausibility. If entries are invalid or absent, the controller uses the following default values:

Parameter	Default value
ETH 1: IP address	192.168.10.150
ETH 1: Subnet mask	255.255.255.0
ETH 1: IP address of default gateway	0.0.0.0
ETH 2: IP address	0.0.0.0
ETH 2: Subnet mask	0.0.0.0
ETH 2: IP address of default gateway	0.0.0.0
ETH 3: IP address	0.0.0.0
ETH 3: Subnet mask	0.0.0.0
ETH 3: IP address of default gateway	0.0.0.0
IP address of DNS server	0.0.0.0
ETH 1: Static route of the IP address	0.0.0.0
ETH 1: Static route of the subnet mask	0.0.0.0
ETH 1: Static route of the gateway	0.0.0.0
ETH 2: Static route of the IP address	0.0.0.0
ETH 2: Static route of the subnet mask	0.0.0.0
ETH 2: Static route of the gateway	0.0.0.0
ETH 3: Static route of the IP address	0.0.0.0
ETH 3: Static route of the subnet mask	0.0.0.0
ETH 3: Static route of the gateway	0.0.0.0
Controller name	JetControl940MC
Suffix type of the name	0
IP port number for debugger	52000
IP port number for JetIP	50000

Related topics

- **Configuration file** *config.ini* (see page 82)
- **Configuration registers** (see page 86)

Configuration file "config.ini"

Introduction

If, for initialization of the IP interface, the data in the configuration memory are not valid, the controller JC-940MC uses the settings in the configuration file **config.ini**. Yet, this only applies to the Ethernet port ETH 1.

Properties

- You can access this file through the file system of the controller.
- For an FTP connection, the user must have administrator or system rights.
- This file is located in the folder **System**.
- You cannot delete the file, but only overwrite it.
- Formatting the Flash disk drive leaves the file unchanged.

File structure

The configuration file is a text file the entries of which are grouped into several sections. The controller replaces missing IP configuration parameters by their default values.

Example: *config.ini*

This is an example of a configuration file **config.ini**:

```
;<Productname> System Configuration
;Copyright (c) 2009 by Jetter AG, Ludwigsburg, Germany

[IP]
Address    = 192.168. 50.  1
SubnetMask = 255.255.255.  0
DefGateway = 192.168. 50. 11
DNSServer  = 192.168.  1. 44

[HOSTNAME]
SuffixType = 0
Name       = JetControl940MC

[PORTS]
JetIPBase = 50000
JVMDebug  = 52000

[FILES]
AutoCopyIni = /USB1/autocopy.ini
```

Section [IP]

In section [IP] the required IP addresses and the subnet mask are specified.

Address	
In the given example	192.168.50.1
Description	IP address of the controller
Allowed values	■ > 1.0.0.0 ■ < 223.255.255.255
Illegal values	■ Network address ■ Broadcast address
In the event of an illegal value	The controller resets all four values to their default values.
SubnetMask	
In the given example	255.255.255.0
Description	Sets the subnet mask
Allowed values	■ >= 128.0.0.0
Illegal values	■ 1 and 0 mixed
In the event of an illegal value	The controller resets all four values to their default values.
DefGateWay	
In the given example	192.168.50.11
Description	IP address of the gateway to other subnets; The controller must be able to reach the subnet (Address/SubnetMask), otherwise it will set this parameter to 0.0.0.0.
Allowed values	■ >= 0.0.0.0 and ■ < 223.255.255.255
Illegal values	■ Network address ■ Broadcast address ■ A value (Address/SubnetMask) which cannot be reached by the controller. ■ The *Address* value
In the event of an illegal value	The controller sets the value to 0.0.0.0
DNSServer	
In the given example	192.168.1.44
Description	IP address of the server for the Domain Name System
Allowed values	■ >= 0.0.0.0 and ■ < 223.255.255.255
In the event of an illegal value	The controller sets the value to 0.0.0.0

Section [HOSTNAME]

In section [HOSTNAME] the name of the controller is specified. If desired, the controller automatically generates an individual name. The controller JC-940MC presently does not use the host name.

SuffixType	
In the given example	0
Description	The type of the automatically generated suffix is attached to the controller name
Allowed values	■ 0: No attachment ■ 1: Low-order byte of the IP address in decimal notation ■ 2: Low-order byte of the IP address in hexadecimal notation
In the event of an illegal value	0
Name	
In the given example	JetControl940MC
Description	Specifies the controller name
Allowed values	■ First character: 'A' ... 'Z', 'a' ... 'z' ■ Next characters: 'A' ... 'Z', 'a' ... 'z', '0' ... '9', '-'
In the event of an illegal value	JetControl940MC

Section [PORTS]

In section [PORTS] the IP port numbers of data and debug servers within the controller are specified. The IP port numbers must be consistent with, for example, the port numbers set in JetSym.

JetIPBase	
In the given example	50000
Description	IP port number for OS update and communication between controllers
Allowed values	■ 1024 ... 65535
In the event of an illegal value	50000
JVMDebug	
In the given example	52000
Description	IP port number for debugger/setup in JetSym
Allowed values	■ 1024 ... 65535
In the event of an illegal value	52000

Changing the IP configuration

Step	Action
1	Create on your PC a configuration file named *config.ini* using a text editor and make the corresponding entries.
2	Open an FTP connection between the PC and the JC-940MC controller.
3	Log in as user with administrator or system rights. Default login information: User: *admin*, Password: *admin*
4	Browse to directory */System* of the controller JC-940MC.
5	Copy the configuration file named **config.ini**, which has been created by you, to the controller.
6	Close the FTP connection
7	Reboot the controller. **Result**: If the data in the configuration memory are invalid, the new configuration is active.

Related topics

- **The configuration memory** (see page 80)
- **Configuration registers** (see page 86)

Configuration registers

Introduction

The parameters of the IP configuration can be read and modified via the configuration registers. An array of registers holds the data contained in the file **\System\config.ini**. Another array contains the parameters used for initializing the IP interface.

Register numbers

The basic register numbers of both arrays are dependent on the device. The register number is calculated by adding the number of the module register (MR) to the number of the basic register.

Controller	Data range	Basic register number	Register numbers
JC-940MC	config.ini	101100	101100 ... 101165
	Parameters used	101200	101200 ... 101265

Configuration registers

The following table lists the registers of both arrays, as well as their connection to the entries in the configuration file **\System\config.ini**:

Registers	Section in config.ini	Name in config.ini	Description
MR 0	IP	`Address`	IP address of the controller
MR 1		`SubnetMask`	Sets the subnet mask
MR 2		`DefGateWay`	IP address of the gateway to other subnets
MR 3		`DNSServer`	IP address of the server for the Domain Name System
MR 32	HOSTNAME	`SuffixType`	The type of the automatically generated suffix is attached to the controller name
MR 33 through 51		`Name`	Specifies the controller name
MR 64	PORTS	`JetIPBase`	IP port number for OS update and communication between controllers
MR 65		`JVMDebug`	IP port number for debugger/setup in JetSym

Related topics

- **The configuration memory** (see page 80)
- **Configuration file** *config.ini* (see page 82)

Changing the IP address of the controller

Introduction

To be able to communicate with the controller JC-940MC via Ethernet, you must set an unambiguous IP address on the controller.

Configuration options

You can configure the IP address of the port ETH 1 in the following ways:

- Configuration via file **config.ini**
- Configuration during runtime via special registers (not remanent)
- Configuration via special registers (remanent)
- Configuration via the program JetipScan (remanent)

You can configure the IP addresses ports ETH 2 and ETH 3 in the following ways:

- Configuration during runtime via special registers (not remanent)
- Configuration via special registers (remanent)

Related topics

- **Setting the IP address via** *config.ini* **file** (see page 82)
- **Setting the IP address during runtime** (see page 95)

Changing the IP address of the controller JC-940MC via JetIPScan

Introduction

The program JetIPScan changes the IP address, subnet mask and the IP address of the default gateway of the controller JC-940MC. Yet, this only applies to the Ethernet port ETH 1.

Download of the program JetIPScan

Jetter AG provide the program JetIPScan on their **homepage http://www.jetter.de**. You can find the file **jetipscan_1-08-01.zip** via *Industrie Automation - Support - Downloads - 06_software - 30_sonstiges - jetipscan*.

Contents of the jetipscan_1-08-01.zip

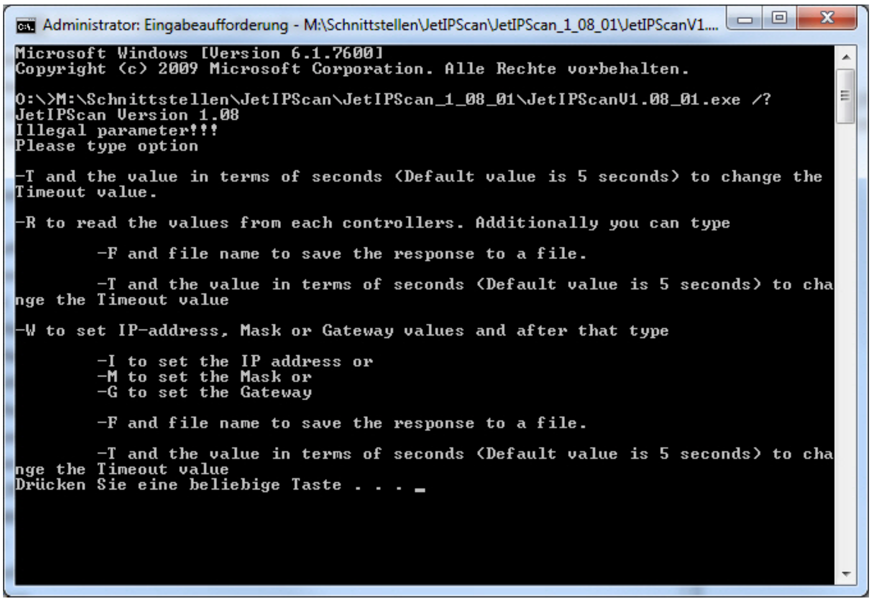
The zip file **jetipscan_1-08-01.zip** contains the following files:

- The program **JetIPScanV1.08_01.exe**
- The help **jetipscan_01_help_en.png**
- A batch file **read_IP_via_JETIPSCAN.bat** to determine the IP address
- A batch file **write_IP_via_JETIPSCAN_10_150.bat** to set IP address 192.168.10.150 for the controller

The batch files start the program JetIPScan

The files are unzipped to the folder **jetipscan_1-08-01**

Possible commands of the software JetIPScan

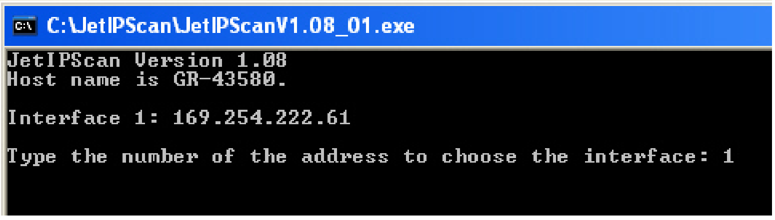


Prerequisites

First, an Ethernet connection between the PC and the controller JC-940MC must be established.

Changing the IP address

To change the IP address of the controller JC-940MC, proceed as follows:

Step	Action
1	Write a batch file. The content of the batch file is **JetIPScanV1.08_01.exe -W -I 192.168.10.150**.
2	Execute the batch file.
⇨	**Result:** JetIPScan is launched and shows all IP addresses, which are presently active on your PC.
3	Select the interface (IP address) connected with the controller JC-940MC, of which you want to determine the IP address.
	
4	JetIPScan shows all the devices found. To change the IP address of a device, select the respective device from the list.
⇨	**Result:** JetIPScan changes the IP address of the controller JC-940MC to the value 192.168.10.150.

Changing the subnet mask

To change the subnet mask of the controller JC-940MC, proceed as follows:

Step	Action
1	Write a batch file. The content of the batch file is, for example **JetIPScanV1.08_01.exe -W -M 250.255.248.0**.
2	Execute the batch file.
⇨	**Result:** JetIPScan is launched and shows all interfaces, which are presently active on your PC.
3	For all further steps, please refer to the instruction **Changing the IP address**.

Changing the default gateway

To change the default gateway of the controller JC-940MC, proceed as follows:

Step	Action
1	Write a batch file. The content of the batch file is, for example, **JetIPScanV1.08_01.exe -W -G 192.168.4.1**.
2	Execute the batch file.
⇨	**Result:** JetIPScan is launched and shows all interfaces, which are presently active on your PC.
3	For all further steps, please refer to the instruction **Changing the IP address**.

Setting the IP address via "config.ini" file

Configuration file *config.ini*

You can set the IP address of the controller JC-940MC in the **config.ini** file.

```
[IP]
Address     = aaa.bbb.ccc.ddd
...
```

Element	Description
`Address`	Enter the IP address into this line
`aaa`	First byte of IP address
`bbb`	Second byte of IP address
`ccc`	Third byte of IP address
`ddd`	Fourth byte of IP address

Note

If the data in the configuration memory are not o.k., the controller uses the IP address settings in the configuration file config.ini. Yet, this only applies to the Ethernet port ETH 1.

Transferring the *config.ini* file

Step	Action
1	Establish an FTP connection to the JC-940MC controller.
2	Log in as user with administrator or system rights. Default login information: User: *admin*; Password: *admin* User: *system*; Password: *system*
3	Open the folder **System**.
4	Copy the file **config.ini** into the folder **System**.
5	Clear the FTP connection.
6	Reboot the controller.

Setting the IP address via registers to be remanent

Introduction

The IP interfaces ETH 1 to ETH 3 are initialized during the boot process by the settings in the configuration memory.

The following remanent settings can also be changed via registers:

- IP address of the controller
- Subnet mask
- IP address of default gateway
- Static route of the IP address
- Static route of the subnet mask
- Static route of the gateway
- IP address of DNS server
- Host name and suffix type
- Port numbers for JetIP and the JetSym debugger

Register overview

Register	Description
101200	ETH 1: IP address
101201	ETH 1: Subnet mask
101202	ETH 1: IP address of default gateway
101203	IP address of DNS server
101210	ETH 2: IP address
101211	ETH 2: Subnet mask
101212	ETH 2: IP address of default gateway
101213	ETH 3: IP address
101214	ETH 3: Subnet mask
101215	ETH 3: IP address of default gateway
101216	ETH 1: Static route of the IP address
101217	ETH 1: Static route of the subnet mask
101218	ETH 1: Static route of the gateway
101219	ETH 2: Static route of the IP address
101220	ETH 2: Static route of the subnet mask
101221	ETH 2: Static route of the gateway
101222	ETH 3: Static route of the IP address
101223	ETH 3: Static route of the subnet mask
101224	ETH 3: Static route of the gateway
101232	Host name suffix type
101233 through 101251	Host name

Registers	Description
101264	Port number for JetIP
101265	Port number for STX debugger
101299	Saving the settings (0x77566152)

Setting IP addresses and subnet mask

For changing of the IP addresses, the subnet mask, and the default gateway of the Ethernet port ETH 1 to be remanent, proceed as follows:

Step	Action
1	Enter the desired IP address of port ETH 1 into register 101200.
2	Enter the desired subnet mask into register 101201.
3	Enter the desired IP address of the default gateway into register 101202.
4	To have the controller take over the values, you must enter a password. For this, write value 2002149714 (0x77566152) to register 101299.
5	Boot the controller JC-940MC.

Result: The settings are completed. Communication is possible again.

To set the IP address, the subnet mask, and the default gateway of the Ethernet ports ETH 2 and ETH 3, proceed accordingly.

Setting the default gateway

Only one default gateway can be set at a time. If in the configuration memory of the configuration several default gateways are set, the controller uses the first default gateway that having got a value unequal 0.0.0.0 which it finds. The controller proceeds in the sequence of the ports.

To set the default gateway on the Ethernet port ETH 2, proceed as follows:

Step	Action
1	Enter the value 0.0.0.0 into register 101202.
2	Enter the desired IP address of the default gateway into register 101212.
3	To have the controller take over the values, enter the password that is needed. For this, write value 2002149714 (0x77566152) to register 101299.
4	Boot the controller JC-940MC.

Result:

The default gateway of port ETH 2 is set. The value set for the default gateway of port ETH 3 does not relate to the controller any more.

Related topics

- **The configuration memory** (see page 80)
- **Setting the IP address during runtime** (see page 95)
- **Setting a static route** (see page 99)

Setting the IP address automatically via the USB flash drive

Introduction

To have the IP configuration of the controller automatically set via a USB flash drive, you can apply the function *Copying controller data automatically (AutoCopy)*. For this, use the registers described in the chapter *Setting the remanent IP address via registers*.

Prerequisites

You are familiar with the AutoCopy function

AutoCopy command file

The example below shows a command file of the AutoCopy function:

```
[OPTIONS]
CommandCount = 1
LogFile = /USB1/autocopy.log
LogAppend = 0

# set registers for IP configuration
[COMMAND_1]
Command = DaFileRead
DaFile = /USB1/ip-setup.da
```

Data file *ip-setup.da*

This example shows a data file, by which the IP address, the subnet mask and the port number for the JetIP protocol is set in the configuration memory:

```
SD101
; Data File - Jetter AG
;
; Register 101200: ip address
RS    101200    -1062729066
; Register 101201; subnet mask
RS    101201    -512
; Register 101264 JetIP port number
RS    101264    51000
; Register 101299: write to configuration memory
RS    101299    2002149714
```

Related topics

- **The configuration memory** (see page 80)
- **Copying controller data automatically** (see page 371)

Setting the IP address during runtime

Introduction

The IP interface is initialized by the settings in the configuration memory during the boot process.

The following settings can also be changed via registers to be non-remanent:

- IP address of the controller
- Subnet mask
- IP address of default gateway

Important note

The settings made during runtime do not change the parameters in the configuration memory. At de-energizing the controller, your settings will be lost.

Prerequisites

- While settings are being made, there is no communication via IP interface. Otherwise, this would lead to a loss of data.
- The values entered are valid, e.g. by including a validity check in the application program.
  If you set the parameters during runtime of the controller, they will not be checked.

Register overview

Register	Description
104531	ETH 1: IP address
104532	ETH 1: Subnet mask
104533	ETH 1: IP address of default gateway
104540	ETH 2: IP address
104541	ETH 2: Subnet mask
104542	ETH 3: IP address
104543	ETH 3: Subnet mask
104544	ETH 2: IP address of default gateway
104545	ETH 3: IP address of default gateway

Setting IP addresses and subnet mask

For changing the IP addresses and the subnet mask of the Ethernet port ETH 1, proceed as follows:

Step	Action
1	Enter the value 0.0.0.0 into 104533.
2	Enter the value 0.0.0.0 into 104532.
3	Enter the desired IP address of Ethernet port ETH 1 into register 104531.
4	Enter the desired subnet mask into register 104532.
5	Enter the desired IP address of the default gateway into 104533.

Result:

The Ethernet port ETH 1 is set. Communication is possible again. The default gateways of ports ETH 2 und ETH 3 are reset. The reason is that only for one of the three ports an IP address other than 0.0.0.0 is permitted to be defined for a default gateway.

To set the IP address, the subnet mask, and the default gateway of the Ethernet ports ETH 2 and ETH 3, proceed accordingly.

Related topics

- **The configuration memory** (see page 80)
- **Setting the IP address via registers to be remanent** (see page 91)
- **Setting a static route** (see page 99)

Using names for IP addresses

Introduction

Names can be specified as IP addresses for target systems, e.g. when configuring the e-mail client. The controller resolves the names into IP addresses. A configuration file or the Domain Name System is used to assign names to their corresponding IP address.

Name resolution

Names are resolved to IP addresses in the following way:

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">During the boot process the controller reads the IP address of the DNS server out of the configuration memory.</td></tr>
<tr><td>2</td><td colspan="2">During the boot process the controller reads the file /etc/hosts. The controller creates a translation table with the names and IP addresses found in this file.</td></tr>
<tr><td>3</td><td colspan="2">After the boot process the controller detects a name instead of an IP address.</td></tr>
<tr><td rowspan="4">4</td><td colspan="2">Based on this translation table, the controller tries to resolve the name into a related IP address.</td></tr>
<tr><th>If ...</th><th>... then ...</th></tr>
<tr><td>... the controller has resolved the name, ...</td><td>... proceed with step 6.</td></tr>
<tr><td>... the controller has not resolved the name, ...</td><td>... proceed with step 5.</td></tr>
<tr><td rowspan="4">5</td><td colspan="2">The controller tries to resolve the name into a related IP address by sending a request to the DNS server.</td></tr>
<tr><th>If ...</th><th>... then ...</th></tr>
<tr><td>... the controller has resolved the name, ...</td><td>... it enters the name and IP address into the translation table;
--> proceed with step 6.</td></tr>
<tr><td>... the controller has not resolved the name, ...</td><td>... the controller aborts the function, e.g. the system function for sending an e-mail, and issues an error message.</td></tr>
<tr><td>6</td><td colspan="2">The controller uses the IP address resolved for further communication.</td></tr>
</table>

Configuration file *hosts*

This file holds the static assignment between name and IP address. The controller reads this file once during boot-up.

File format:	Text
Location:	/etc
File name:	hosts

Example

```
# Example hosts file for JC-9xx
192.168.33.209    jetter_mail
192.168.33.208    jetter_demo
192.168.1.1       JC940MC
192.168.1.2       JC940MC
```

Domain Name System (DNS)

If a name cannot be found in the file **/etc/hosts**, the controller tries to obtain the corresponding IP address from a DNS server. During boot-up, the controller reads the IP address of the DNS server out of the configuration memory.

Related topics

- **The configuration memory** (see page 80)

Setting a static route

Introduction

The operating system of the JC-940MC offers the possibility to configure simple routing of IP-packets among the three supported Ethernet ports ETH 1 to ETH 3. If the addressed device can be reached this way, the controller then forwards IP packets received at one of the ports to one of the other ports. If networks have to be accessed which are not in the directly accessible range of the JC-940MC, the user must established static routes to these networks. Static routes especially have to be established, if the default gateway cannot reach the respective network.

Saving a route to a configuration memory

For each Ethernet port, entries can be made in the configuration memory. These entries can be changed via registers 101216 through 101224 to be remanent. All entries saved to the configuration memory are activated when the controller is rebooted.

Saving a route at runtime

Another register set (R 104550 through 104555) lets you - during runtime - make further entries into the routing table or delete existing routes. Five entries per port can be made.

R 104550

Status

This register lets you read out the status when adding or deleting routes during runtime.

Register properties		
Reading values	0	No error
	-1	Routing table is full
	-2	Entry not found
	-3	Port is not active
	-4	TCP/IP stack not initialized

R 104551

Command

This command lets you define, whether you want to add or delete a route.

Register properties		
Writing values	1	Add route
	2	Delete route

R 104552

Port number

Use this register to define the Ethernet port.

Register properties		
Writing values	1	ETH 1 (X14)
	2	ETH 2 (X15)
	3	ETH 3 (X16)

R 104553

IP address of the static route

Enter the IP address of the route into this register.

Register properties	
Writing values	0.0.0.0 ... 255.255.255.255

R 104554

Subnet mask of the static route

Enter the number of the subnet mask of the route into this register.

Register properties	
Writing values	0.0.0.0 ... 255.255.255.255

R 104555

Gateway of the static route

Enter the gateway IP address of the route into this register.

Register properties	
Writing values	0.0.0.0 ... 255.255.255.255

Setting a static route

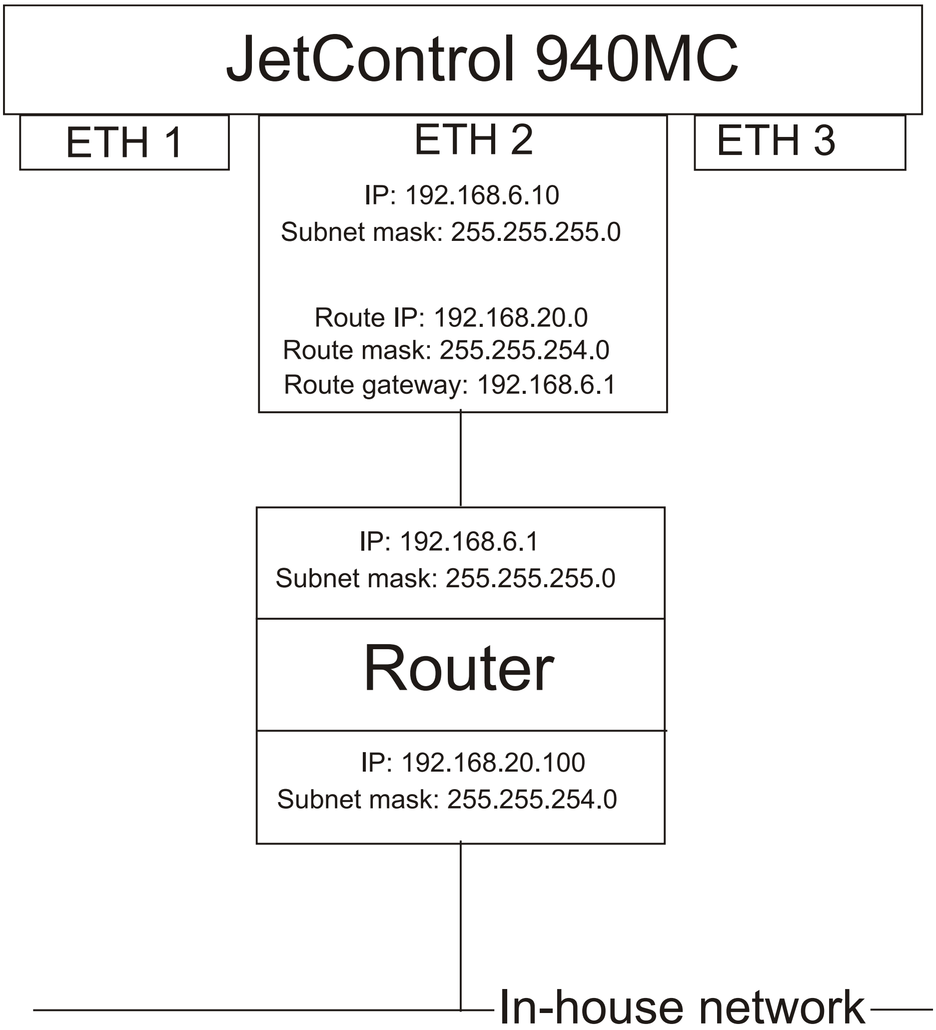
To either add the route to the routing table or delete it from there during runtime, proceed as follows:

Step	Action
1	Enter the port number into register 104552.
2	Enter the desired IP address into register 104553.
3	Enter the desired subnet mask into register 104554.
4	Enter the desired IP address of the Gateway into register 104555.
5	Enter the command into register 104551.

Result: If register 104550 has got value 0, the command has been executed successfully. The route settings are active.

Example

The JC-940MC is connected with your in-house network, for example, via its ETH 2 port and a router.



The JC-940MC transmits frames to addresses within the subnets 192.168.20.xxx and 192.168.21.xxx.The route set in the JC-940MC connects these with your in-house network via router.

Related topics

- **Setting the IP address via registers to be remanent** (see page 91)
- **Setting the IP address during runtime** (see page 95)

5 Initial commissioning

Purpose of this chapter

The first part of this chapter gives a compact description of the initial commissioning of the controller JC-940MC and covers the following functions:

- Creation and execution of a program which increments a variable.

This chapter covers the initial commissioning of the bus node JX3-BN-ETH with the aid of the following steps:

- Configuring the hardware and installing of a JX3 station
- Configuring the software in JetSym
- Creating and executing a program to read out a module register of a peripheral module connected to the JX3-BN-ETH.
- Addressing the bus node JX3-BN-ETH
- Describing the Jetter Ethernet system bus by the following means:
    - Hardware Manager
    - Publisher/Subscriber mechanisms

Prerequisites

For initial commissioning the JC-940MC controller, the following prerequisites must be fulfilled:

- The controller is connected to a PC directly via Ethernet or via switch or a hub.
- The programming tool JetSym 4.2 or higher is installed on the PC.
- Mode selector S11 is in *STOP* position.
- The controller is connected with the JX3-BN-ETH bus node.

Contents

Preparations for initial commissioning of the controller

Ethernet connection with the controller

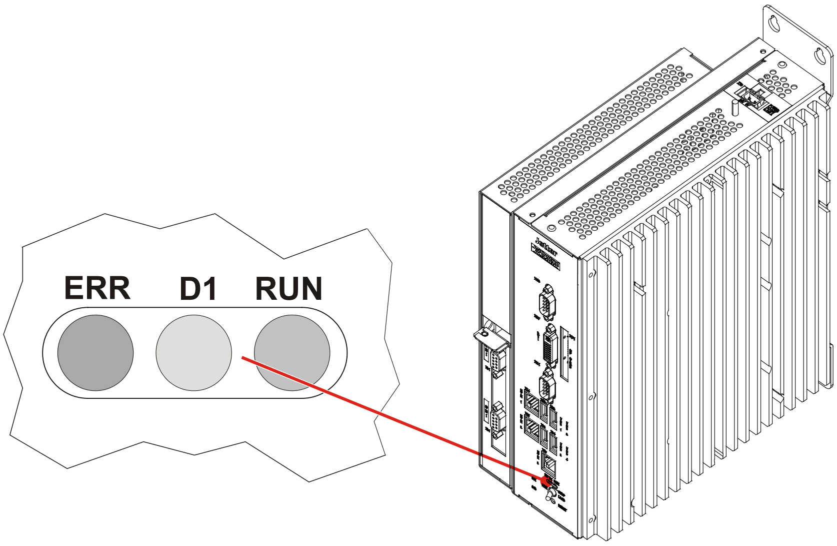
The controller JC-940MC in delivered condition has got IP address 192.168.1.1. Configure the Ethernet interface of your PC so that it is able to communicate with the controller via this IP address.

Behavior after power-up

If the mode selector is in position *STOP* when the controller is powered-up, the application program will not be launched.

States of the LEDs

Following a correct commissioning, the LEDs are lit as follows:

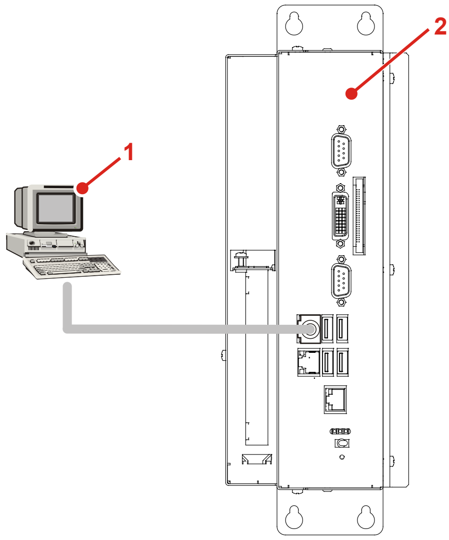


LED	State	Description
RUN	1Hz	Logic voltage supply is OK; application program has been stopped
ERR	OFF	No error
D1	OFF	No error

Initial commissioning of a JC-940MC

Configuration

The initial commissioning is based on the following configuration:



Number	Part	Description
1	PC	Programming system
2	JC-940MC	Controller

Creating a program

If you wish to create and check the program, proceed as follows:

Step	Action
1	Start the programming tool JetSym
2	Create a new project
3	Set the controller model (JC-940MC)
4	Set the IP address of the controller
5	Open the program editor
6	Enter the program specifications
7	Upload the project to the controller
8	Open a setup window
9	Enter the variable name (Count)
10	Activate the setup

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

JetSym STX program

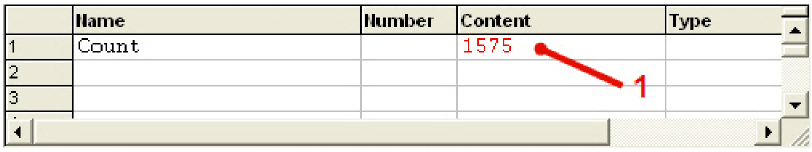
The following program increments the content of a variable by one every 2 seconds:

```
Var
    Count:    Int;
End_Var;

Task Increment Autorun
    Loop
        Inc(Count);
        Delay(T#2s);
    End_Loop;
End_Task;
```

Setup pane

The JetSym setup window displays the content of the variable:



Number	Description	Function
1	Present content of the variable	The content of the variable is incremented by one every 2 seconds

Configuration of a JX3 station at a JX3-BN-ETH

Introduction

JetSym is used to configure and program the bus node JX3-BN-ETH. This topic covers the following:

- Configuring the controller
- Configuring the bus node

Prerequisites

The following requirements must be satisfied:

- JetSym is installed on the PC used.
- JetSym has been licensed (see online help in JetSym).
- Bus node and controller have got different IP addresses.
- Limitations to be taken into account when engineering a JX3 station

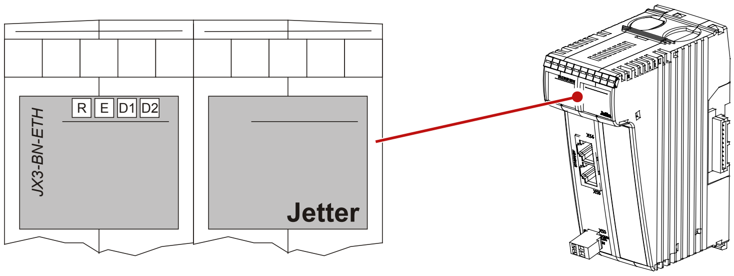
Overview: Commissioning steps

To successfully commission the bus node JX3-BN-ETH proceed as follows:

Step	Action
1	Make sure that the controller is disabled.
2	Add the required JX3 modules with a JX3-BN-ETH to form the JX3 station. When engineering a JX3 station, consider the limitations applying to its setup. Please refer to *jx3-bn-eth_ba_xxxy_manual*
3	Use Ethernet cables to connect the controller JX-9xx, the PC and the bus node JX3-BN-ETH to be a Jetter Ethernet system bus.
4	Set an IP address on the JC-9xx controller. For more information, refer to chapter *IP Configuration* in the manual on JC-9xx.
5	Set the IP address on the bus node JX3-BN-ETH. This IP address must differ from the IP address of the controller. Please refer to the chapter *IP Configuration* in *jx3-bn-eth_ba_xxxy_manual* For instance, the controller has got IP address 192.168.10.170. The bus node can then be assigned the default IP address 192.168.10.15. Take care that the first three elements of the IP addresses are identical.
6	Supply the JX3 station with power.
7	Launch JetSym Then configure the JX3 station following an example For more information, refer to chapter **Configuration in JetSym** (see page 109).
8	Configure the JX3 station using the Hardware Manager. For more information, refer to chapter **Hardware Manager** (see page 207).
9	Enter the sample program. Then, upload the program to the controller. For more information, refer to chapter **Hardware Manager Publish/Subscribe** (see page 214).

State of LEDs after power-up

If commissioning has been completed without error, the states of the LEDs on JX3-BN-ETH must be as follows:



LED	State	Description
R	ON	Logic voltage supply is OK
E	OFF	No error
D1	OFF	No error
D2	OFF	No error

Configuration in JetSym

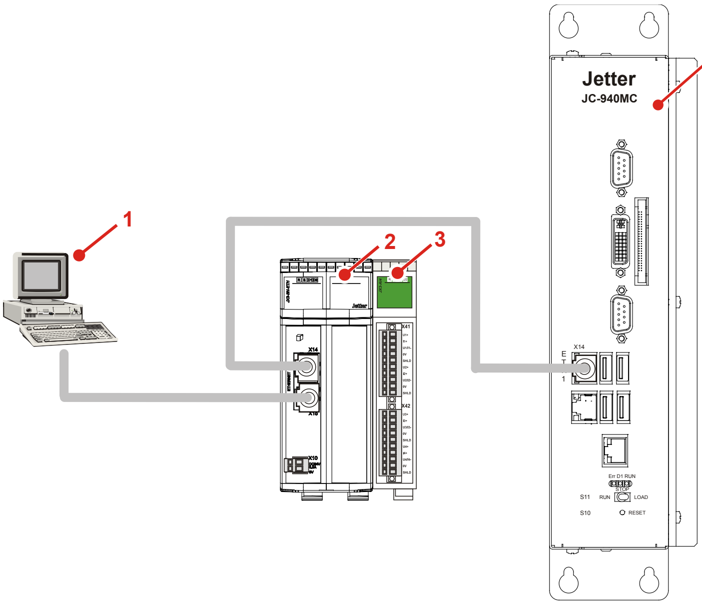
Introduction

A simple example is to illustrate configuration in JetSym: Connect the peripheral module JX3-AI4 with a bus node JX2-BN-ETH.

The minimalist program cyclically retrieves the state of the peripheral module.

Configuration

This example is based on the following configuration:

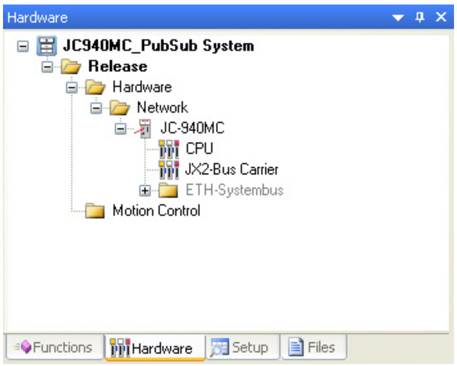
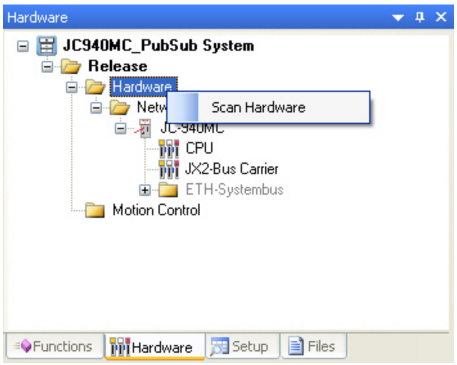


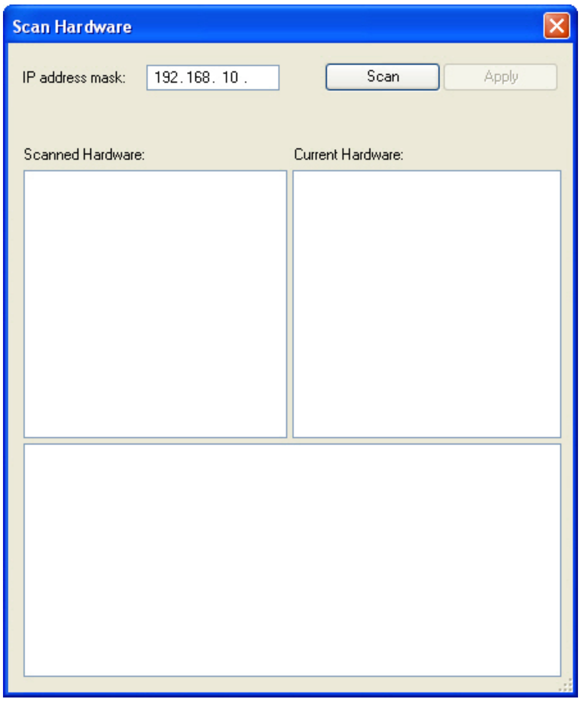
Number	Part	Description
1	PC	Programming system
2	JX3-BN-ETH	Bus nodes
3	JX3-AI4	Peripheral module
4	JC-940MC	Controller

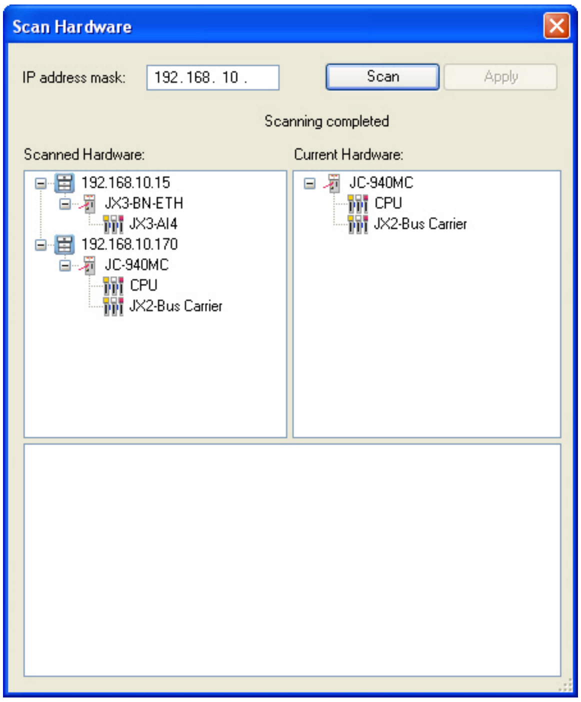
Preparatory work

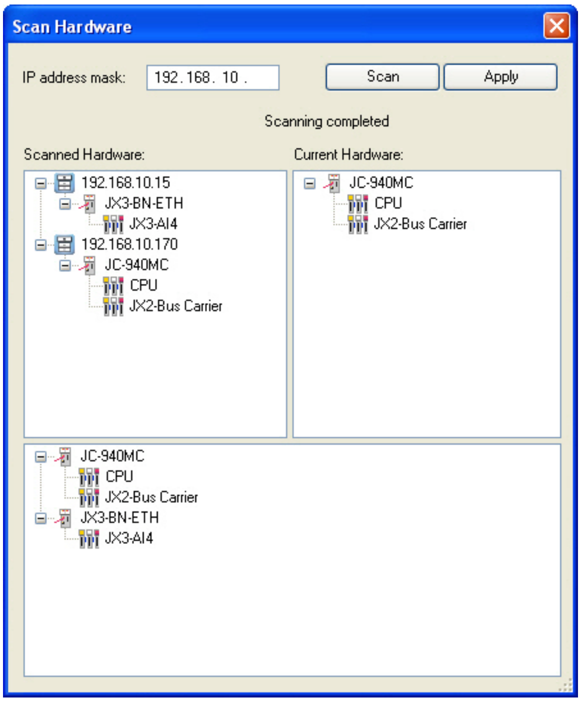
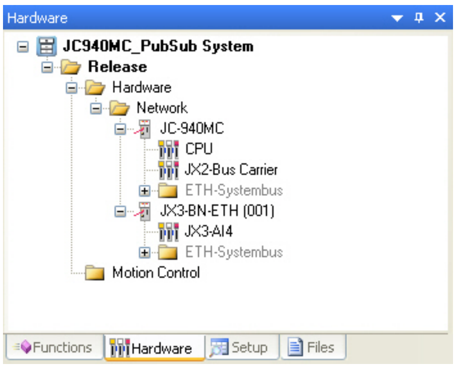
To properly configure the JX3 station, proceed as follows:

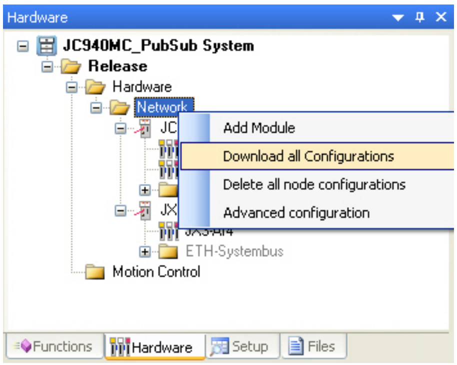
Step	Action
1	Start the programming tool JetSym.
2	Create a new project.

Step	Action
3	In JetSym, start the Hardware Manager by clicking on the tab **Hardware** with the mouse or by pressing **Alt + 5** on your keyboard. 
4	In Hardware Manager, click on the folder **Hardware** Right-click **Scan Hardware**. 

Step	Action
5	Enter an IP address mask. A hardware scan applies to the whole IP subnet. Therefore, you have to enter at least the first three elements of the IP address. In this example, the IP address of the controller JC-9xx is 192.168.10.170. To detect the control systems and all bus nodes, enter 192.168.10. 
6	Click the button **Scan**.

Step	Action
⇨	The Hardware Manager scans the Jetter Ethernet system bus and compares the scanned hardware with the really set hardware. 
7	In the window **Scanned Hardware**, click the name of the controller In this example, it is JC-940MC.
⇨	The Hardware Manager has the tree of the controller JC-940MC displayed in the bottom window.
8	Drag the entire tree of the JX3-BN-ETH into the lower window by Drag and Drop.

Step	Action
⇨	The Hardware Manager has the tree of the controller JX3-BN-ETH displayed in the bottom window. 
9	Click the button **Submit**.
⇨	The window closes. The Hardware Manager has taken over the hardware parameters. 

Step	Action
10	Download the current configuration to the controller by clicking the file **Network** and right-click on the menu item **Download All Configurations**. 
11	Activate the programming environment by entering **Alt + 0** on your keyboard. As an alternative, you can click the tab **File**.
12	Enter the program shown below.
13	Compile the program.
14	Upload the program to the controller.
⇨	The trace window shows the status of the peripheral module JX3-AI4. The status indication is updated every 2 seconds.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18
- Module JX3-AI4 of OS version 1.04

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

The following program lets you retrieve the global status register on the peripheral module JX3-AI4.

```
Var
// Reading the status of JX3-AI4 via JX3-BN-ETH:
JX3_AI4_Status:    Int At %VL 1001020000;
End_Var;

Task ReadState Autorun
Loop
// By Trace, display the content of the JX3-AI4 status register
// in hexadecimal notation:
Trace ('State JX3-AI4: ' + StrFormat('Status = %x', JX3_AI4_Status)
+ '$n');
Delay(T#2s);
End_Loop;
End_Task;
```

Related topics

- **Hardware Manager** (see page 207)

6 File system

Introduction

This chapter describes the file system of the JC-940MC. The file system enables access to files located on the internal flash disk or an USB flash drive.

Categories

The files of the file system are categorized as follows:

- System directories or system files used by the operating system
- A range that is at the user's disposal

System directories

The system directories cannot be deleted. They even still exist after formatting.

Directory	Description
/System	▪ System configuration ▪ System information
/USB1	▪ Root directory of the USB flash drive

Contents

6.1 Properties

Introduction

This chapter covers the properties of the file system. The description distinguishes between the internal flash disk and the USB flash drive.

General properties

The following properties apply to the internal flash disk and USB flash drive:

- 8 files max. to be opened simultaneously
- Separate directory names by a slash "/", not by a backslash "\".
- When the controller creates a file, the file contains date and time assigned by the controller.
- Date, time, and file size are not available for all system files.

Contents

Flash disk - Properties

Size

The following disk space is available to the user:

Parameter	Value
Flash disk capacity	8 MBytes

Properties

The internal flash disk drive has got the following properties:

- Up to 7 directory levels and 1 file level are allowed.
- Upper- and lower-case are distinguished.
- Directory and file names are permitted to have a total length of 63 characters.
- All characters except "/" and ".." are permitted for directory and file names.
- User/access administration for a maximum number of 31 locks and 33 users

USB flash drive - Properties

Size

The available capacity depends on the USB flash drive used:

Parameter	Value
Tested capacity	256 MByte ... 16 GByte

Properties

The USB flash drive has got the following properties:

- The USB flash drive must be compatible with FAT 16 or FAT 32.

To store data to the USB flash drive, please mind:

- Directory and file names are permitted to have a total length of 260 characters.
- Upper- and lower-case are not distinguished.
- The following characters are not allowed in directory and file names: "/", "\", ":", "*", "?", "<", ">" and "|"
- There is no user/access administration.

Jetter AG cannot guarantee the proper functioning of all USB flash drives available on the market.

6.2 User administration

Introduction

The file system for the internal flash disk lets you define authorization for access (locks) to directories, and set up users.

For each user, you can set individual access rights (keys).

Users are not allowed to access directories and files for which they do not have the required key. In case of an FTP/IP connection, these directories and files are not displayed.

Prerequisites

Administrator rights are required for user administration.

Properties

The properties of user administration are as follows:

Property	Max. value
Number of users	33
Number of predefined users	2
Length of a user name	31 alphanumeric characters
Password length	31 alphanumeric characters
Number of keys for read access	31
Number of keys for write access	31
Number of predefined keys	2

Files

You can make settings for user administration in three files located in the folder **System**.

File	Description
flashdisklock.ini	Assignment of locks to directories
keys.ini	Assignment of names to locks/keys
users.ini	Administration of users

These files are always existing. They cannot be deleted, but only modified or overwritten.

Restrictions

Please take the following restrictions into account:

- User administration can only be applied to the internal flash disk. It cannot be applied to the USB flash drive.
- If user administration has been assigned to a file, its contents are readable at once. The settings become active when the system is rebooted.

Contents

Administration of users

Introduction

The configuration file **/System/users.ini** lets you manage the user administration for the file system.

Prerequisites

If you want to use names for the keys, you must make them known to the device beforehand. Therefore, set up the names first as described in *Setting up names for keys/locks* (see page 128).

Administration of users

To manage user administration, proceed as follows:

Step	Action
1	Establish an FTP connection to the device. Log on as administrator.
2	Open the file **/System/users.ini**.
3	Enter the required information.
4	Save the changed file to the device.
5	Reboot the device.

Result: The changed user administration settings are now enabled.

Structure of the file */System/users.ini*

This configuration file is a text file the entries of which are grouped into several sections.

- For each user a separate section is to be created.
- In these sections values can be set which are then used by the file system.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Sections

The sections are named *[USER1]* through *[USER33]*. Here, the user name and the related password, as well as read and write permissions are specified.

Example:

```
[USER4]
NAME=TestUser3
PW=testpass
READKEYS=5,openLock2,10,11
WRITEKEYS=openLock2,10,11
SYSKEYS=
```

NAME	
In the given example	TestUser3
Description	User's login name
Allowed values	A maximum of 31 alphanumeric characters
In case of illegal value or missing entry	User administration settings are not made
PW	
In the given example	testpass
Description	User's login password
Allowed values	A maximum of 31 alphanumeric characters
In case of missing entry	The user is allowed to log in without password
READKEYS	
In the given example	5,openLock2,10,11
Description	Keys for read access (read keys)
Allowed values	1 ... 31 (or corresponding names)
In case of missing entry	No read keys are assigned to the user
WRITEKEYS	
In the given example	openLock2,10,11
Description	Keys for write access (write keys)
Allowed values	1 ... 31 (or corresponding names)
In case of missing entry	No write keys are assigned to the user
SYSKEYS	
Description	No function assigned; reserved for future extensions

As-delivered condition/Predefined users and keys

Introduction

Two predefined users with set rights are included in the file system. It is not possible to delete these two users. The user administration lets you only change the password for these two users.

Factory settings

In delivered condition the content of the configuration file included in the controller is as follows.

```
[USER1]
NAME=admin
PW=admin
READKEYS=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,2
2,23,24,25,26,27,28,29,30,31
WRITEKEYS=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31
SYSKEYS=

[USER33]
NAME=system
PW=system
READKEYS=2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
23,24,25,26,27,28,29,30,31
WRITEKEYS=2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
,23,24,25,26,27,28,29,30,31
SYSKEYS=
```

User *admin*

All keys are available to this user *admin*, and he/she is, therefore, able to read all directories and files and to write to them.

User *system*

All keys except for key **1** are available to user *system*, too.

Predefined keys

Two out of the 31 keys have a predefined function.

Lock/key	Function
1	▪ Ethernet configuration ▪ User administration
2	▪ Operating system update of the CPU ▪ Operating system update of JX2 and JX3 modules

Assigning locks

Introduction

In the configuration file **/System/flashdisklock.ini** you assign locks to directories located on the flash memory. Only users with the corresponding key are allowed to read or write (delete) files and subdirectories located in these directories.

Prerequisites

If you want to use names for the locks, you must make them known to the device beforehand. Therefore, set up the names first *Setting up names for keys/locks* (see page 128).

Installing the lock

To assign a lock to a directory, proceed as follows:

Step	Action
1	Establish an FTP connection to the device; when doing so, log in with administrator rights.
2	Open the file **/System/flashdisklock.ini**.
3	Adjust the file entries.
4	Save the changed file to the device.
5	Reboot the device.

Result: A lock is assigned to this directory.

Structure of the file */System/flashdisklock.ini*

This configuration file is a text file containing one section.

- In this section values can be set which are then used by the file system.
- Specify each directory with its lock number in an individual line.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Section

The section is named *[LOCKS]*. Here, locks are assigned to directories in accordance with the following rule:

Directory=Lock

Example:

```
[LOCKS]
test1=0
test1/sub1=2
test1/sub2=5
test2=userlock2
```

Lock numbers

Use the following lock numbers:

- The valid lock numbers are 0 ... 31.
- Lock number 0: No lock is assigned to this directory. You can access this directory without any special permissions.
- You can use numbers or previously defined names.

Assigning names to keys/locks

Introduction

Keys/locks are consecutively numbered from 1 through 31. To provide ease of handling, a name can be assigned to each key/lock combination. These names are assigned in the configuration file **/System/keys.ini**.

Configuring names

To assign names to keys/locks, proceed as follows:

Step	Action
1	Establish an FTP connection to the device; when doing so, log in with administrator rights.
2	Open the file **/System/keys.ini**.
3	Adjust the file entries.
4	Save the changed file to the device.
5	Reboot the device.

Result:

The names are available now. The names are now available and can be used when assigning locks and managing user accounts.

Structure of the file *?/System/keys.ini*

This configuration file is a text file containing one section.

- In this section values can be set which are then used by the file system.
- Each key is specified with its name in an individual line.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Section

The section is named *[KEYS]*. Here, names are assigned to keys/locks in accordance with the following rule:

KEYxx=Name

xx: Number of the key (01 ... 31)

Example:

```
[KEYS]
KEY01=Admin
KEY02=System
KEY03=
KEY04=
KEY05=service
...
KEY31=
```

Names for Locks/Keys

For names the following definitions are true:

- A maximum of 15 alphanumeric characters
- Lock and key have the same name.

6.3 Reviewing the flash disk capacity used

Introduction

You can view the application scope of the internal flash disk.
Details on the allocation of the application scope are given in this chapter.

Contents

Flash disk capacity used

README

You can view the application scope of the internal flash disk.

You can see the capacity used of the user area from the file **/System/flashdiskinfo.txt**.

Example

In this example, the fictive capacity used of a flash disk in a JetControl 940MC (8 MByte) is shown:

```
Name  : flash disk
Date  : 07.04.2011
Time  : 08:58
Tracks: 126

Track   0:  sectors: 128  (used:  97 / blocked:  31 / free:   0)
Track   1:  sectors: 128  (used: 111 / blocked:  17 / free:   0)
Track   2:  sectors: 128  (used:   5 / blocked:  20 / free: 103)
Track   3:  sectors: 128  (used:  97 / blocked:  31 / free:   0)
Track   4:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track   5:  sectors: 128  (used:  90 / blocked:  38 / free:   0)
Track   6:  sectors: 128  (used:   5 / blocked:   0 / free: 123)
Track   7:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track   8:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track   9:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  10:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  11:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  12:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  13:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  14:  sectors: 128  (used:  66 / blocked:  62 / free:   0)
Track  15:  sectors: 128  (used:  80 / blocked:  48 / free:   0)
Track  16:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  17:  sectors: 128  (used: 103 / blocked:  25 / free:   0)
Track  18:  sectors: 128  (used:  83 / blocked:  45 / free:   0)
Track  19:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  20:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  21:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  22:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  23:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  24:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  25:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  26:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  27:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  28:  sectors: 128  (used:  79 / blocked:  49 / free:   0)
Track  29:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  30:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  31:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  32:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  33:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
```

```
Track  34:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  35:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  36:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  37:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  38:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  39:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  40:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  41:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  42:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  43:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  44:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  45:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  46:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  47:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  48:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  49:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  50:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  51:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  52:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  53:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  54:  sectors: 128  (used: 106 / blocked:  22 / free:   0)
Track  55:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  56:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  57:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  58:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  59:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  60:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  61:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  62:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  63:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  64:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  65:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  66:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  67:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  68:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  69:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  70:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  71:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  72:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  73:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  74:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  75:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  76:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  77:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  78:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  79:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  80:  sectors: 128  (used:  36 / blocked:  33 / free:  59)
Track  81:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  82:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
```

```
Track  83:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  84:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  85:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  86:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  87:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  88:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  89:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  90:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  91:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  92:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  93:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  94:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  95:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  96:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  97:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  98:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  99:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 100:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 101:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 102:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 103:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 104:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 105:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 106:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 107:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 108:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 109:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 110:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 111:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 112:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 113:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 114:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 115:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 116:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 117:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 118:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 119:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 120:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 121:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 122:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 123:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 124:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track 125:  sectors: 128  (used:   0 / blocked:   0 / free: 128)

Total:  sectors: 16128  (used: 4286 / blocked: 421 / free: 11421)

Used   :  2177288 byte
Blocked:   213868 byte
Free   :  5801868 byte
```

```
Total  :  8193024 byte
```

Elements of info file

Tracks and sectors represent the administration units of the flash disk. The info file comprises the following elements:

Element	Description
`Name`	Dedicated name of the flash disk
`Date/Time`	Point in time when the flash disk was formatted last
`Tracks`	Total number of tracks
`Track xx: sectors: 128`	Assignment of sectors of a track
`Total: sectors:`	Overall statistical data of sectors
`Used`	Total number of used bytes
`Blocked`	Total number of blocked bytes
`Free`	Total number of available bytes
`Total`	Total size of the flash disk

States of sectors

The smallest administrative unit of the flash disk, i.e. the sector, may enter the following states:

State	Meaning
`used`	The sector is occupied by data.
`blocked`	The sector is no longer occupied, but cannot be used yet due to administrative reasons.
`free`	The sector is not occupied and can be used.

6.4 Operating system update and application program

Introduction

An OS update for a controller, an HMI or an I/O module, as well as access to the application program can be carried out via file system. For a detailed description on this topic refer to the following chapter:

- *Operating system update* (see page 393)
- *Application program* (see page 409)

6.5 Formatting and checking

Introduction

This chapter describes formatting the internal flash disk.

The internal flash disk needs not be checked using a separate function, since it provides maximum safety of its administrative structures by design.

Operating principle

When the device boots up, the OS system checks the content of the control register. The control register is part of the file system.

Depending on the value contained in this register the following functions are carried out:

- Formatting the flash disk

Register number

The register number of the control register is dependent on the device:

Device	Register number
JC-24x	2936
JM-D203-JC-24x	2936
JC-340, JC-350, JC-360, JC-360MC	202936
JC-940MC	202936
JVM-407	202936
JX3-BN-ETH	202936
JX3-COM-EIPA	202936

Contents

Formatting the flash memory

Introduction

In the following cases, reformatting the flash disk is required:

- When you upload an OS version that has got another flash disk format
- When information for flash disk administration has been destroyed

Consequences

- All files and directories located in the user area will be deleted!
- Formatting will not affect system files and directories.

Formatting the flash disk

To have the device format the internal flash disk, proceed as follows:

Step	Action
1	Switch the device ON.
2	Enter value -999720373 (0xc4697a4b) into the control register of the file system.
3	Switch the device OFF.
4	Switch the device ON.

Result: During the boot process of the JC-940MC, the flash disk is formatted and the control register is set to 0.

7 FTP server

Introduction

The FTP server allows access to directories and files using an FTP client. The files can be stored to the following storage media:

- Flash disk integrated into the controller
- USB flash drive

This chapter covers the login process and describes the commands supported by the FTP server.

FTP clients

The user has the option of using a command line FTP client, which comes with many PC operating systems, or graphic FTP tools.

Number of possible connections

The FTP server on the JC-940MC is able to manage four FTP connections simultaneously. That is, up to 4 FTP client programs can be connected with the JC-940MC at the same time.

Any additional client, which tries to connect with the FTP server, will get no response to its request for establishing a connection.

Required programmer's skills

To be able to use the functions described in this chapter, the following skills are required:

- The user must be familiar with the file system of the controller.
- The user must be familiar with IP networks.

Contents

Logon

Logon

To be able to access the file system via FTP, the FTP client must log on when the connection is established.

- As **Server Name** enter the IP address of the device.
- As **User Name** enter your user name, e.g. *admin*.
- As **Password** enter your password, e.g. *admin*.

Factory settings

The factory settings include two user accounts:

[USER1]
NAME=admin
PW=admin

[USER33]
NAME=system
PW=system

Administration of users

The user administration of the file system lets you change the password and add new users.

Related topics

- **User administration** (see page 121)

Supported commands

Supported commands

The following table lists the commands known to the FTP server, as well as their purpose.

Command	Description
USER	Sends the user name; is used at the beginning of the login process
PASS	Sends the password; is sent after USER to complete the login process
QUIT	Terminates the connection
PORT	Specifies the IP address and port number to which the FTP server is to connect for the next file transfer.
TYPE	Sets the transfer type; the following types are possible: ■ Type A with interpretation N ■ Type I ■ Type L with 8 bits per character
MODE	Sets the transfer mode; here, only "S" (stream) is possible
STRU	Sets the file structure when transferring data; here, only "F" (file) is possible
NLST	Returns a list containing the file names of a directory
LIST	Returns a list containing the file names and file information of a directory
PWD	Returns the name of the current directory
CWD	Switches to another directory
CDUP	Moves up by one directory level
MKD	Creates a new directory
RMD	This instruction is for removing a directory
STOR	Stores a file
RETR	Reads a file
DELE	Deletes a file
RNFR	Indicates the file name to be changed; Command *RNFR* must be followed by *RNTO*
RNTO	Indicates the new name of the file which has been selected by the command *RNFR* before.
PASV	The FTP server enters into *passive mode*

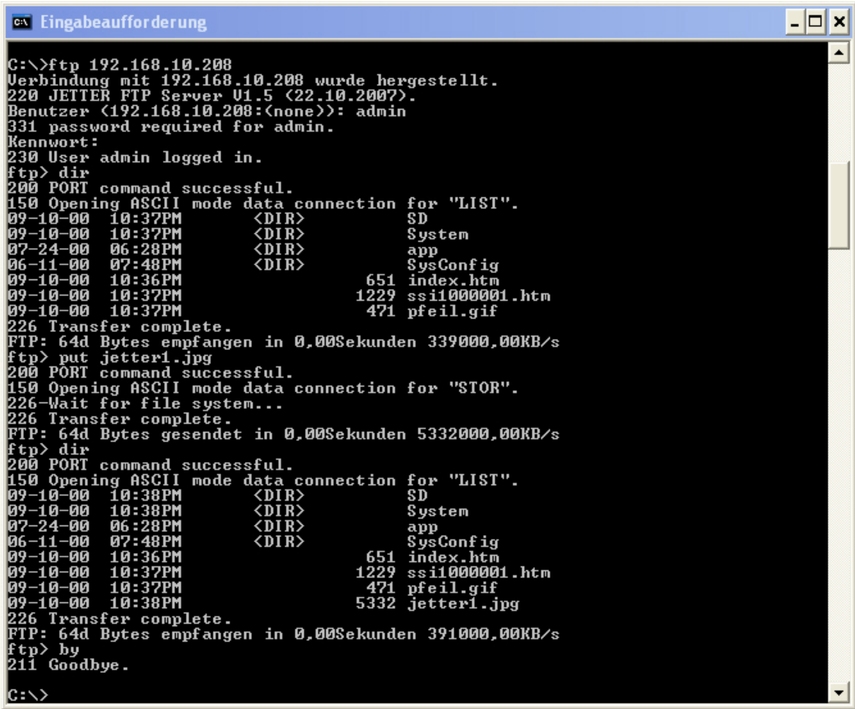
Example: Windows FTP client

Task

Carry out the following tasks using an FTP client, for example, the one which comes with Windows XP:

- Launch the FTP client by opening a connection and entering the IP address.
- Log in as user *admin* with password *admin*
- Displaying the content of the current directory using *dir*
- Transferring the file **jetter1.jpg** to the JetControl using the command *put*
- Displaying the content of the current directory using *dir*
- Terminating the session and the FTP client using *bye*

Action



8 HTTP server

Introduction

A standard browser is sufficient for accessing the HTTP server.

The browser is for reading and displaying files which have been downloaded to the controller via FTP.

Here, it may be necessary to enter the user name and password to have access to certain pages (depending on the file system configuration).

This chapter covers the *Server Side Includes (SSI)* function included in the HTTP server.

Default file names

The default file names are **index.htm** and **index.html**.

Supported file types

The following file types are supported:

- *.htm, *.html, *.shtml
- *.txt, *.ini
- *.gif, *.tif, *.tiff, *.bmp, *.wbmp
- *.jpg, *.jpe, *.jpeg, *.png
- *.xml
- *.js, *.jar, *.java, *.class, *.cab
- *.ocx
- *.pdf, *.zip, *.doc, *.rtf
- *.css
- *.wml, *.wmlc, *.wmls, *.wmlsc
- *.ico, *.svg

Required programmer's skills

To be able to use the functions described in this chapter, the following skills are required:

- The user must be familiar with the file system of the controller.
- The user must be familiar with IP networks.

Contents

8.1 Server Side Includes

Introduction

The HTTP server features **Server Side Includes** (SSI). This function is for showing present real-time controller values on an HTML page.

Rules

You must specify a **namespace tag** at the beginning of the HTML page that is to contain the real-time controller values.

This **namespace tag** is for defining the **namespace** used in the HTML page.

In the body section of the HTML page the **data tags** are specified.

Updating real-time controller values

When the HTML page is uploaded to the browser, the HTTP server once replaces the data tags by actual real-time controller values.

To refresh the controller values, the HTML page must be reloaded over and over again.

Contents

Namespace tag

Namespace tag - Structure

The name **namespace tag** must be the first entry in the HTML file. It has got the following structure:

```
<NS:DTAG xmlns:NS=http://jetter.de/ssi/jetcontrol/
```

with **NS** representing the **namespace**. The **namespace** is a character string with a maximum length of 63 characters.

The **namespace** introduced here will be re-used for the subsequent data tags. The remaining parts of the line are preassigned and have to be specified in exactly the same way.

In the following examples, the namespace applied is JW.

Inserting real-time controller values

Introduction

Actual real-time controller values are integrated into parameter entries within the sections via tag functions. This way, the contents respectively states of registers, text registers, inputs, outputs and flags can be displayed.

Tag delimiters

All tags start and end with defined strings. Between these tag delimiters, the variables are defined.

Delimiter	String
Tag start	<JW:DTAG
Tag end	/>

Variable definition

The variable definition in a tag contains attributes which are used to set, for example, how the value of a variable is to be displayed:

name	
Description	Variable name
Comments	Code letter followed by the variable number
Example	`name="R1000023"`

type	
Description	Variable type of notation
Example	`type="REAL"`

format	
Description	Representation format
Comments	Refer to format definition
Example	`format="+0####.###"`

factor	
Description	Factor by which the real-time controller value is multiplied
Comments	Multiplication is executed prior to adding the offset.
Example	`factor="1.5"`

offset	
Description	Value which is added to the real-time controller value
Comments	Multiplication by the factor is executed prior to adding the value to the real-time controller value.
Example	`offset="1000"`

Format definition

You can define the representation of variables by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" sets the output of leading zeros. This applies to the register types INT, INTX and REAL.
- Prefix "+" sets the output of a sign. This applies to the register types INT and REAL.
- Prefixing a blank sets the output of a blank. This applies to the register types INT and REAL.

Registers/text registers

The variable name begins with a capital "R" followed by the register number.

The following types are possible:

Type	Notation
INT	Integer, decimal
INTX	Integer, hexadecimal
INTB	Integer, binary
BOOL	Register content = 0 --> Display: 0 Register content != 0 --> Display: 1
REAL	Floating point, decimal
STRING	Text register

Standard type: INT

Example:

```
JW:DTAG name="R1000250" type="REAL" format="+0####.###"
factor="3.25" offset="500" /
```

Result:

This instruction causes the contents of register 1000250 to be multiplied by 3.25 and then added to product 500. The result appears in the Web browser with sign and at least five integer positions before the decimal point. If need be, five leading zeros are added. Furthermore, three decimal positions are added.

Flags

The variable name begins with a capital "F" followed by the flag number.

The following types are possible:

Type	Notation
BOOL	Flag = 0 --> Display: 0 Flag = 1 --> Display: 1
STRING	Flag = 0 --> Display: FALSE Flag = 1 --> Display: TRUE

Standard type: BOOL

Example:

```
<JW:DTAG name="F100" type="STRING" format="#" />
```

Result:
The state of flag 100 is displayed as string "T" or "F".

Inputs

The variable name begins with a capital "I" followed by the input number. The following types are possible:

Type	Notation
BOOL	Input = 0 --> Display: 0 Input = 1 --> Display: 1
STRING	Input = 0 --> Display: OFF Input = 1 --> Display: ON

Standard type: BOOL

Example:

```
<JW:DTAG name="I201200308" type="STRING" />
```

Result:
The state of input 201200308 on the CPU is displayed as string "ON" or "OFF".

Outputs

The variable name begins with a capital "O" followed by the output number. The following types are possible:

Type	Notation
BOOL	Output = 0 --> Display: 0 Output = 1 --> Display: 1
STRING	Output = 0 --> Display: OFF Output = 1 --> Display: ON

Standard type: BOOL

Example:

```
<JW:DTAG name="O201100308" />
```

Result:
The state of output 201100308 is inserted as "1" or "0".

Access via pointer register

Access via pointer register is realized by inserting the capital letter "P" in front of the variable name. In each case the value of the variable is displayed whose number corresponds to the content of the register specified in the variable name.

Examples:

```
<JW:DTAG name="PR1000300" />
```

Result: The content of the register is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PF1000300" />
```

Result: The state of the flag is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PI1000300" />
```

Result: The state of the input is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PO1000300" />
```

Result: The state of the output is displayed whose number is contained in register 1000300.

Access via pointer register and offset

To specify the number of the variable to be displayed, it is also possible to add a constant value or another register content to the pointer register value

Examples:

```
<JW:DTAG name="PR1000300 + 100" />
```

Result: The content of the register is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PR1000300 + R1000100" />
```

Result: The content of the register is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PF1000300 + 100" />
```

Result: The state of the flag is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PF1000300 + R1000100" />
```

Result: The state of the flag is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PI1000300 + 100" />
```

Result: The state of the input is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PI1000300 + R1000100" />
```

Result: The state of the input is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PO1000300 + 100" />
```

Result: The state of the output is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PO1000300 + R1000100" />
```

Result: The state of the output is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

Example of an HTML page

Task

Insert current real time controller values into an HTML page.

It should then be possible to display the HTML page in a browser using the **Server Side Include** feature.

Action

```
<JC:DTAG xmlns:JC="http://jetter.de/ssi/jetcontrol" />
<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgID" content="FrontPage.Editor.Document">
<title>Index</title>
</head>

<body>
Hello World, 
<p>Actual controller values can be inserted into an html page like
this: </p>
<p>Register 201000 = <JC:DTAG name="R201000" type = INT
format="+####" />,
or Hex: 0x<JC:DTAG name="PR201000+10" type="INTX" format="0###" />,
or rather this way: <JC:DTAG name="R201000" type="BOOL" />, if only
Boolean is queried.
But binary is also possible: <JC:DTAG name="R201000" type="INTB"
format=######## />b. </p>
<p>Strings could also be defined "<JC:DTAG name="R201000"
type="STRING" />". </p>
<p>A real number looks as follows: <JC:DTAG name="R1001500"
type="REAL" />
or this way: <JC:DTAG name="R1001500" type="REAL" factor="1.3"
format="###.##" />. </p>
<p>The value of a flag is represented as follows: <JC:DTAG name="F10"
/>
or <JC:DTAG name="PF1000000" type="STRING" />. </p>
<p>The same way, it is done the same way: <JC:DTAG name="PI1000130"
type="BOOL" />
or <JC:DTAG name="201100205" type="STRING" />. </p>
<p>R201000 = <JC:DTAG name="R201000" type="INT"
format="+0##########" /> </p>
<p>Regards </p>
<p>Yours JetControl</p>
</body>

</html>
```

9 Programming

Purpose of this chapter

This chapter is for supporting you when programming a JC-940MC controller in the following fields of activity:

- Determining the register numbers of connected modules
- Determining the I/O numbers of connected modules
- Programming additional functions

Prerequisites

To be able to program the JC-940MC controller, the following prerequisites must be fulfilled:

- The controller is connected to a PC.
- On the PC, the JetSym programming software has been installed.

Contents

Abbreviations, module register properties and formats

Abbreviations

The abbreviations used in this document are listed in the table below:

Abbreviation	Description
R 100	Register 100
MR 150	Module register 150

Module register properties

Each module register is characterized by certain properties. Most properties are identical for many module registers - the value after reset is always zero, for example. In the following description, module register properties are mentioned only if a property deviates from the following default properties.

Module register properties	Default property for most module registers
Access	Read/write
Value after reset	0 or undefined (e.g. release number)
Takes effect	Immediately
Write access	Always
Data type	Integer

Numerical formats

The numerical formats used in this document are listed in the table below:

Notation	Numerical format
100	Decimal
0x100	Hexadecimal
0b100	Binary

JetSym sample programs

The notation for sample programs used in this document is listed in the table below:

Notation	Description
`Var, When, Task`	Keyword
`BitClear();`	Commands
`100 0x100 0b100`	Constant numerical values
`// This is a comment`	Comment
`// ...`	Further program processing

9.1 Memories - Overview

Introduction

The JC-940MC features several types of program and data memories. There is volatile memory. Volatile memory loses its content at switching off. Non-volatile memory keeps its content even when the power supply is off.

The memory is located directly in the CPU or in separate memory or I/O modules.

This chapter gives an overview of the available memory.

Contents

Operating system memory

Introduction

The OS is stored to a non-volatile flash memory in the CPU. Therefore, the OS can be executed immediately after the device is powered up.

Characteristics

- Internal flash memory for storing OS data
- Internal volatile RAM for storing OS data

Memory access

- The user is not allowed to directly access the OS memory.
- To modify the OS, it must be updated.

Related topics

- **OS update** (see page 394)

File system memory

Introduction

The file system memory is for storing data and program files.

Properties

- Internal flash disk
- Non-volatile
- Slow access: Milliseconds up to seconds
- A limited number of write/delete cycles: approx. 1 million
- Internal flash disk size: 8 MBytes

Memory access

- By operating system
- By JetSym
- Via FTP connection
- By the e-mail client
- By browser (via HTTP server)
- By means of file commands from within the application program

Application program memory

Introduction

By default, the application program is uploaded from JetSym to the controller and is stored to it.

Properties

- Stored as file within the file system
- Default directory */app*
- Files may also be stored to other directories
- Size limited by flash memory

Memory access

- By operating system
- By JetSym
- Via FTP connection
- By means of file commands from within the application program

Related topics

- **Application program** (see page 409)

Memory for volatile application program variables

Introduction

Volatile variables are used to store data which need not be maintained when the JC-940MC is de-energized.

Properties

- Global variables which are not assigned to permanent addresses (not %VL or %RL)
- Local variables
- Variables are stored in a compact way.
- Variables are initialized with value 0 when they are created.

Memory access

- By JetSym
- From the application program

JetSym STX program

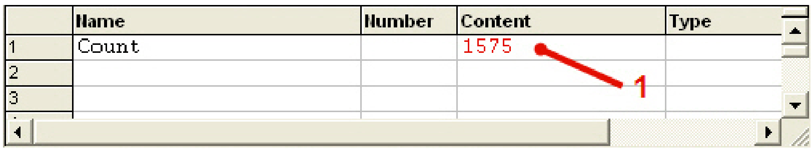
The following program increments the content of a global variable by one every 2 s.

```
Var
    Count:    Int;
End_Var;

Task Inkrement Autorun
    Loop
        Inc(Count);
        Delay(T#2s);
    End_Loop;
End_Task;
```

Setup pane

The JetSym setup pane displays the content of the variable.



Number	Description	Function
1	Present content of the variable	The content of the variable is incremented by one every two seconds.

Memory for non-volatile application program registers

Introduction

Non-volatile registers are used to store data which must be maintained when the JC-940MC is de-energized.

Properties

- Global variables which are assigned to permanent addresses (%VL)
- Register variables always occupy 4 bytes.
- Register variables are not initialized by the operating system.
- Number of register variables: 120,000
- Register numbers: 1000000 ... 1119999

Memory access

- By JetSym
- By the e-mail client
- By browser (via HTTP server)
- From HMIs
- From the application program
- From other controllers

JetSym STX program

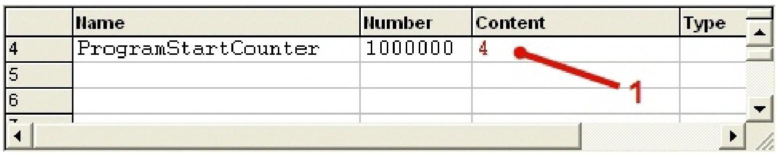
The following program increments the content of a register variable every time the application program is started. This way, the number of program starts is counted.

```
Var
    ProgramStartCounter:    Int At %VL 1000000;
End_Var;

Task Work Autorun
    ProgramStartCounter := ProgramStartCounter + 1;
    Loop
        // ...
    End_Loop;
End_Task;
```

Setup pane

The JetSym setup pane displays the content of the register variable.



Number	Description	Function
1	Present content of the register variable	The content of the register variable is incremented by one every time the program is started.

Memory for non-volatile application program variables

Introduction

Non-volatile variables are used to store data which must be maintained when the JC-940MC is de-energized.

Properties

- Global variables which are assigned to permanent registers (%RL)
- Variables are stored in a compact way.
- Size: 480,000 bytes
- Register numbers: 1000000 ... 1119999

Memory access

- By JetSym
- From the application program

JetSym STX program

The following program increments the content of four non-volatile variables every second.

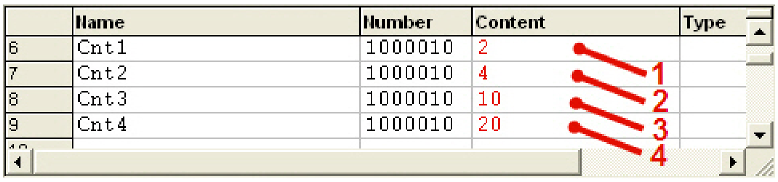
The working range of the counters is between 0 and 255 (variable type: byte). For these four variables the four bytes of register 1000010 are used.

```
Var
    Cnt1, Cnt2, Cnt3, Cnt4:    Byte At %RL 1000010;
End_Var;

Task Count4 Autorun
    Loop
        Inc(Cnt1);
        Inc(Cnt2, 2);
        Inc(Cnt3, 5);
        Inc(Cnt4, 10);
        Delay(T#1s);
    End_Loop;
End_Task;
```

Setup pane

The JetSym setup pane displays the content of the variable. As the type of the four counters is byte, this will result in counter overflow after a relatively short time:



Number	Description	Function
1	Present content of the variable Cnt1	The content of the variable is incremented by one every second.
2	Present content of the variable Cnt2	The content of the variable is incremented by two every second.

Number	Description	Function
3	Present content of the variable Cnt3	The content of the variable is incremented by five every second.
4	Present content of the variable Cnt4	The content of the variable is incremented by ten every second.

Registers on I/O modules

Introduction

These registers are located on modules connected to the JX2 system bus.

Properties

- Global variables which are assigned to permanent addresses (%VL)
- Type depending on the module
- Register numbers on JX2 system bus: 201100000 ... 201227999

Memory access

- By JetSym
- By the e-mail client
- By browser (via HTTP server)
- From HMIs
- From the application program
- From other controllers

JetSym STX program

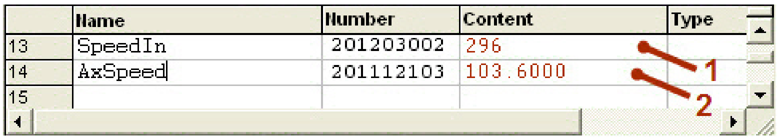
The following program calculates the set speed of a servo axis at the first JX2 system bus (AxSpeed) from an analog value. The analog value results from a measuring (SpeedIn) via a module at the second JX2 system bus.

```
Var
    AxSpeed:    Float At %VL 201112103;
    SpeedIn:    Int   At %VL 201203002;
End_Var;

Task SetSpeed Autorun
    Loop
        If SpeedIn > 100 Then
            AxSpeed := SpeedIn * 0.35;
        End_If;
        Delay(T#100ms);
    End_Loop;
End_Task;
```

Setup pane

The JetSym setup window displays the content of the register variable:



Number	Description	Function
1	Present content of the register variable SpeedIn	Analog value on channel 1 of the JX2-AI4 module on the JX2 system bus
2	Present content of the register variable AxSpeed	Set speed of the servo amplifier JetMove on the first JX2 system bus

Special registers

Introduction

Special registers let you control OS functions and retrieve status information.

Properties

- Global variables which are assigned to permanent addresses (%VL)
- When the operating system is launched, special registers are initialized using default values.
- Register numbers: 100000 ... 999999
- Network registers: 1nnnxxxxxx (nnn = GNN)

Memory access

- By JetSym
- By the e-mail client
- By browser (via HTTP server)
- From HMIs
- From the application program
- From other controllers

Related topics

- **Access by registers to remote participants** (see page 190)
- **Indirect addressing of remote modules** (see page 198)
- **Indirect addressing with variable destination window** (see page 203)

Inputs and outputs

Introduction

Inputs and outputs are 1-bit variables. This means they can either have the value TRUE or FALSE.

Properties of virtual inputs/outputs

- Global variables assigned to permanent addresses (%IX, %QX)
- Used for RemoteScan via Modbus/TCP
- Quantity: 16,000
- I/O numbers: 20001 ... 36000

Properties of digital inputs/outputs

- Global variables assigned to permanent addresses (%IX, %QX)
- Located on modules connected to the JX2 system bus
- I/O numbers on the JX2 system bus: 201100201 ... 201203216
- I/O numbers of remote participants connected to a JX3-BN-ETH: 1nnn010201 ... 1nnn011716 (nnn = GNN)

Memory access

- By JetSym
- By the e-mail client
- By browser (via HTTP server)
- From HMIs
- From the application program

JetSym STX program

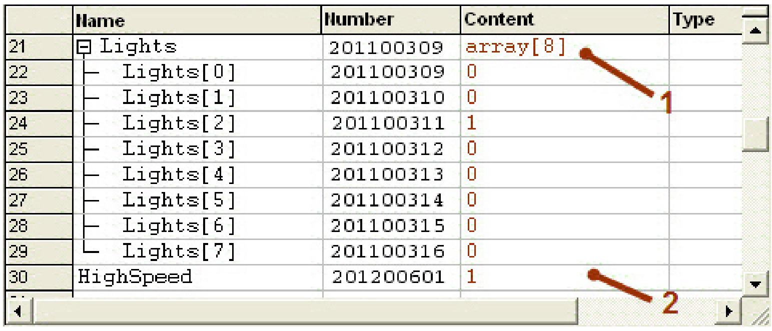
In the following program, moving lights are triggered on a digital output module connected to the first JX2 system bus. If a digital input is set at the second JX2 system bus, the corresponding output is activated for 100 ms.

```
Var
    Lights:    Array[8] Of Bool At %QX 201100309;
    HighSpeed:    Bool At %IX 201200601;
End_Var;

Task RunningLight Autorun
    Var
        Idx:    Int;
    End_Var;
    Loop
        For Idx := 0 To 7 Do
            Lights[Idx] := True;
            If HighSpeed Then
                Delay(T#100ms);
            Else
                Delay(T#300ms);
            End_If;
            Lights[Idx] := False;
        End_For;
    End_Loop;
End_Task;
```

Setup pane

The JetSym setup window displays the state of inputs and outputs:



Number	Description	Function
1	Present state of outputs	The outputs are set and reset one after another.
2	Present state of the output	When the input is set, the corresponding output is activated for 100 ms.

Flags

Introduction	Flags are one-bit operands. This means they can either have the value TRUE or FALSE.
Properties of user flags	▪ Global variables which are assigned to permanent addresses (%MX) ▪ Non-volatile ▪ Quantity: 256 ▪ Flag numbers: 0 ... 255
Properties of overlaid user flags	▪ Global variables which are assigned to permanent addresses (%MX) ▪ Non-volatile ▪ Overlaid by registers 1000000 through 1000055 ▪ Quantity: 1,792 ▪ Flag numbers: 256 ... 2047
Properties of special flags	▪ Global variables which are assigned to permanent addresses (%MX) ▪ When the operating system is launched, special flags are initialized using their default values. ▪ Quantity: 256 ▪ Flag numbers: 2048 ... 2303
Memory access	▪ By JetSym ▪ By the e-mail client ▪ By browser (via HTTP server) ▪ From HMIs ▪ From the application program

9.2 Numbering registers and I/Os for a JC-9xx

Introduction

Controllers and modules produced by Jetter AG offer a host of functions which can be accessed by the user via registers. A unique number is assigned to each register and each digital input or output.

Applying a register number

Register numbers are applied in the following cases:

- You want to read or write to a module register in the JetSym setup.
- You want to declare a module register a variable in the JetSym application program.
- You want to declare a module register a tag in JetViewSoft

Applying an I/O number

I/O numbers are applied in the following cases:

- You want to read from a digital input in the JetSym setup.
- You want to read or write to a digital output in the JetSym setup.
- You want to declare a digital input or output a variable in the JetSym application program.
- You want to declare a digital input or output a tag in JetViewSoft.

Contents

Registers and module registers

Definition - Module register

By means of module registers, process, configuration and diagnostic data can be read by the module, or written to the module. The module register number within the module is unique.

Definition - Registers

This way, you can access registers directly:

- In an application program
- In a JetSym setup pane
- In a visualization process

The register number within the system is unique.

Definition - Global Node Number

The Global Node Number (GNN) is an ID number ranging from 000 ... 199 which lets you address controllers and bus nodes. The Hardware Manager configures the GNN and assigns a unique and specific ID to each controller and bus node:

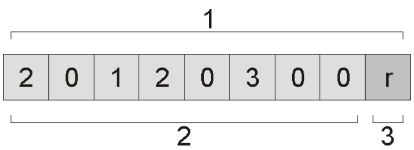
- In each project, GNN 000 is assigned to the controller by default.
- In the same project, the GNN assigned to bus nodes ranges from 001 ... 199.

Example - Module registers

Via module register 9 the OS version of a JX2-IO16 module can be accessed.

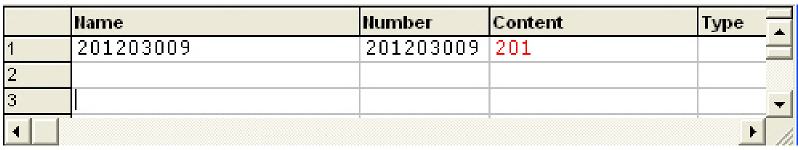
Example: Registers on the JX2 system bus

A JX2-IO16 module is connected with the JX2 system bus at its first position placed in the PCI slot 1 (module board 1) at the upper part of the system bus. At the JX2 system bus, the module number of this module is 2.



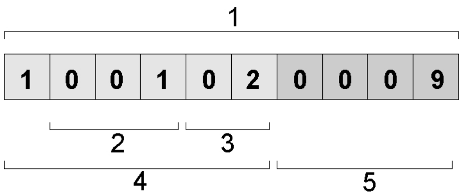
Number	Element	Description
1	Register number	Can be used directly
2	Register prefix	20120300: For the first JX2-I/O module connected to a JC-940MC controller.
3	Module register number	r = 9: OS version of the JX2-IO16

In the setup window of JetSym you can directly read out the operating system version 2.01 via register number 201203009.



Example: Registers on the Ethernet system bus

A JX3-AI4 module is connected to a bus node JX3-BN-ETH. The module number of the JX3 module is 2. The bus node has got the ID (GNN) 001.



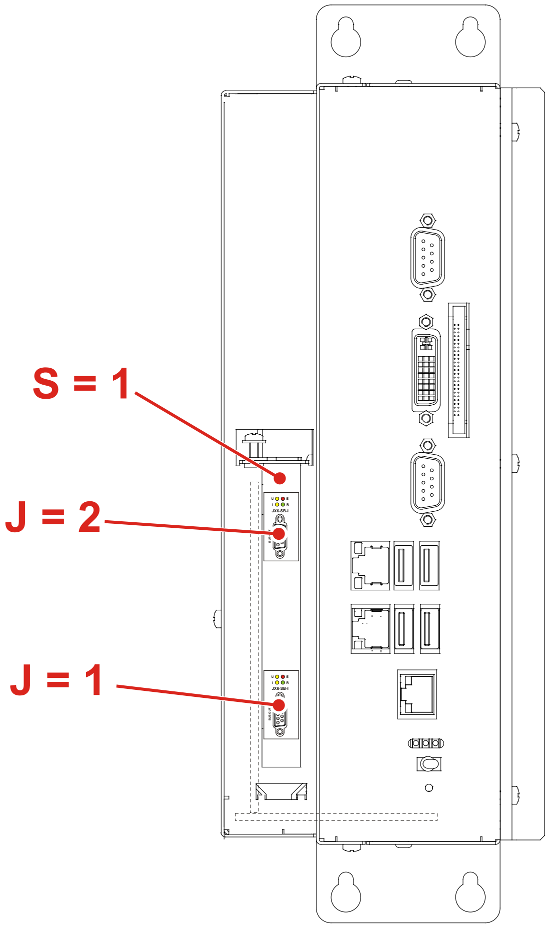
Number	Element	Description
1	Register number	Can be used directly
2	Bus node ID, GNN	001: ID of the first JX3-BN-ETH
3	Module number	02: The first JX3-I/O module connected to the JX3-BN-ETH
4	Register prefix	100102
5	Module register number	0009: OS version of the JX3-AI4

In the setup window of JetSym you can directly read out the operating system version 1.4.0.0 via register number 1001020009.

	Name	Number	Content	Type
1	1001020009	100102000	1.4.0.0	
2				

Slot numbering

Configuration



Code	Description
S	Slot number of the module board on the PCI bus
J	Number of the JX6-I/O board located on the module board

Numbering

This way you can determine the numbers:

- The counting direction of the module board slots on the PCI bus is right-to-left.
- The module board slot on the PCI bus directly next to the CPU is assigned number 1.
- The lower JX6-I/O board is assigned number 1.
- The upper JX6-I/O board is assigned number 2.

Register and I/O numbers of local JX6-I/O submodules

Register numbers for local JX6-I/O submodules

Register numbers for local JX6-I/O submodules connected to the JC-940MC consist of the following elements:

2	0	S	J	0	0	z	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zzz	Module register number	100 ... 999

I/O numbers for local JX6-I/O submodules

I/O numbers for local JX6-I/O submodules connected to the JC-940MC consist of the following elements:

2	0	S	J	0	0	1	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zz	Module-specific I/O number	01 ... 08

Register numbers of JX2 slave modules connected to the JX2 system bus

Slave module numbers of JX2 slave modules

This way you can determine the slave module numbers of intelligent JX2 slave modules and JetMoves at the JX2 system bus of the JC-940MC:

- Count the JX2 slave modules left-to-right, starting with 2.
- Leave out the power supply module JX2-PS1.
- Leave out non-intelligent JX2-I/O modules.

Register numbers for JX2 slave modules

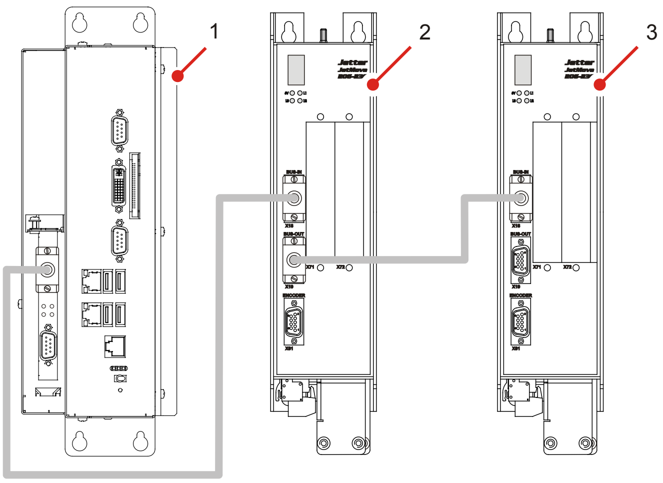
Register numbers for JX2 slave modules connected to the JX2 system bus of the JC-940MC consist of the following elements:

2	0	S	J	x	x	z	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	Slave module number + 10	12 ... 27
zzz	Module register number	000 ... 999

Example

Several JM-200 drives are connected to a JC-940MC controller.



Number	Module	Slave module number	Register
1	JC-940MC	1	Refer to documentation on JC-940MC
2	JM-206	2	201212zzz
3	JM-206	3	201213zzz

Register and I/O numbers of JX2- and JX3-I/O modules on the JX2 system bus

I/O module numbers of JX2- and JX3-I/O modules

This way you can determine the I/O module numbers of JX2- and JX3-I/O modules at the JX2 system bus of the JC-940MC:

- Count the JX2- and JX3-I/O modules left-to-right, starting with 2.
- Leave out the intelligent JX2 slave modules and JetMoves.
- Leave out the power supply module JX2-PS1.
- Count the JX3-BN-CAN modules left-to-right, starting with 33.

Register numbers for JX2- and JX3-I/O modules

Register numbers for JX2- and JX3-I/O modules connected to the JX2 system bus of the JC-940MC consist of the following elements:

2	0	S	J	0	3	x	x	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	I/O module number minus 2 Module number of the JX3-BN-CAN minus 2	00 ... 30 31 ... 61
z	Module register number	0 ... 9

I/O numbers for JX2- and JX3-I/O modules

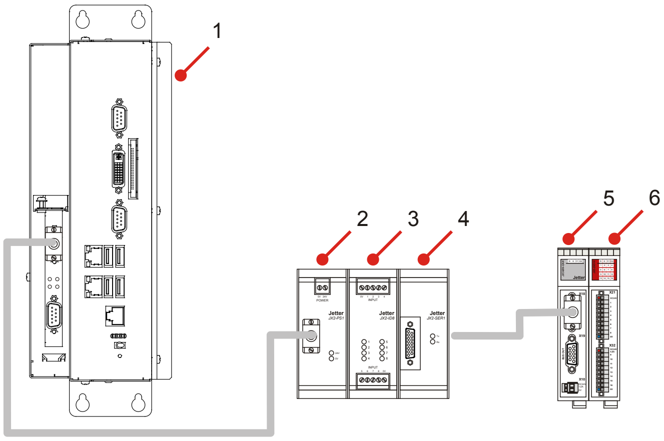
I/O numbers for JX2- and JX3-I/O modules connected to the JX2 system bus of the JC-940MC consist of the following elements:

2	0	S	J	0	x	x	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	Module-specific I/O number	02 ... 32
zz	Module-specific I/O number	1 ... 16

Example

Several JX2- and JX3-I/O modules are connected to a JC-940MC controller.



Number	Module	I/O module number	Register	I/O
1	JC-940MC	1	Refer to documentation on JC-940MC	
2	JX2-PS1	-	-	-
3	JX2-ID8	2	20120300z	2012002zz
4	JX2-SER1	3	20120301z	2012003zz
5	JX3-BN-CAN	33	20120331z	-
6	JX3-DIO16	4	20120302z	2012004zz

Register and I/O numbers of IP67-I/O modules on the JX2 system bus

I/O module numbers of IP67-I/O modules

This way you can determine the I/O module numbers of IP67-I/O modules at the JX2 system bus of the JC-940MC:

- The I/O module numbers you set by means of the addressing switch located on the module itself.
- LioN-S and LJX7-CSL modules are counted among IP67-I/O modules.

Register numbers for IP67-I/O modules

Register numbers for IP67-I/O modules connected to the JX2 system bus of a JC-3xx consist of the following elements:

2	0	S	J	0	3	x	x	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	I/O module number minus 2	00 ... 30
z	Module register number	0 ... 9

I/O numbers for IP67-I/O modules

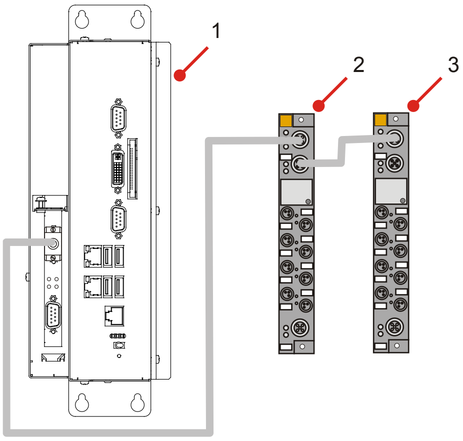
Register numbers for IP67-I/O modules connected to the JX2 system bus of a JC-940MC consist of the following elements:

2	0	S	J	0	x	x	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	Module-specific I/O module number	02 ... 32
zz	Module-specific I/O number	1 ... 16

Example

Several IP67-I/O modules are connected to a JC-940MC controller.



Number	Module	I/O module number	Register	I/O
1	JC-940MC	1	Refer to documentation on JC-940MC	
3	LioN-S	2	20120300z	2012002zz
4	LioN-S	3	20120301z	2012003zz

Registers and I/O numbers of CANopen® modules on the JX2 system bus

I/O module numbers of CANopen® modules

This way you can determine the I/O module numbers of CANopen® modules at the JX2 system bus of the JC-940MC:

- In most cases, the I/O module numbers correspond to the node ID of the CANopen® module.
- Exceptions: SMC EX120 and Lenze frequency inverter

Register numbers for CANopen® modules

Register numbers for CANopen® modules connected to the JX2 system bus of the JC-940MC consist of the following elements:

2	0	S	J	0	x	x	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	I/O module number	70 ... 79
z	Module register number	00 ... 99

I/O numbers for CANopen® modules

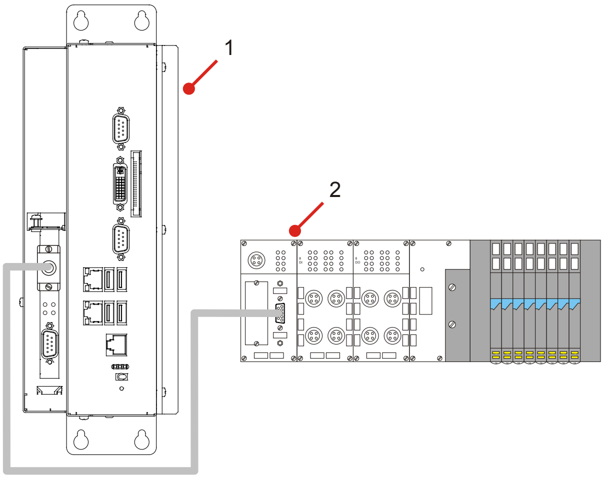
I/O numbers for CANopen® modules connected to the JX2 system bus of a JC-940MC consist of the following elements:

2	0	S	J	0	x	x	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O board (JX2 system bus) located on the module board	1 ... 2
xx	Module-specific I/O module number	70 ... 79
zz	Module-specific I/O number	1 ... 64

Example

A CANopen® module is connected to a JC-940MC controller.



Number	Module	I/O module number	Register	I/O
1	JC-940MC	1	Refer to documentation on JC-940MC	
2	Festo CPX	2	2012070zz	2012070zz

Register and I/O numbers of JX3 modules connected to a JX3-BN-ETH

Global Node Numbers on the Jetter Ethernet system bus of a JX3-BN-ETH

The JetSym Hardware Manager assigns Global Node Numbers to the bus node JX3-BN-ETH on the Jetter Ethernet system bus.

Register numbers for JX3 modules

The register number for JX3 modules at the Ethernet bus node consists of the following elements:

1	n	n	n	x	x	z	z	z	z

Element	Description	Value range
nnn	Global Node Number of a JX3-BN-ETH on the Ethernet system bus	001 ... 199
xx	Module number of the module within the JX3 station	02 ... 17
zzzz	Module register number	0000 ... 9999

I/O numbers for JX3 modules

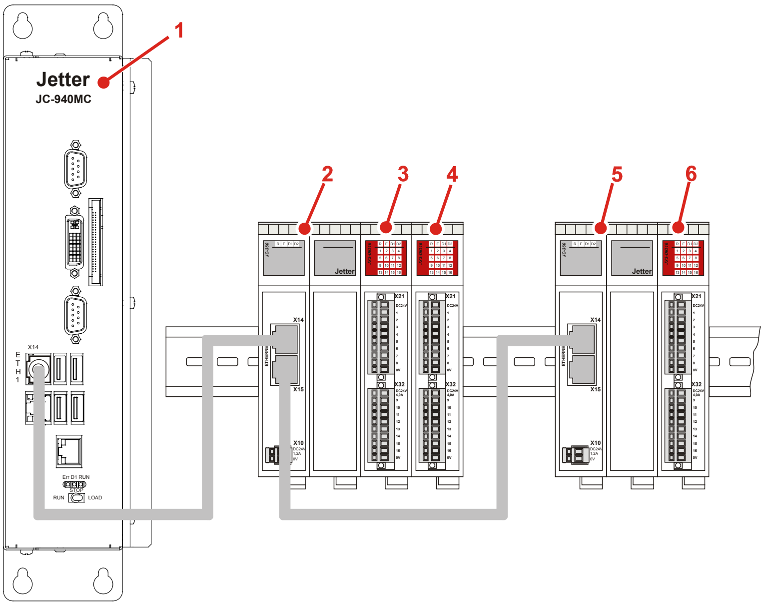
The I/O number for JX3 modules connected to an Ethernet bus node consists of the following elements:

1	n	n	n	0	1	x	x	z	z

Element	Description	Value range
nnn	Global Node Number of a JX3-BN-ETH on the Ethernet system bus	001 ... 199
xx	Module number of the module within the JX3 station	02 ... 17
zz	Module-specific I/O number	1 ... 16

Example

Two bus nodes JX3-BN-ETH are connected to a JC-940MC controller.



Number	Module	Module number	GNN	Register	I/O
1	JC-940MC	1	0	Refer to documentation on JC-940MC	
2	JX3-BN-ETH	-	1	Refer to documentation on JX3-BN-ETH	
3	JX3-DIO16	2		100102zzzz	10010102zz
4	JX3-DIO16	3	-	100103zzzz	10010103zz
5	JX3-BN-ETH	-	2	Refer to documentation on JX3-BN-ETH	
6	JX3-DIO16	2	-	100202zzzz	10020102zz

JX3 module register and I/O numbers from the JX3-BN-ETH view

Module number of a JX3 station

This way, you determine module numbers in a JX3 station:

- Count the JX3-I/O modules left-to-right, starting with 1.
- Leave out the power supply module JX3-PS1.

Register numbers for JX3 modules

From the perspective of the Ethernet bus node, the register numbers consist of the following elements:

1	0	0	x	x	z	z	z	z

Element	Description	Value range
xx	Module number of the module within the JX3 station	02 ... 17
zzzz	Module register number	0000 ... 9999

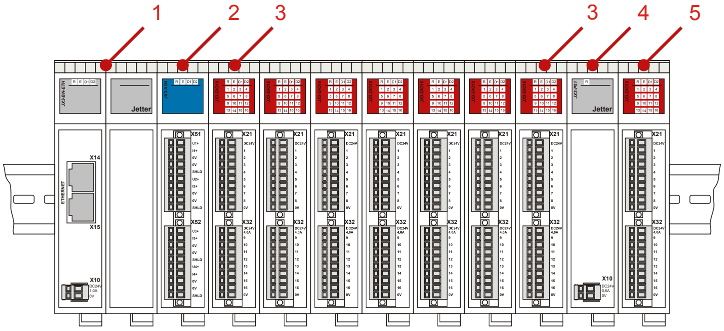
I/O numbers for JX3 modules

From the perspective of the Ethernet bus node, the I/O numbers consist of the following elements:

1	0	0	0	0	x	x	z	z

Element	Description	Value range
xx	Module number of the module within the JX3 station	02 ... 17
zz	Module-specific I/O number	1 ... 16

Example of a JX3 station at a JX3-BN-ETH



Number	Module	Module number	Register	I/O
1	JX3-BN-ETH	1	Refer to documentation on JX3-BN-ETH	
2	JX3-AO4	2	10002zzzz	1000002zz
3	JX3-DIO16	3 ff.	10003zzzz	1000003zz
4	JX3-PS1	-	-	-
5	JX3-DIO16	10	10010zzzz	1000010zz

9.3 Jetter Ethernet system bus

Introduction

Controllers and modules produced by Jetter AG offer a host of functions which can be accessed by the user via registers.

The Jetter Ethernet system bus has been designed for data exchange between controllers, bus nodes and communication modules via standard Ethernet.

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 the release numbers listed below.

Component	As of version
JC-340	V 1.09.0.00
JC-350	V 1.09.0.00
JC-360	V 1.09.0.00
JC-360MC	V 1.09.0.00
JC-940MC	V 1.00.0.50
JX3-BN-ETH	V 1.09.0.00
JX3-COM-EIPA	V 1.01.0.00
JetSym	V 4.3.0

Configuring the Jetter Ethernet system bus

Jetter AG advises the use of the JetSym Hardware Manager for configuring the Jetter Ethernet system bus.

Contents

9.3.1 Data exchange via Jetter Ethernet system bus

Introduction

This chapter covers data exchange via Jetter Ethernet system bus.

Contents

Data exchange

Introduction

This chapter describes the varieties and features of data exchange when applying the Jetter Ethernet system bus.

Various kinds of data exchange

If you want to access data of two remote peripheral modules, you can transmit the data to the controller in two ways:

- Publish/subscribe
- NetCopy

Publish/subscribe integrates process data registers into the controller's address space. This lets you address input and output data like local peripheral modules via address offset.

When programming, you have to decide whether to transmit data by publish/subscribe or by NetCopy.

Publish/subscribe is helpful in those cases:

- Variables that have to be transmitted quickly
- Small amounts of data
- Short response times for the remote system

NetCopy is helpful in those cases:

- Any other types of data transmission

Items that are not transmitted by publish/subscribe are internally resolved as NetCopy by JetSym.

Comparison

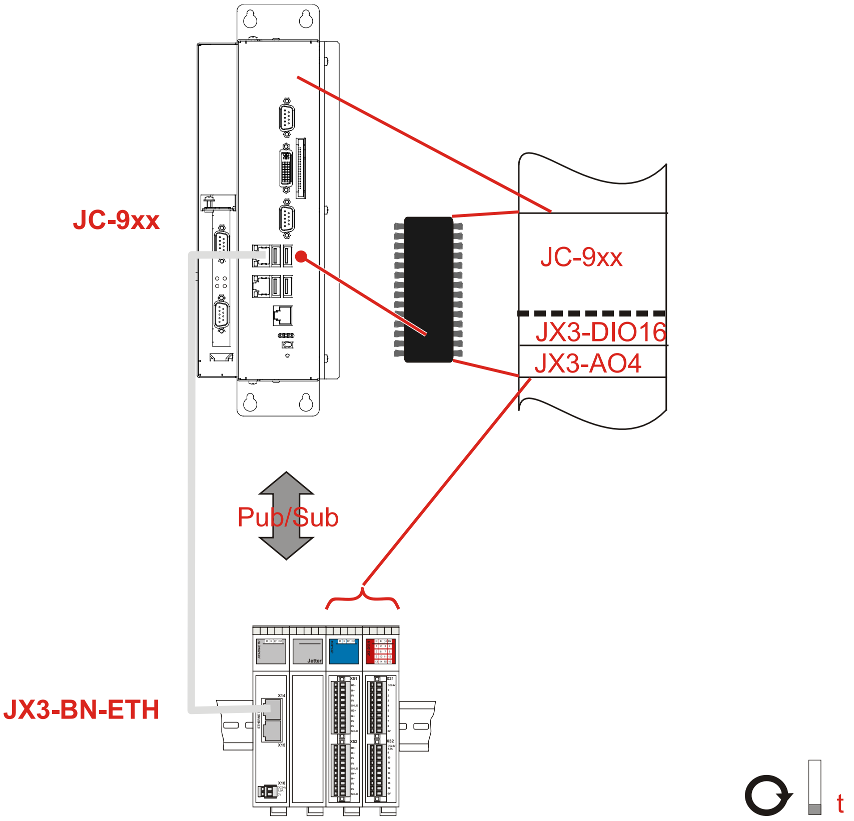
The Jetter Ethernet system bus lets you exchange data in the following way:

Data exchange by the following means:	Description
Publish/subscribe	**Type of data:** ■ Process data **Publish/subscribe** ■ Cyclic data exchange ■ Automatic data exchange ■ Very fast ■ Limited number of data
NetCopy ()	**Type of data:** ■ ■ Variables **Features of NetCopy ():** ■ Explicit data exchange ■ Data exchange by JetSym program ■ Not as fast as the process data
NetBitSetReg () **NetBitClearReg ()**	**Type of data:** ■ **Properties:** ■ Explicit data exchange ■ Data exchange by JetSym program ■ Not as fast as the process data
Indirect addressing	**Type of data:** ■ ■ Variables **Features of indirect addressing:** ■ Register numbers starting from 1 billion ■ A range of 200 registers is visible in the controller. ■ Access is also possible via variable destination window. ■ Internet resolution of the addresses by means of NetCopy

Features of publish/subscribe

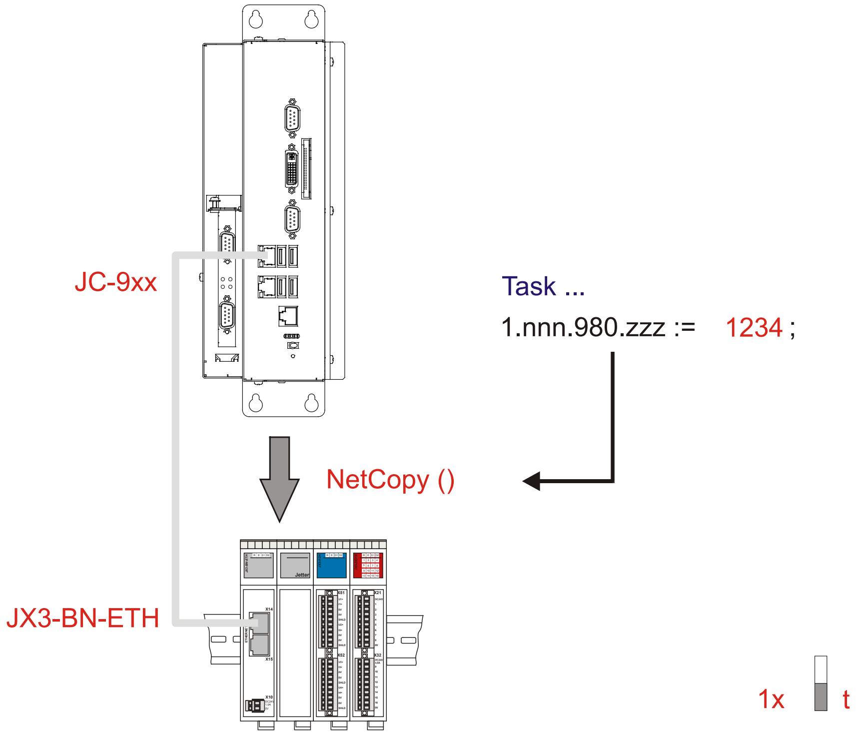
The *publish/subscribe* mechanism functions in the following way:

The remote modules are integrated into the controller's address space as process data registers. Publish/subscribe by default transmits important data, such as the analog value of a peripheral module, cyclically, . This means you need not make inquiries on any values.



Features of NetCopy ()

All data, except process data, are exchanged via NetCopy. JetSym internally resolves inquiries or variable and register assignments via NetCopy. This operation takes more time than exchange by means of process data.



Multicast in other networks

Please note that the Jetter Ethernet system bus operates with multicasts (multipoint connections). If you couple the Jetter Ethernet system bus with your local network, you have to filter out unwanted multicasts by a router.

Publish/subscribe operates with multicasts. You can sort multicasts of publish/subscribe in groups of 0 ... 254.

Related topics

- **RFC5771 http://www.rfc-archive.org/getrfc.php?rfc=5771**

Register access

Introduction

This chapter describes register access via GNNs (Global Node Numbers) when applying the Jetter Ethernet system bus.

Addressing via GNNs

To reach remote peripherals, addressing via GNNs is needed. 200 network nodes can be addressed in the Jetter Ethernet system bus. The Hardware Manager of JetSym assigns an unambiguous GNN to each controller and each remote network node in the Jetter Ethernet system bus.

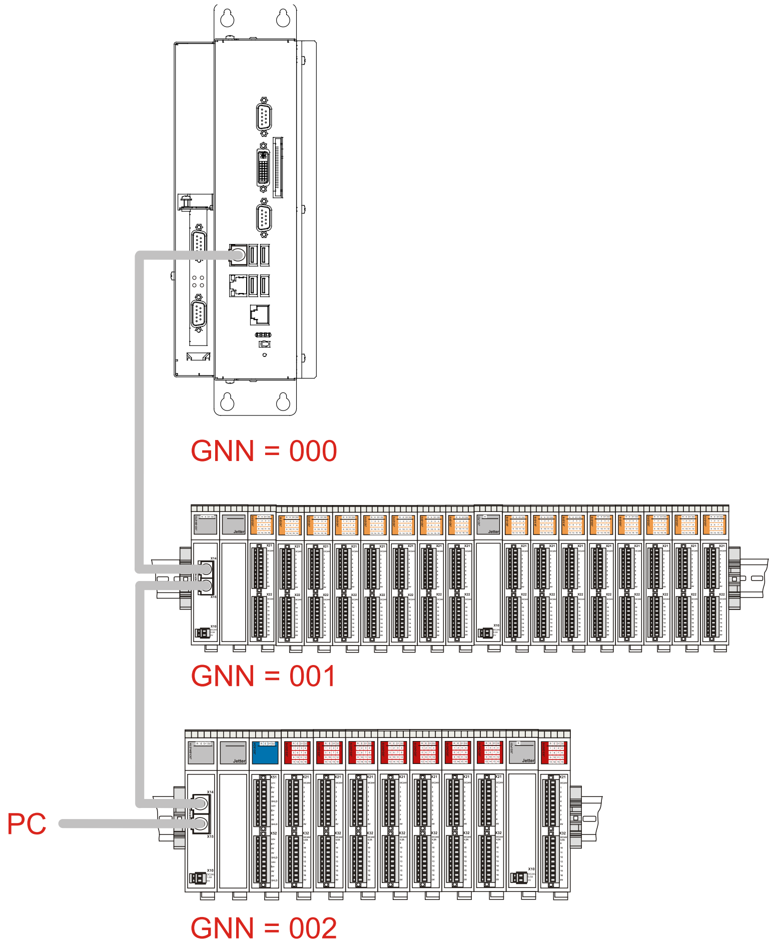
By default, the controller has got the GNN 000. This corresponds to nnn = 000.

The register array as of 1 billion is structured in the following way:

Register array	Description
1nnn020000 ... 1nnn179999	JX3 peripheral modules at the JX3-BN-ETH - **Register addressing**
1nnn980000 ... 1nnn980199	**Indirect addressing** of remote modules via register arrays, resolved by NetCopy
1nnn990000 ... 1nnn999999	**Indirect addressing** with variable **destination window**; resolved by NetCopy

Networking example

The following illustration shows networking of a possible JX3 system. Each network node is assigned a GNN.



Publish/subscribe

Introduction

Publish/subscribe has been developed for automatic data exchange in the JX3 system. Enter the process data which the local controller is to exchange automatically with remote network nodes in the Hardware Manager.

Features of publish/subscribe

If you add modules and make them known to the Hardware Manager, it will automatically generate the process data belonging to these modules as publish/subscribe variables.

Features of publish/subscribe

Parameter	Value	Description
Number of network nodes	01 ... 199	200 network nodes max.: They are entered into the Hardware Manager by their name and as GNN
Number of process variables per network node	64	64 process variables max: This corresponds to an 256 byte of process data
Cycle time	1 ... 6 ms	Default: 2 ms

Network nodes are the controller, the communication modules and the bus nodes.

For details on characteristic features of publish/subscribe, please turn to chapter *Hardware Manager*.

Related topics

- **Hardware Manager** (see page 207)

NetCopy

Introduction

The NetCopy command is a versatile tool for data exchange between Jetter products via Ethernet.

The NetCopy command lets you copy the following data:

- Register values
- Values of register blocks
- Variable values
- Values of variable blocks

Access via NetCopy

NetCopy functions with the following nodes:

- Controllers
- Bus nodes
- Communication modules

To access other nodes, use the command NetCopy as follows:

If ...	... then ...
you wish to copy data from the controller to another node,	use the command ■ NetCopyRegToReg ■ NetCopyVarToReg
you wish to copy data from another node to the controller,	use the command ■ NetCopyRegFromReg ■ NetCopyVarFromReg

Parameters of the NetCopy functions

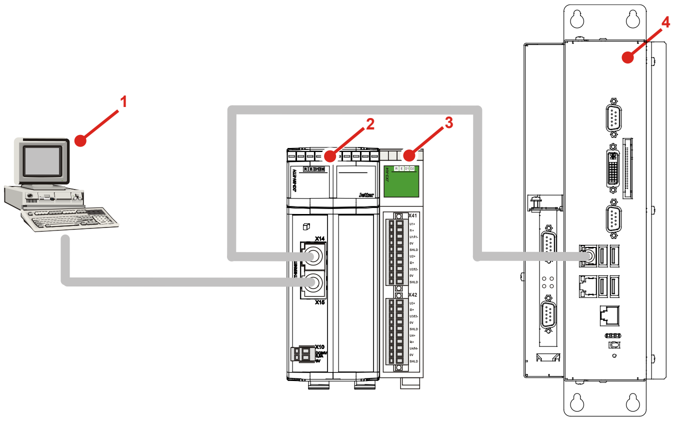
For detailed information on the parameters, refer to the JetSym help.

NetCopy: Example: NetCopy and the bus node

As you can see in the following illustration, a controller JC-9xx is connected to a PC. The bus node JX3-BN-ETH is connected to a peripheral module JX3-AI4.

When the Hardware Manager detects the peripheral module JX3-AI4, it creates publish/subscribe variables. You may use these variables. The Hardware Manager sets the status register and the measured values as process data.

This example describes how to access a module register of the peripheral module.



Number	Part	Description
1	PC	PC with JetSym
2	JX3-BN-ETH	Bus nodes
3	JX3-AI4	Peripheral module with analog inputs
4	JC-9xx	Controller

Task

Status register of input 1 is to be retrieved using NetCopy.

Solution

The command NetCopy copies the content of MR 1100 on the JX3-AI4 module into a variable.

The register number of the peripheral module is seen from the perspective of the JX3-BN-ETH:

1	0	0	x	x	z	z	z	z

with

- First module at the JX3-BN-ETH
- zzzz = 1100: Status registers of the JX3-AI4

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

```
Var
    n_Get : Int;
End_Var;

Task t_FromReg Autorun

// Content of MR 1100 is now in variable n_Get
Loop
// Required for pub/sub
End_Loop;

End_Task;
```

NetBitSetReg and NetBitClearReg

Introduction

The NetBit command is an all-purpose tool to set or clear register bits of Jetter products The Jetter products are interconnected via an Ethernet network.

Access via NetBit

NetBit functions with the following nodes:

- Controllers
- Bus node
- Communication modules

To access other nodes, use the command NetBit as follows:

If ...	... then ...
you wish to set register bits for another node,	use the command ▪ NetBitSetReg
you wish to clear register bits of another node,	use the command ▪ NetBitClearReg

Parameters of the NetBit functions

For detailed information on the parameters, refer to the JetSym help.

Benefit of applying NetBit functions

NetBit functions let you both set and clear bits in one go.

Without the NetBit functions, only the folllowing actions can be taken:

- A NetCopy command lets you copy the register value from the remote node to the local controller.
- Changing the state of the bits on the local controller as desired
- Another NetCopy command lets you copy the register value to the remote node again.

For this, several commands are required. Thus, a register value may be changed during this action by an application program running on the remote controller. The second NetCopy command will then overwrite this value again. There is an undefined data condition, which is prevented by the NetBit functions.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

```
Var
    bi_BitSet, bi_BitClear   : Bool;
    n_IPaddr                 : Int;
    n_PortNr                 : Int;
    n_BitMask                : Int;
    n_RegNum                 : Int;
    n_RegValue               : Int;
    n_Result                 : Int;
End_Var;

Task t_BitSetClear Autorun

    n_IPaddr      := IP#192.168.10.209;
    n_PortNr      := 50000;

    // Bit mask
     The register bit remains as it was
     The register bit is set at NetBitSetReg
     The register bit is cleared at NetBitClearReg
    n_BitMask     := 0x00000040;

    // Application registers
    // Range: R 1000000 ... 1119999
    n_RegNum      := 1000100;

    // JetSym setup lets you change the bit mask and the
    // register number
    Loop
        bi_BitSet   := FALSE;
        bi_BitClear := FALSE;
    // JetSym setup lets you set the state of the flags bi_BitSet
    // or bi_BitClear to TRUE
    // n_RegValue contains the value of register nRegNum
    When bi_BitSet Then
        n_Result := NetBitSetReg(n_IPaddr,n_RegNum,n_BitMask,
                                n_RegValue,3,n_PortNr,0);
        Else_When bi_BitClear Then
        n_Result := NetBitClearReg(n_IPaddr,n_RegNum,n_BitMask,
                                n_RegValue,3,n_PortNr,0);
        End_When;
    End_Loop;
End_Task;
```

Indirect addressing of remote modules

Introduction

Indirect addressing lets you access module registers of the controller and enter addresses of remote devices into a register array of the controller. The controller internally resolves indirect addressing via JetSym. This way, it permits read and write access to register contents of the remote network nodes.

Register overview

Overview of the registers allowing indirect addressing of remote nodes:

Registers	Value range	Characteristics
235 000 + GNN	235000 ... 235199	Register array for **IP addresses**
235400 + GNN	235400 ... 235599	Register array of **port number** registers
236000 + ppp	236000 ... 236199	Register array for the **Index**
1nnn980ppp	1nnn980000 ... 1nnn980199	Register array for the **Content**

with

- nnn = GNN in the array 000 ... 199
- ppp in the array 000 ... 199

Addressing scheme

In order to access network nodes via register number, there are three register arrays of 200 registers each available in the JC-9xx controller series. The three register arrays have got room for variables, by which you have at the moment got read and write access to 200 contents of remote network nodes.

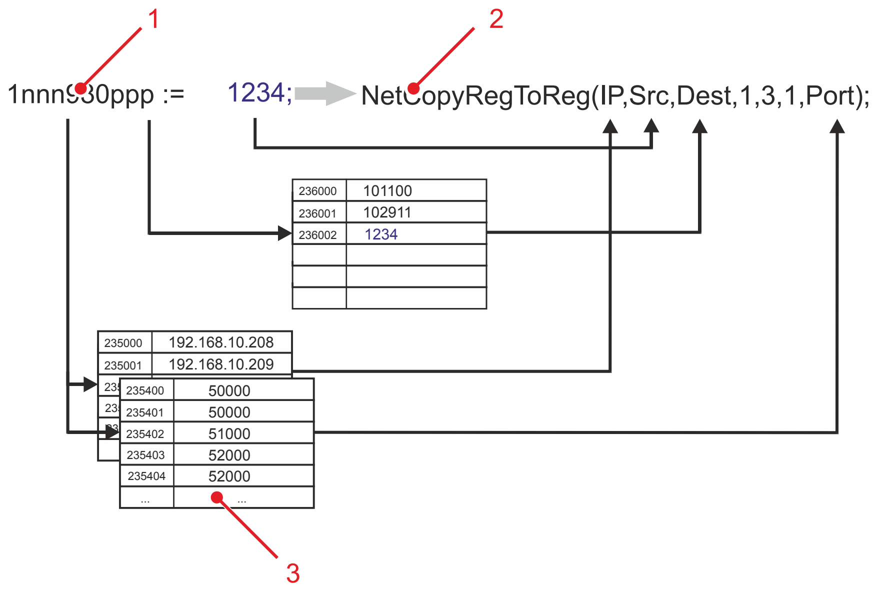
The addressing scheme for contents of indirect addressing is as follows:

$$\overbrace{\mathbf{1}\;\underbrace{n\;n\;n}_{2}\;\underbrace{\mathbf{9}\;\mathbf{8}\;\mathbf{0}}_{3}\;\underbrace{p\;p\;p}_{4}}^{1}$$

Number	Element	Description
1	Register number	Can be used directly
2	GNN	nnn = 001 ... 199: Global Node Number (GNN) of the remote network node
3	Designation:	980 indicates indirect addressing of the register number
4	Index for contents	Array: ppp = 0 ... 199

Internal resolution of indirect addressing

JetSym resolves inquiries or assignments at a register address of numbers greater than 1 billion via NetCopy. Before assigning, you must enter the IP address, port number and the index into the respective register arrays to enable JetSym to correctly resolve the command. When the register arrays of remote network nodes have been defined, resolution via Hardware Manager is carried out.



Number	Element	Description
1	Indirect address in a JetSym program or a JetSym setup pane	The address contains the GNN of the remote network node (nnn = 000 ... 199) and the index (ppp = 000 ... 199, here 002).
2	NetCopy command	Here: NetCopyRegToReg
3	Variable register tables of 200 entries max.	Variable register array for the following items: ■ IP address ■ Port number ■ Index to contents

Action

If you want to enter a value into the register of a remote network node using register addresses as of 1 billion, proceed as follows:

Step	Action
1	Enter the **IP address** of the remote network node into register **235000 + GNN**. Value range of GNN: 0 ... 199
2	Enter the **port number** into register **235400 + GNN**. Value range of GNN: 0 ... 199
3	Enter the required **register number** of the remote network node into **register 236000 + ppp**. **Result:** Now you can access the value via register **1nnn980ppp**. Value range of GNN = nnn: 000 ... 199 Value range of ppp: 000 ... 199

This configuration lets you indirectly access via 200 controller registers all module registers of the remote network node.

Example

A controller of the JC-9xx series is connected with a JX3-BN-ETH bus node.

Task:

You want to take the following action:

- Read the IP address from the local module register of the JX3-BN-ETH bus node
- Read the seconds of the real-time clock from the local module register of the JX3-BN-ETH bus node

Solution:

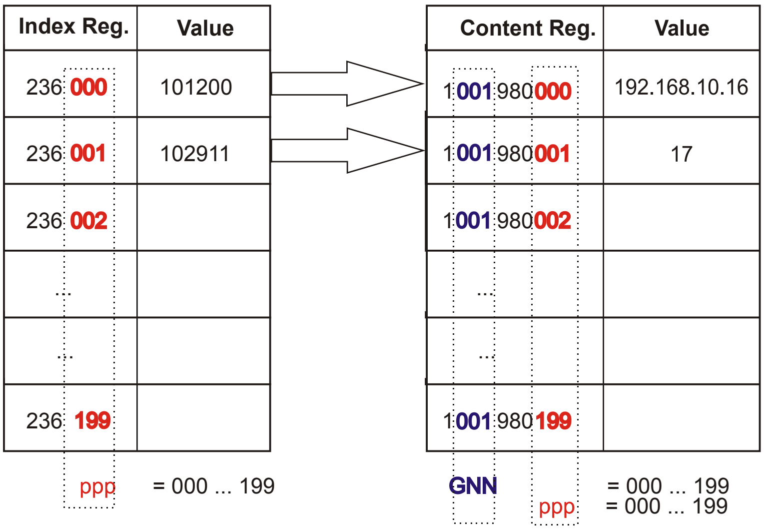
The IP address of the JX3-BN-ETH is contained in MR 101200. The seconds of the real-time clock of the JX3-BN-ETH are contained in MR 102911.

Using the Hardware Manager, you have configured the controller and the bus node. The controller has got GNN = 000. The bus node JX3-BN-ETH has got GNN = 001.

In the controller, addressing looks as follows:

- Enter the number of the local module register *IP address* from the JX3-BN-ETH to R 236000.
  From the quick reference, you know that the IP address of the JX3-BN-ETH can be read from MR 101200.
- In case of read access to R 1001980000, read now the IP address of the JX3-BN-ETH.
- Enter the number of the local module register *seconds of the real-time clock* from the JX3-BN-ETH to R 236001.
- In case of read access to R 1001980001, read now the IP seconds of the real-time clock on the JX3-BN-ETH.

- Write access to R 1001980ppp is also possible.



Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

```
Var
    // Controller with GNN = 000
    // Bus node with GNN = 001 - see Hardware Manager
    IP_AdrContr:          Int At %VL 235000;
    IP_BusNode:           Int At %VL 235001;
    PortNr_Contr:         Int At %VL 235400;
    PortNr_BusNode:       Int At %VL 235401;
    IndexReg1:            Int At %VL 236000;
    IndexReg2:            Int At %VL 236001;
    ContentReg1:          Int At %VL 1001980000;
    ContentReg2:          Int At %VL 1001980001;
End_Var;

Task JX3_BN_ETH_Access Autorun
    // Initialization
    IP_AdrContr     := IP#192.168.10.10;
    IP_BusNode      := IP#192.168.10.15;
    PortNr_Contr    := 50000;
    PortNr_BusNode  := 50000;

    // Access to IP addresses of the JX3-BN-ETH (R101200):
    IndexReg1 := 101200;
    // Now, index register 1 points to the register "IP address"
    // of the JX3-BN-ETH
```

```
    Trace('IP von JX3-BN-ETH: ' + InttoStr(Contentreg1));

    // Access to the seconds of the clock of the JX3-BN-ETH
(R102911):
    IndexReg2 := 102911;
    // Now, index register 2 points to the register
    // "Seconds of the real-time clock"
    Trace('Seconds:  + InttoStr(Contentreg2));

    Loop
        // Show the values by the Trace function
            Trace(' IP vom JX3-BN-ETH: ' + InttoStr(Contentreg1));
            Trace(' Seconds of the JX3-BN-ETH:
                InttoStr(Contentreg2) + '$N');
        Delay(T#900ms);
    End_Loop;

End_Task;
```

File *ModConfig.da*

When you download the configuration files, the Hardware Manager transfers the file **ModConfig.da** to the controller.

The OS of the controller loads this file when the controller is energized or when the corresponding command is automatically issued by the Hardware Manager after download.

The file **ModConfig.da** lists registers with their corresponding values. The OS enters the corresponding values into these registers.

This file also holds the IP addresses (register 235000 + GNN) and port numbers (register 235400 + GNN) of the nodes on the network.

It is no longer required to enter values into registers via application program.

Related topics

- **NetCopy** (see page 193)

Indirect addressing with variable destination window

Introduction

Indirect addressing also allows for a variable destination window. You shift the register array of 10,000 registers of the remote network nodes by an offset by entering a value into R 272702 of the remote network nodes.

Register overview

Overview of the registers allowing indirect addressing with variable destination window:

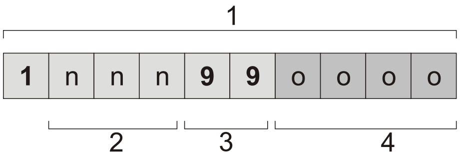
Register	Value range	Characteristics
235000 + GNN	235000 ... 235199	Register array for **IP addresses**
235400 + GNN	235400 ... 235599	Register array of **port number** registers
1nnn99oooo	1nnn990000 ... 1nnn999999	**Register content** of a remote network node; The register is in the variable destination window which consists of 10,000 registers.
R 272702	0 ... 2,147,483,647	Variable destination window: The destination window is a register array of a remote network node. This destination window is shifted by this **offset**.

with

- nnn = GNN in the array 000 ... 199
- oooo in the array 0 ... 9,999

Addressing scheme

The addressing scheme for contents of indirect addressing with variable destination window is as follows:



Number	Element	Description
1	Register number	Can be used directly
2	GNN	nnn = 001 ... 199: Global Node Number (GNN) of the remote network node
3	Designation: **99**	99 indicates indirect addressing of the register number with offset.
4	Index to a register	oooo: 0 ... 9,999: The register is in the variable destination window which consists of 10,000 registers.

Steps to take for indirect addressing with destination window

To use register addresses starting from 1 billion with variable destination window (offset), proceed as follows:

Step	Action
1	Enter the **IP address** of the remote network node into register **235000 + GNN**. Value range of GNN: 0 ... 199
2	Enter the **port number** into register **235400 + GNN**. Value range of GNN: 0 ... 199
3	Set the base address of the **destination window:** Enter a value into R 272702 of the remote network node.
⇨	Now, registers **1nnn990000 ... 1nnn999999** let you access the value.

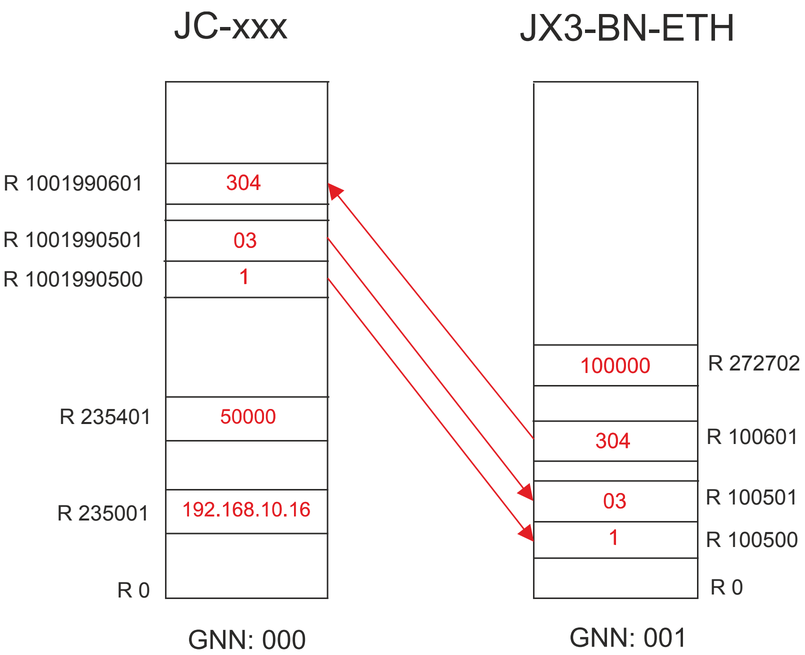
Example

A JetControl is to read a register value from a JX3-BN-ETH. Control system and bus node are interconnected via the Jetter Ethernet system bus.

There are JX3 modules connected to the JX3-BN-ETH, such as a JX3-AO4 of module number 03.

By entering value 100000 into R 272702 of the JX3-BN-ETH, you get read access to the EDS of the connected JX3 modules.

In this example, the module code of the JX3-AO4 is to be read.



Reading is carried out in three steps:

Step	Action
1	Enter value 1 for a JX3 module into R 1001990500.
2	Enter module number 03 into R 1001990501.
3	Read module code 304 for JX3-AO4 from R 1001990601.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

```
Var
    // Controller with GNN = 000
    // Bus node with GNN = 001
    IP_AdrContr        : Int At %VL 235000;
    IP_BusNode         : Int At %VL 235001;
    PortNr_Contr       : Int At %VL 235400;
    PortNr_BusNode     : Int At %VL 235401;
    IndexReg           : Int At %VL 236000;
    ContentReg         : Int At %VL 1001980000;
    Target_Window_500 : Int At %VL 1001990500;
    Target_Window_501 : Int At %VL 1001990501;
    Target_Window_601 : Int At %VL 1001990601;
    Module_code          : Int;
End_Var;

Task EDS_JX3_Modules Autorun
    // Initialization
    IP_AdrContr    := IP#192.168.10.15;
    IP_BusNode     := IP#192.168.10.16;
    PortNr_Contr   := 50000;
    PortNr_BusNode := 50000;

    // Setting a basic address of the destination window on the
JX3-BN-ETH
    // (R 272702):
    IndexReg    := 272702;
    ContentReg  := 100000;
        // Selecting an EDS of a JX3 module
        // connected to a JX3-BN-ETH
    Target_Window_500   := 1;
    Target_Window_501   := 3;
        // Reading the module code
    Module_code  := Target_Window_601;

    Loop
    // ...
    End_Loop;

End_Task;
```

File *ModConfig.da*

When you download the configuration files, the Hardware Manager transfers the file **ModConfig.da** to the controller.

The OS of the controller loads this file when the controller is energized or when the corresponding command is automatically issued by the Hardware Manager after download.

The file **ModConfig.da** lists registers with their corresponding values. The OS enters the corresponding values into these registers.

This file also holds the IP addresses (register 235000 + GNN) and port numbers (register 235400 + GNN) of the nodes on the network.

It is no longer required to enter values into registers via application program.

9.3.2 Hardware Manager

Introduction

The Hardware Manager lets you easily configure the peripheral devices. If possible, always use the Hardware Manager that is part of JetSym. Making configurations by hand is complicated and prone to errors.

Contents

Hardware Manager

Hardware Manager

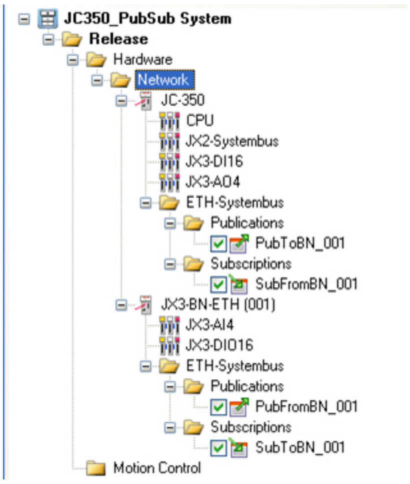
The Hardware Manager manages all connected hardware components.

The Hardware Manager assists you in the following aspects:

- Engineering and configuring control systems and bus nodes
- Engineering modules and axes at the JX2 system bus and configuring axes at the JX2 system bus
- Engineering JX3 modules at a JX3-BN-ETH and a JC-3xx
- Engineering and configuring Ethernet axes
- Engineering an axis group (path group and technology group)
- Configuring a path group
- Configuring technology group

Launching the Hardware Manager

For launching the Hardware Manager, klick, in JetSym, the tab **Hardware**. As an alternative, launch the Hardware Manager via keys **Alt + 5**.



Related topics

- **Ethernet system bus** (see page 184)

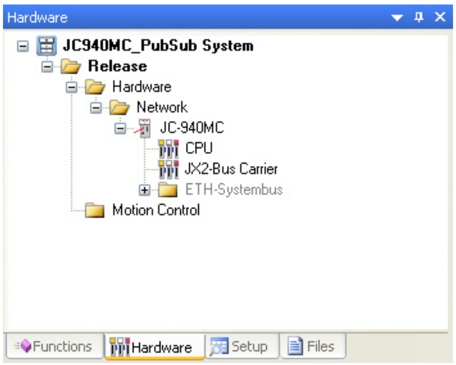
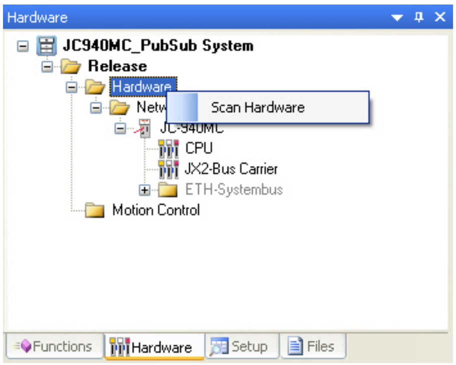
Configuring the hardware

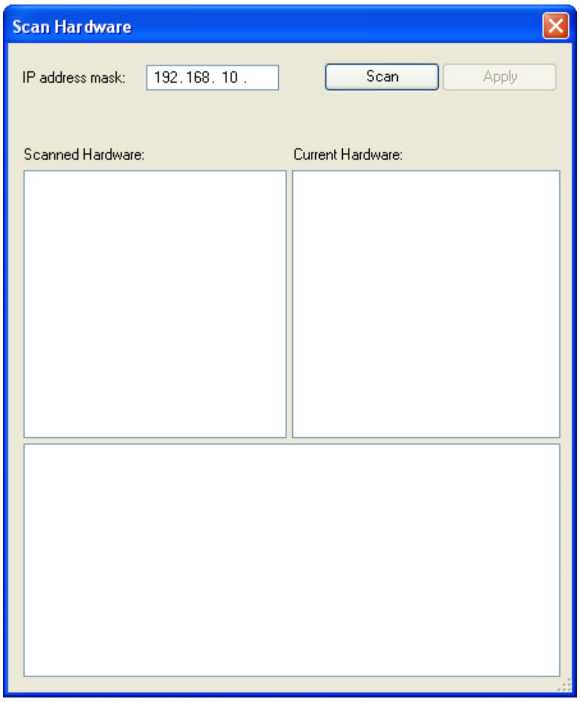
Introduction

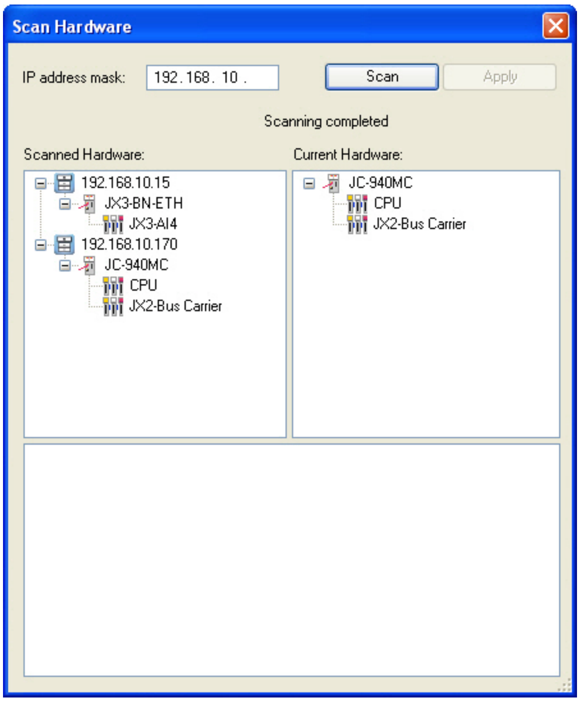
JetSym Hardware Manager lets you register your hardware in a tree structure. Information on detected hardware components is stored to the file **project_name.hardware** along with project data. You can change the hardware any time. For this, update the tree structure in the Hardware Manager.

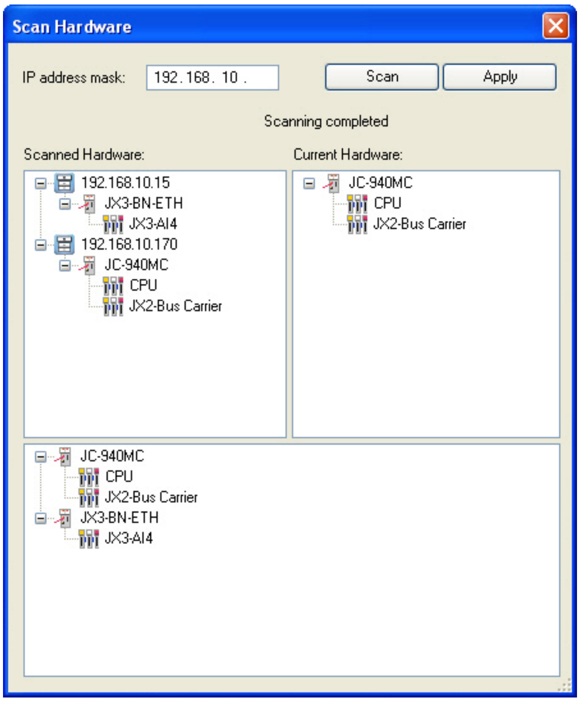
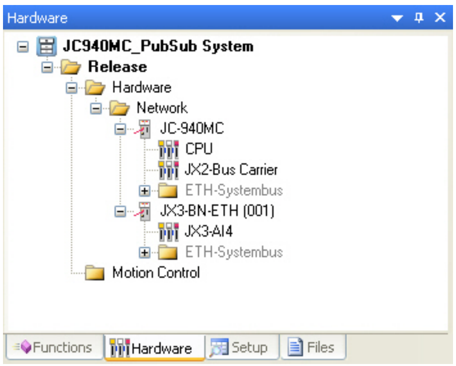
Configuring the hardware

To detect and configure all hardware devices connected to the controller, take the following steps in JetSym:

Step	Action
1	In JetSym, start the Hardware Manager by clicking on the tab **Hardware** with the mouse or by pressing **Alt + 5** on your keyboard. **Result**: 
2	Connect the following Jetter AG products with the PC via Ethernet system bus: ■ Controller ■ Bus nodes ■ Peripheral modules ■ Axes ■ HMIs
3	Energize the power supply.
4	In Hardware Manager, click on the folder **Hardware**, right-click **Scan Hardware**. 

Step	Action
5	Enter an IP address mask. A hardware scan applies to the whole IP subnet. Therefore, you have to enter at least the first three elements of the IP address. In this example, the IP address of the controller JC-9xx is 192.168.10.170. To detect the control systems and all bus nodes, enter 192.168.10. 
6	Click the button **Scan**.

Step	Action
⇨	The Hardware Manager scans the Jetter Ethernet system bus and compares the scanned hardware with the really set hardware. 
7	In the window **Scanned Hardware**, click the name of the controller. In this example, it is JC-940MC.
⇨	The Hardware Manager has the tree of the controller JC-940MC displayed in the bottom window.
8	Drag the entire tree of the JX3-BN-ETH into the lower window by Drag and Drop.

Step	Action
⇨	The Hardware Manager has the tree of the controller JX3-BN-ETH displayed in the bottom window. 
9	Click the button **Submit**.
⇨	The window closes. The Hardware Manager has taken over the hardware parameters. 
10	Check the result of the automatic hardware scan.

Advice on publish/subscribe

When testing peripheral modules, observe the following rule:

- If you change the configuration, always write the changed configuration to the connected devices by clicking **Download all Configurations**.
- To maintain communication via the Ethernet system bus, there must be an active minimum program running in JetSym.
  The setup pane displays correct data, for example, if a minimum program is running in the controller.
- If you write a minimum program without endless loop, publish/subscribe access does not work properly. The task is run through once and then stopped by TaskBreak.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

```
Task Main Autorun
    // Do not forget!
    Loop
    // Do not do anything
    // Only for testing purposes: Please refer to changes made in
JetSym setup
    End_Loop;
    //...
End_Task;
```

File *ModConfig.da*

When you download the configuration files, the Hardware Manager transfers the file **ModConfig.da** to the controller.

The OS of the controller loads this file when the controller is energized or when the corresponding command is automatically issued by the Hardware Manager after download.

The file **ModConfig.da** lists registers with their corresponding values. The OS enters the corresponding values into these registers.

This file also holds the IP addresses (register 235000 + GNN) and port numbers (register 235400 + GNN) of the nodes on the network.

It is no longer required to enter values into registers via application program.

Publish/subscribe - Functioning principle

Introduction

This chapter covers the functioning principle of the publish/subscribe mechanisms. The publish/subscribe feature lets you exchange process data over Jetter's Ethernet system bus. Each JC-9xx controller, as well as each bus node JX3-BN-ETH is able to publish and subscribe data.

Detected peripheral devices

If you connect peripheral devices to a controller JC-9xx or to a bus node JX3-BN-ETH, the Hardware Manager detects the peripheral devices. The Hardware Manager sets features of peripheral Jetter modules as process data.

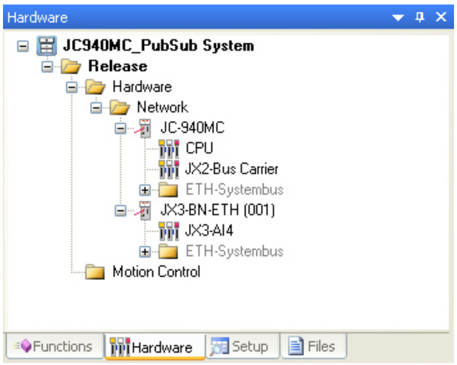
Some of them are:

- Status registers of the peripheral module
- Command registers of the peripheral module
- Error registers of the peripheral module
- Input data for peripheral modules with analog or digital inputs
- Output data for peripheral modules with analog or digital outputs

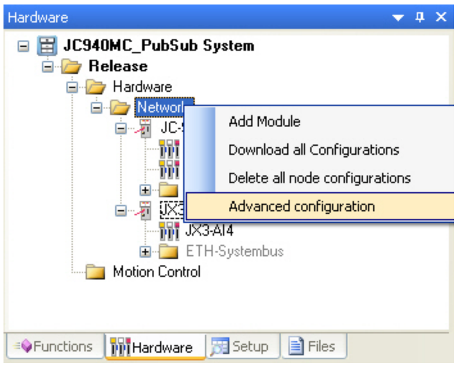
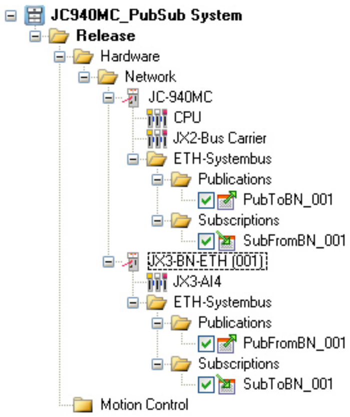
Advanced settings

If you want to access the publish/subscribe settings of controller and bus node, take the following steps:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>In JetSym, start the Hardware Manager by clicking on the tab Hardware with the mouse or by pressing Alt + 5 on your keyboard.</td></tr>
<tr><td>⇨</td><td>The Hardware Manager shows the hardware components in a tree structure.



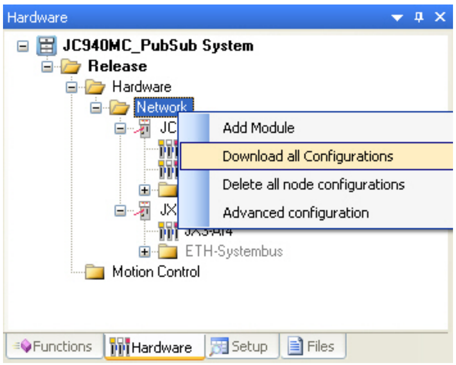
</td></tr>
<tr><td>2</td><td>To enable the publish/subscribe feature, right click the folder Network.</td></tr>
</table>

Step	Action
3	Select **Advanced configuration**. 
⇨	In the hardware tree below controllers and bus nodes, JetSym automatically creates a folder, which, in this case, is named **ETH-Systembus**. This folder contains two subfolders named **Publications** and **Subscriptions**. For each controller and each device used, the Hardware Manager creates a **Publication** and a **Subscription** item by default. Publications and subscriptions communicate with each other. 

Sending changes to controller and bus node

Once you have configured the hardware according to your requirements, you have to send the current configuration to the controller and the other connected devices. The Hardware Manager transmits the updated configuration to all bus nodes and controllers in the Jetter Ethernet system bus.

Download the current configuration to the controller by clicking the file **Network** and right-click on the menu item **Download All Configurations**



Result:
The controller and all further bus nodes have received the configuration data.

Internal administration of publish/subscribe

If you make any changes to the hardware, you have to send these changes by **Download All Configurations** to the connected hardware via Hardware Manager.

If you click in Hardware Manager **Download All Configurations,** the following happens:
In the project, JetSym has created a directory, e.g. Release, below the active configuration. This directory contains the following files:

- ModConfig.da
- ModConfig_BN_001.da
- NetConsistency.xml
- Publisher.pub
- Publisher_BN_001.pub
- Subscriber.sub
- Subscriber_BN_001.sub

The following files are copied to the local controller: Normally, the controller has got GNN 000.

- ModConfig.da
- NetConsistency.xml

The following files are copied to the bus node:

- Publisher.pub
- Subscriber.sub
- ModConfig.da

Recommendation:

Jetter AG advises the use of JetSym Hardware Manager for this task. Experienced users may also transfer these files via FTP, e.g. when the bus node has been replaced.

Important Note!

Do not delete these files. They are required by JetSym, the controller, and the bus nodes.

If you delete, for instance, the file **ModConfig_BN_001.da**, you are no longer able to access remote I/Os.

Publish/subscribe - Configuration

Publish/subscribe identification (pub ID, sub ID):

An **ID** is assigned to each publication and subscription. The range of 0 ... 9,999,999 is reserved for user-defined publication and subscription IDs. JetSym uses the range starting from 10,000,000.

The structure of a publication ID is **10.nnn.bcc**, where:

- **nnn** = number of the bus node
  nnn is the Global Node Number (GNN).
  Value range: 001 ... 199; normally, 000 is assigned to the local controller.
- **b** = Sender identification (0 = controller, 1 = bus node [also for controllers used as I/O expansion], etc.)
- **cc** = consecutive number of the publication

Publication name:

Besides the ID, a name is assigned to each publication. The default name of a publication consists of the following elements:

Name: Pub<From|To><bus node name>

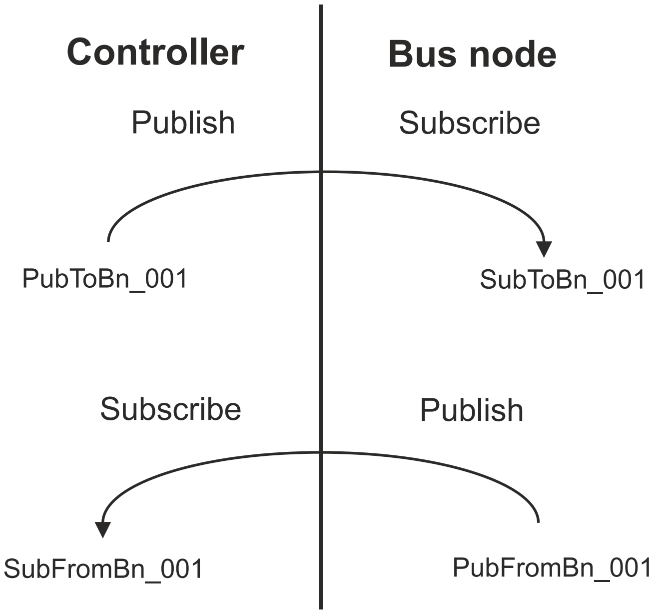
- "From" for publication of input data from the bus node
- "To" for publication of output data to the bus node

Subscription name:

Besides the ID, a name is assigned to each subscription. The default name of a subscription consists of the following elements:

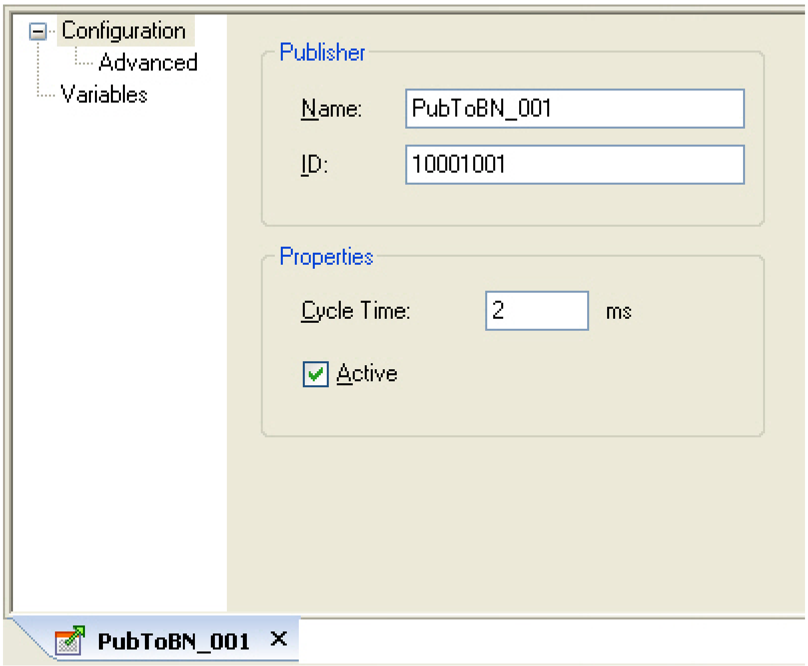
Name: Sub<From|To><bus node name>

- "From" for subscription to input data from the bus node by the controller
- "To" for subscription to output data by the bus node



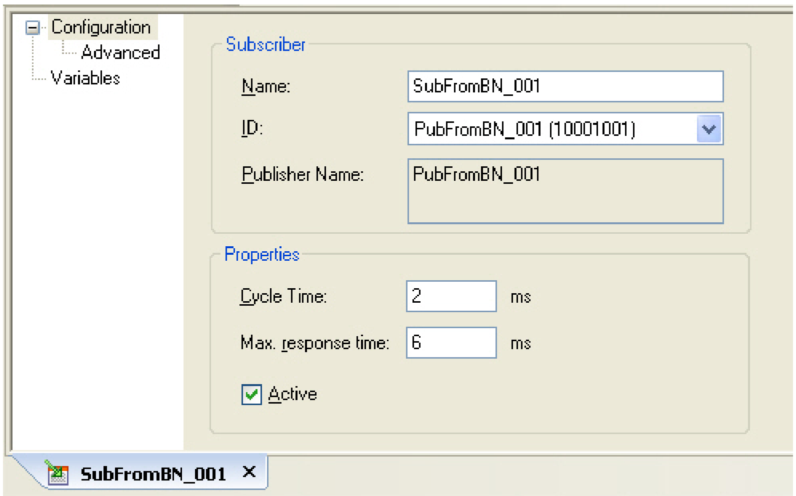
Configuring the controller

If you click in Hardware Manager/local controller on the publication **PubToBN_001**, the configuration for publishing opens.



This dialog shows the configuration for the publication (PubToBN_001) of a controller. This configuration applies to output data (process data) to be sent to the bus node, such as the state of an output of a peripheral module that is connected to the bus node.

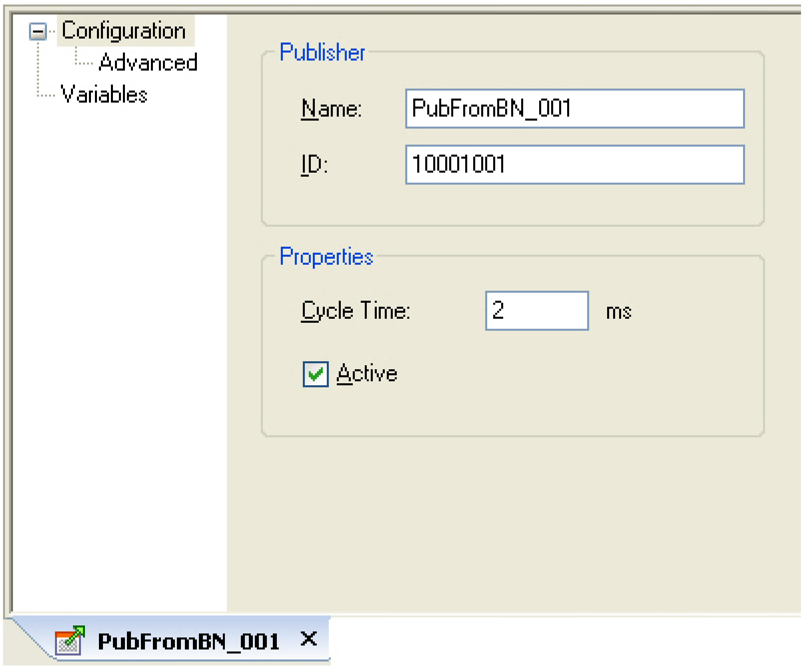
If you click in Hardware Manager/local controller on the subscription **SubFromBN_001**, the configuration for subscribing opens.



This dialog shows the configuration for the subscription (SubToBN_001) of a controller. The subscription is assigned to the publication (PubFromBN_001). JetSym creates the subscription automatically which lets the bus node publish input data and the controller subscribe to them.

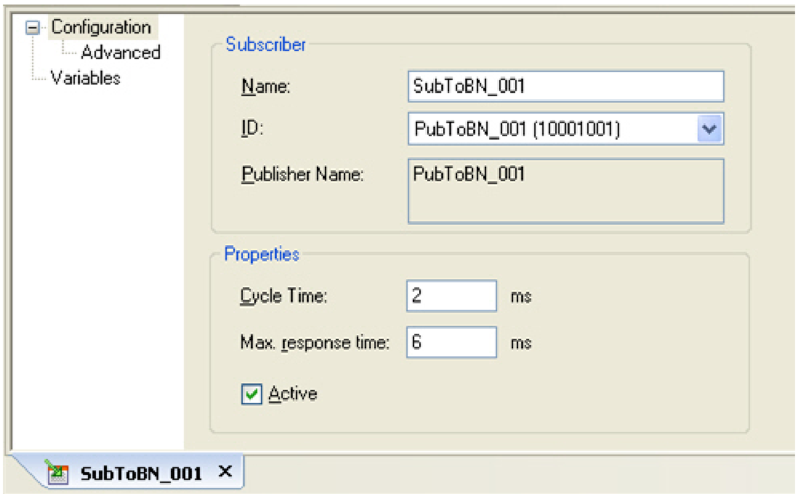
Configuring the bus node

If you click in Hardware Manager/bus node on the publication **PubFromBN_001**, the configuration for publishing opens.



This dialog shows the configuration for the publication (PubFromBN_001) of a bus node. This configuration applies to input data (process data) of the bus node, such as input data of a peripheral module which is connected to the bus node.

If you click in Hardware Manager/bus node on the subscription **SubToBN_001**, the configuration for subscribing opens.



This dialog shows the configuration for the subscription (SubToBN_001) of the bus node. The subscription is assigned to the publication (PubToBN_001), that is, to publishing. JetSym creates the subscription automatically which lets the controller publish output data and the bus node subscribe to them.

Publication parameter options

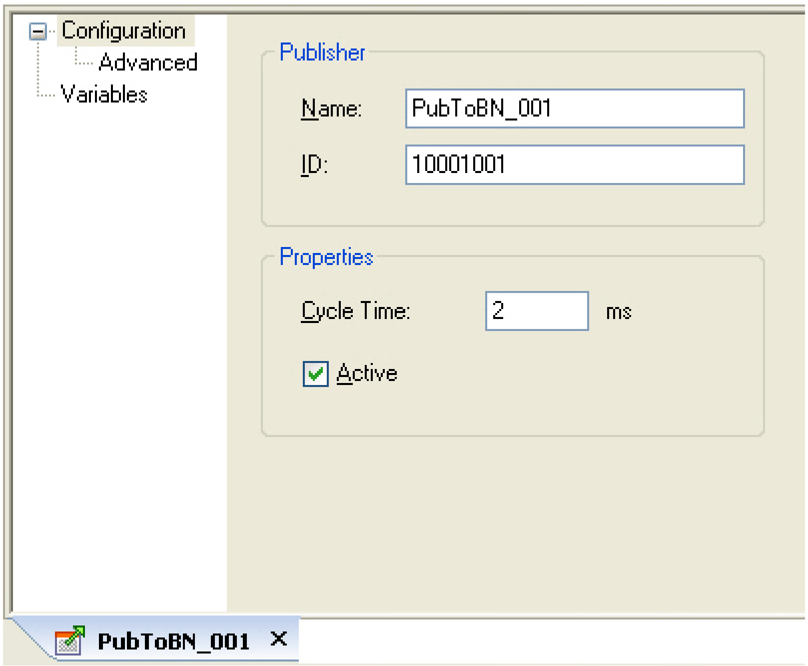
Introduction

In the Hardware Manager, the following publication parameters can be set:

- GNN and name of publication
- Times
- Network properties
- Using and creating further variables

Configuration parameter options

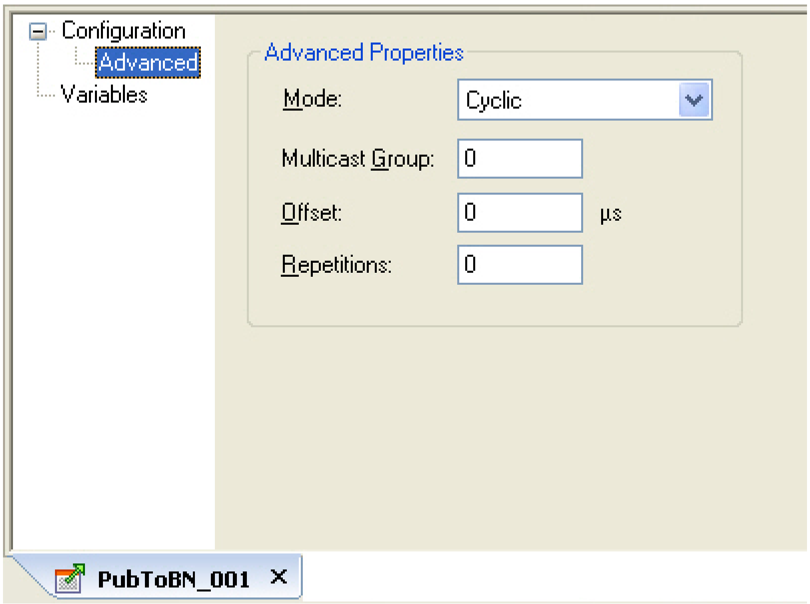
The respective configuration menu lets you make the following settings:



Part	Description
Name of the publication	The name is generated automatically. It consists of the following components: ▪ **Pub** being the prefix in **Publication** ▪ **To** designating the direction ▪ Here: **BN** being the target
ID of the publication project	JetSym manages each publication and subscription using IDs
Properties of the cycle time	The publisher sends the message in the set time interval. The cycle time for publishing is typically 2 ms. Theoretically, a value up to 2^{32} - 1 is possible.
Autorun properties	By clearing the check box, publication is disabled.

Advanced configuration parameter options

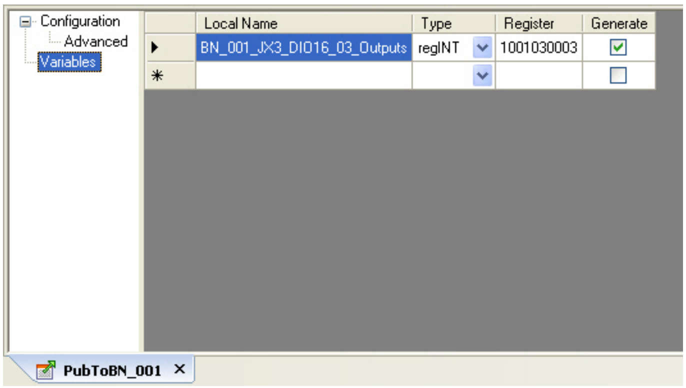
The respective advanced configuration menu lets you make the following settings:



Part	Description
Mode	The mode can be set as follows (presently, there is no alternative): Cyclic transmission of publications
Multicast group	You can sort multicasts in groups of 0 ... 254.
Offset	Publication is offset, that is, it is delayed by a defined time interval. Value range of the offset: 0 ... 2^{32} - 1 µs
Repetitions:	You can set the repeat count of the publication. Value range of the repetitions: 0 ... 2^{32} - 1 This only makes sense, though, if you have set a longer cycle time.

Variable parameter options

The **Variables** menu of the publication lets you make the following settings:



Part	Description
1/64	Hardware Manager lets you create a maximum of 64 process variables to be publish/subscribe variables. Hardware Manager automatically generates the variables of peripheral modules and the device-specific variables.

Local name	The local name of a variable consists of the following components:
	■ Prefix **Pub**
	■ BN for Bus Node
	■ GNN of the corresponding device
	■ JX3_DIO16: Name of the corresponding peripheral module
	■ Second peripheral module
	■ Outputs: What is being addressed?
	  ■ Input
	  ■ Output
	  ■ Status
	  ■ Channel x

Type	Presently supported: regInt and regFloat The published variable is 32 bits wide; the interpretation can be set.
Register	The automatically computed register
Generate	If the **Generate** check box has been marked, Hardware Manager creates a variable by the displayed name in the file **PubSubVariables.stxp**.

Subscription parameter options

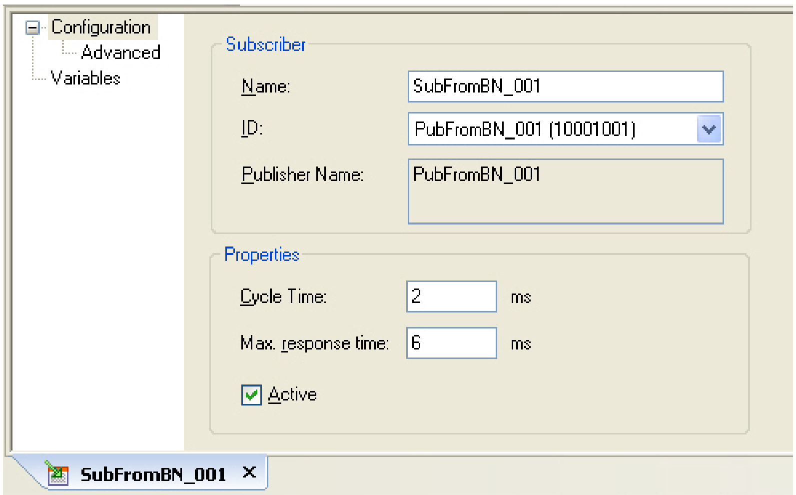
Introduction

In the Hardware Manager, the following subscription parameters can be set:

- GNN and name of subscription
- Times
- Network properties
- Using and creating further variables

Configuration parameter options

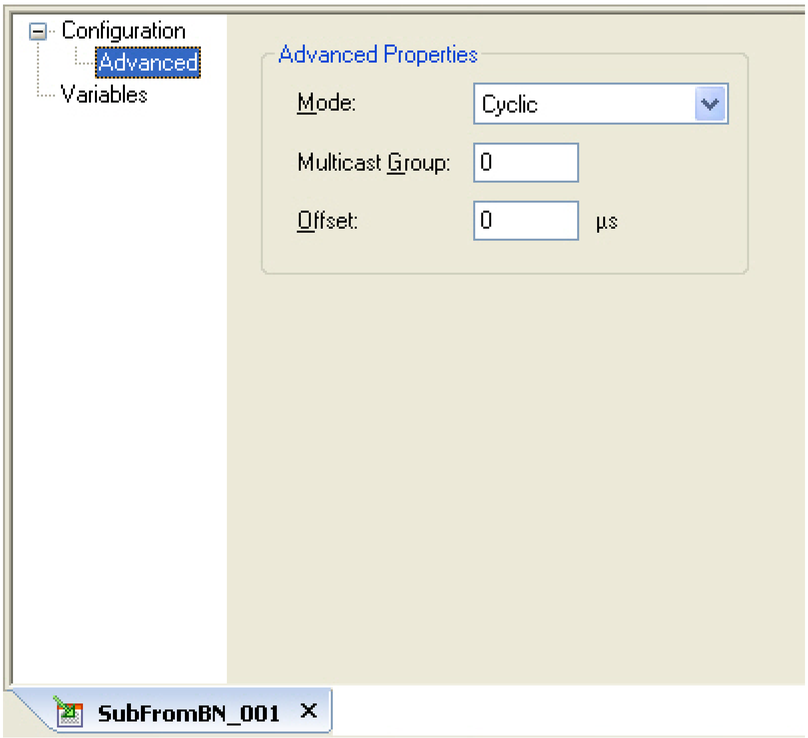
The respective configuration menu lets you make the following settings:



Part	Description
Name of the subscription	The name is generated automatically. It consists of the following components: ▪ **Sub** being the prefix in **Subscription** ▪ **From** designating the direction ▪ Here: **BN** being the source
ID of the subscription project	JetSym manages each publication and subscription using IDs
Properties of the cycle time	The device expects the subscription in the set time interval.
Properties of the maximum response time	At the remote bus node as viewed from the local controller, you can define a maximum response time. If the response time of a subscription has elapsed, an error bit is set in register R 255000.
Autorun properties	By clearing the check box, subscription is disabled.

Advanced configuration parameter options

The respective advanced configuration menu lets you make the following settings:

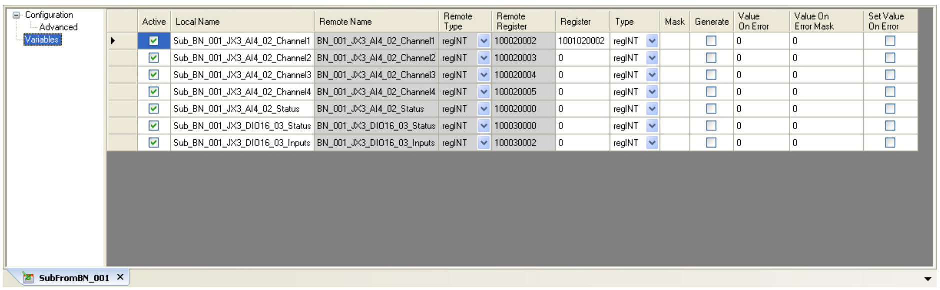


Part	Description
Mode	The mode can be set as follows (presently, there is no alternative): Cyclic reception of subscriptions
Multicast group	You can sort multicasts in groups of 0 ... 254.
Offset	Subscription is offset, that is, it is delayed by a defined time interval. Value range of the offset: 0 ... 2^{32} - 1 µs

Properties of the variables

Hardware Manager creates subscriptions depending on connected bus nodes and peripheral devices.

The **Variables** menu of the subscription lets you make the following settings:



<table>
<tr><th>Part</th><th>Description</th></tr>
<tr><td>Active</td><td>In the check box Active you can add or remove individual subscriptions.</td></tr>
<tr><td>Local Name</td><td>Local Name of the subscription consists of the following components:
Prefix Sub
BN for Bus Node
GNN of the corresponding device
JX3_DIO16: Name of the corresponding peripheral module
Second peripheral module
Outputs: What is being addressed?
Input
Output
Status
Channel x</td></tr>
<tr><td>Remote Name</td><td>Name of the local module register of the removed device</td></tr>
<tr><td>Remote Type</td><td>Presently supported: regInt and regFloat
The variable subscribed to is 32 bits wide; the interpretation can be set.</td></tr>
<tr><td>Remote Register</td><td>Module register of the remote device</td></tr>
<tr><td>Register</td><td>The copy of the Remote Register subscribed to is stored to this register.</td></tr>
<tr><td>Type</td><td>Presently supported: regInt and regFloat
The variable subscribed to is 32 bits wide; the interpretation can be set.</td></tr>
<tr><td>Mask</td><td>You can hide desired bits by a mask.</td></tr>
<tr><td>Generate</td><td>If the Generate check box has been marked, Hardware Manager creates a variable by the displayed name in the file PubSubVariables.stxp.</td></tr>
<tr><td>Value On Error</td><td>If you mark the check box, you can have the variable subscribed to set to a defined value under fault condition.</td></tr>
<tr><td>Value On Error Mask</td><td>Under fault condition, the mask hides the bits of the respective value.</td></tr>
<tr><td>Set Value On Error</td><td>If the Set Value On Error check box has been marked, the subscribed variable takes the preset Value On Error.</td></tr>
</table>

Generated publish/subscribe variables

Introduction

If you mark the **Generate** check box for the variables of publish/subscribe, they will be to your disposal in the JetSym program.

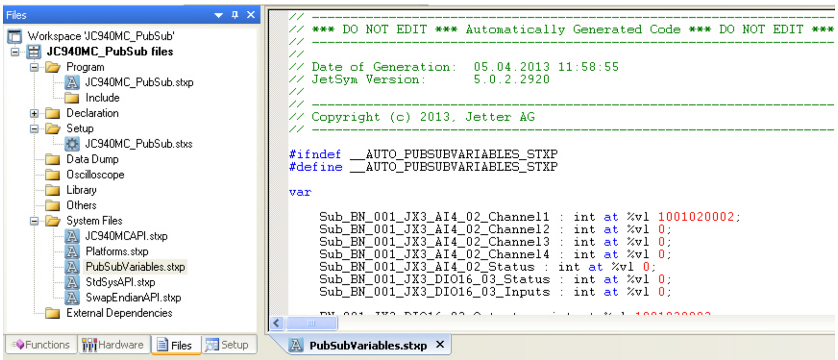
Variable storage

You can view the generated variables in the JetSym file **PubSubVariables.stxp**. The include file is automatically included in the code. To access the generated publish/subscribe variables, you do not need an `#include`-instruction.

Important Note!

Do **not change** the contents of the file **PubSubVariables.stxp**.

By means of the Hardware Manager, JetSym manages the file **PubSubVariables.stxp**.



Updating within JetSym

If, in JetSym you execute a build (F7), JetSym checks your configurations by means of the Hardware Manager and then updates the file **PubSubVariables.stxp**.

Then upload the new configuration to the controller by selecting **Download All Configurations.**

Example of an include file

In the following example, Hardware Manager has generated the include file **PubSubVariables.stxp**. Hardware Manager has detected the following hardware:

- JC-940MC of GNN = 000
- JX3-BN-ETH of GNN = 001
- JX3-DIO16 at the bus node JX3-BN-ETH

```
//
--------------------------------------------------------------------
----
// *** DO NOT EDIT *** Automatically Generated Code *** DO NOT EDIT
***
//
--------------------------------------------------------------------
----
//
```

```
// Date of Generation:  04.04.2013 10:03:47
// JetSym Version:      5.0.2.2920
//
//
-----------------------------------------------------------------
----
// Copyright (c) 2013, Jetter AG
//
-----------------------------------------------------------------
----

#ifndef __AUTO_PUBSUBVARIABLES_STXP
#define __AUTO_PUBSUBVARIABLES_STXP

var

    BN_001_JX3_DIO16_02_Outputs : int at %vl 1001020003;

end_var;

#endif //__AUTO_PUBSUBVARIABLES_STXP
```

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05
- Bus node JX3-BN-ETH of OS version 1.18
- Module JX3-AI4 of OS version 1.04

For more up-to-date sample programs, please refer to the JetSym online help.

JetSym STX program

```
Task Use_PubSub Autorun
    // All digital outputs of the peripheral module JX3-DIO16
    // are flashing
    Loop
        BN_001_JX3_DIO16_02_Outputs := 0xFF;
        Delay(T#100ms);
        BN_001_JX3_DIO16_02_Outputs := 0x00;
        Delay(T#500ms);
    End_Loop;
End_Task;
```

Publish/subscribe - Registers

Introduction

If you transmit cyclic data by publish/subscribe, there are several module registers available for administration, configuration and error detection. You have got read and partial write access to these module registers.

Register overview

Module registers	Description
210004, 200008, 200009	General error registers
250000 ... 250004	Registers for administration of all subscriptions
250x10 ... 250x11	Registers for administration of one subscription
250x20 ... 250x30	Registers for configuring one subscription
254001 ... 254003	Registers for error detection
255000 ... 255004	Registers for administration of all publications
255x10 ... 255x11	Registers for administration of one publication
255x20 ... 255x30	Registers for configuring one publication
Flag 2080	Enable for publishing an error
Flag 2081	Error collection of the subscriber

x = 0 ... 9

Availability

Administration and configuration registers are available as follows:

- For subscriptions and publications, 10 arrays for administration and configuration registers are available.
- The register arrays differ by the hundred's place of the respective register number.
- The placeholder x indicates the number of the register array. Value range of x: 0 ... 9
- External clients use register array x = 1, such as JetSym with visualization and PCOMX protocol.
- STX functions use register array x = 0.
- In order to gain faster access to individual publish/subscribe administration registers, several register arrays are at your disposal: There are individual publish/subscribe IDs to be called in each register array.

Registers for administration of all subscriptions

There are several registers available which go with all subscriptions.

Register	Name	Description
250000	Status	Status register
250001	Command	Command register
250002	ID in case of error	Displays the ID of the subscription, in which an error has occurred.
250003	Amount	Total amount of subscriptions
250004	CRC	16-bit CRC (**C**yclic **R**edundancy **C**ode) of the subscriber configuration file

Subscriber status

Status registers of all subscriptions

From MR 250000, you can read the collective status of all subscriptions. In case of an error, you first read out the ID of the subscription, in which an error has occurred.

Meaning of the individual bits		
Bit 0	**Error in CRC computing of the configuration file**	
	0 =	No error has occurred.
	1 =	For CRC computing, the configuration file does not exist. For this reason, CRC computing has not taken place.
Bit 1	**Error in connection with a subscription**	
	1 =	An error has occurred in a subscription. At the moment, this is only a timeout error.
Bit 7	**Subscription is functioning.**	
	0 =	If a subscription fails, bit 7 is reset.
	1 =	The subscriptions are functioning.

Module register properties	
Type of access	Read

Subscriber command

Command registers of all subscriptions

Via MR 250001, you transmit commands to all subscriptions.

Commands	
102	**Reboot all subscribers**
105	**Stop all subscribers**
110	**Acknowledge error**

Selecting a subscription

The following registers let you select a subscription as follows:

- The index is for selecting subscriptions.
  - If the subscription exists, R 250x11 shows its ID.
  - If the subscription does not exist, R 250x11 shows value -1.
- In this case, enter the ID of the subscription into R 250x11.
  - If the subscription exists, the content of R 250x11 is kept.
  - If the subscription does not exist, R 250x11 shows value -1.

Register	Name	Description
250x10	Index	Index of the subscriptions: 0: Selects the first subscription 1: Selects the next subscription 2: etc.
250x11	ID	The ID of the subscription is entered

Configuring a subscription

The following registers show the configuration of a subscription, which you have selected via R 250x10 and R 250x11.

Register	Name	Description
250x20	Status	Bit 0: Publication received Bit 1: Timeout
250x21	Mode	0: Cyclic 1: Upon request
250x22	Number of variables	As configured
250x23	Group address	As configured
250x24	Hash	Internal usage
250x25	Sequence number	Internal usage
250x26	Data size	Internal usage
250x27	Timeout in ms	Bus cycle * 3
250x28	Number of received publications	-
250x29	Amount of timeouts	-
250x30	Amount of missing sequence numbers	The subscriber of a publication computes the difference between present and last received sequence number. If the value of the difference is greater than one, certain publications have not been received.

Registers for error detection

If a subscription has not received any process data from the assigned publication before timeout, the subscription will generate an error. Further, the operating system writes the address of the bus node into registers 254001 to 254003, with which communication has been terminated.

This helps you to search for the error exactly in this bus node using NetCopy commands.

Register	Name	Description
254001	GNN	Global Node Number
254002	IP address	
254003	Port number	

Registers for administration of all publications

There are several registers available which go with all publications.

Register	Name	Description
255000	Status	Status register
255001	Command	Command register
255002	ID in case of error	Displays the ID of the publication, in which an error has occurred.
255003	Amount	Amount of all publications
255004	CRC	16-bit CRC (Cyclic Redundancy Code) of the publication configuration file

Publisher status

Status registers of all publications

From MR 255000, you can read the collective status of all publications. In case of an error, you first read out the ID of the publication, in which an error has occurred.

Meaning of the individual bits		
Bit 0	**Error in CRC computing of the configuration file**	
	0 =	No error has occurred.
	1 =	For CRC computing, the configuration file does not exist. For this reason, CRC computing has not taken place.
Bit 1	**Error in connection with a publication**	
	1 =	An error has occurred in a publication.
		Presently, no error messages have been sent yet.
Bit 7	**Publication is functioning**	
	0 =	If a publication fails, bit 7 is reset.
	1 =	The publications are functioning.

Module register properties	
Type of access	Read

Publisher command

Command registers of all publications

Via MR 255001, you transmit commands to all publications.

Commands	
102	**Reboot all publishers**
105	**Stop all publishers**
110	**Acknowledge error**

Selecting a publication

The following registers let you select a publication:

- The index is for selecting publications.
  - If the publication exists, R 255x11 shows its ID.
  - If the publication does not exist, R 255x11 shows value -1.
- In this case, enter the ID of the publication into R 255x11.
  - If the publication exists, the content of R 255x11 is kept.
  - If the publication does not exist, R 255x11 shows value -1.

Register	Name	Description
255x10	Index	Index of the publications: 0: Selects the first publication 1: Selects the next publication 2: etc.
255x11	ID	The ID of the publication is entered

Configuring a publication

The following registers show the configuration of a publication, which you have selected via R 255x10 and R 255x11.

Register	Name	Description
255x20	Status	Bit 0: Publication transmitted
255x21	Mode	0: Cyclic 1: Upon request
255x22	Number of variables	As configured
255x23	Group address	As configured
255x24	Hash	Internal usage
255x25	Sequence number	Internal usage
255x26	Data size	Internal usage
255x27	Timeout in ms	Bus cycle
255x28	Number of publications sent	-

Register	Name	Description
255x29	Number of retries	-
255x30	Number of transmit errors	-

9.3.3 Error handling at the Jetter Ethernet system bus

Introduction

This chapter covers error handling at the Jetter Ethernet system bus.

Contents

Error message during CRC computing

Detecting the error

Both publisher and subscriber carry out a CRC of their configuration files. The calculated value can be read from registers 255004 and 250004. If there is no configuration file, they report an error.

Root cause of the error

This error may be caused by the following root cause:

- CRC computing failed, because there is no configuration file.

Response of the device to this error

The operating system of the device responds to the error by taking the following steps:

Step	Description
1	The operating system sets bit 0 in the status register of the publisher (R 255000) or of the subscriber (R 250000).

Fixing the root cause

Deploying a configuration file

Acknowledging the error

After deploying a configuration file, restart both publisher and subscriber.

Error message on part of a subscription

Detecting the error

If a subscriber has not received any process data from the assigned publisher before timeout, the subscriber will generate an error. The subscriber for the subscription of which the error has been generated, can run either on a controller or on a network node. The remote network node is a JX3-BN-ETH, for example.

Root cause of the error

The error may be caused as follows:

- Communication with the network client providing the process data is terminated.

Response of the device to this error

The operating system of the device responds to the error by taking the following steps:

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">Sets bit 1 in R 250000.</td></tr>
<tr><td>2</td><td colspan="2">Writes the subscription ID to R 250002.</td></tr>
<tr><td>3</td><td colspan="2">Sets flag 2081.</td></tr>
<tr><td>4</td><td colspan="2">Writes value 11103 and the ID to the error buffers.
The error buffer can be accessed via registers 380000 ff. (error history).</td></tr>
<tr><td>5</td><td colspan="2">Writes the GNN of the network node communication with which has been terminated to R 254001.</td></tr>
<tr><td>6</td><td colspan="2">Writes the IP address of the network node communication with which has been terminated to R 254002.</td></tr>
<tr><td>7</td><td colspan="2">Writes the port number of the network node communication with which has been terminated to R 254003.</td></tr>
<tr><td rowspan="2">8</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>... flag 2080 is set,</td><td>... bit 3 is set in R 210004 and R 200008. The red status LED of the controller is lit.</td></tr>
</table>

Fixing the root cause

By means of NetCopy commands, you can precisely locate the error and fix the root cause. This works, because GNN, IP address and port number of the other network node are known.

Acknowledging the error

To acknowledge the error, write command 110 to register 250001.

Controller evaluates errors reported by a remote network node

Access to the status registers

The controller has got read access to the contents of the following status registers of all network nodes at the Jetter Ethernet system bus.

The contents are accessed via registers 39nnn0 through 39nnn5. (GNN: nnn = 001 ... 199).

Registers	JX3-BN-ETH, JX3-COM-EIPA	Controller
Error register	200008	39nnn0
Enhanced error register 1	200009	39nnn1
Enhanced error register 2	200010	39nnn2
JetSync status	240010	39nnn3
Subscriber status	250000	39nnn4
Subscription ID	250002	39nnn5

The operating system writes the ID of the subscription for which last an error has been reported to register 250002.

Locating faults

If the value of register 39nnn0 is unequal zero, an error has occurred. A network node has reported this error to the controller via its status registers.

In consequence, the operating system of the controller reacts by taking the following steps:

<table>
<tr><th>Step</th><th colspan="3">Description</th></tr>
<tr><td>1</td><td colspan="3">The operating system sets bit 10 in R 200009.</td></tr>
<tr><td rowspan="2">2</td><th>If ...</th><th>... or ...</th><th>... then ...</th></tr>
<tr><td>... Bit x = 1 of R 200009,</td><td>Bit x = 1 of R 200010,</td><td>... the operating system sets bit 7 of R 200008.</td></tr>
<tr><td>3</td><td colspan="3">The operating system enters the GNN of the network node having last reported an error to the controller into R 394001.</td></tr>
<tr><td>4</td><td colspan="3">The operating system enters the IP address of the network node having last reported an error to the controller into R 394002.</td></tr>
<tr><td>5</td><td colspan="3">The operating system enters the port number of the network node having last reported an error to the controller into R 394003.</td></tr>
</table>

Fixing the root cause

By means of NetCopy commands, you can precisely locate the error and fix the root cause. This works, because GNN, IP address and port number of the other network node are known.

Make sure the contents of registers 39nnn0 through 39nnn5 are read by the application program. Further registers having got a value unequal zero indicate that further network nodes have reported an error. Make sure you also clear these errors.

9.3.4 NetConsistency function

Target

The goal of NetConsistency is automated comparison of actual system properties with the set system properties. 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 and transfers them 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.

Availability

NetConsistency is available for the following product versions:

Product	As of version
JetSym	V 5.1.0
JC-940MC	V 1.05.0.08
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-BN-ETH	V 1.18.0.02
JX3-COM-EIPA	V 1.01.0.00

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.
- 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 system property *Network parameters*:

- IP address
- Subnet mask
- Default gateway

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

The JC-940MC assigns the network parameters to those bus nodes which you have configured in Hardware Manager.

As subnet mask, the JC-940MC assigns its own subnet mask to the bus node.

As default gateway, the JC-940MC assigns its own IP address to the bus node.

System launch of the bus nodes

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 the Hardware Manager via JetControl - which is the NetConsisteny master - are assigned to the bus node.

Remanent storing via NetConsistency of the network parameters assigned last is not implemented.

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

System launch of the JX3-BN-ETH

The network parameters assigned by NetConsistency are saved to the remanent store in the config.ini file of the JX3-BN-ETH, when the DIP switch sliders 9 through 12 of the JX3-BN-ETH are in the position listed below.

DIP switch sliders	Position
9	ON
10	OFF
11	OFF
12	OFF

The GNN of the JX3-BN-ETH is 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 JX3-BN-ETH uses the network parameters stored in **config.ini**, until the network parameters configured in the Hardware Manager via JetControl - which is the NetConsistency master - are assigned to the JX3-BN-ETH. If NetConsistency has already assigned the network parameters configured in Hardware Manager to the JX3-BN-ETH, the JX3-BN-ETH uses these for system launch.

The JX3-BN-ETH stores the assigned network parameters in the file **System\config.ini** in the file system. In this case, the already existing file **config.ini** is overwritten.

The GNN set by the DIP switch of the JX3-BN-ETH is for identifying the JX3-BN-ETH within the system in order to assign the network parameters configured in Hardware Manager.

Assigning the network parameters dependent on the GNN

Introduction

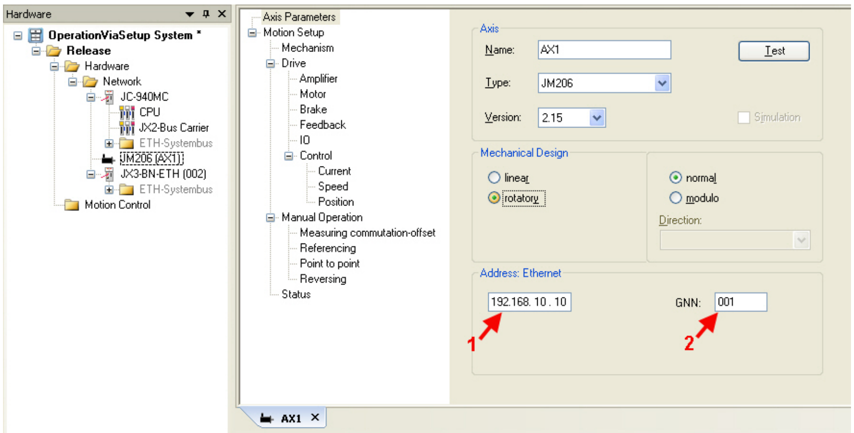
Via JetIPScan, NetConsistency sets the network parameters automatically for the following devices:

- Ethernet axes JM-xxx (JM-2xx-OEM)
- Ethernet axes MC-JM-xxx (JM-2xx-OEM)
- JX3-BN-ETH
- JX3-COM-EIPA

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.

Network parameter assignment for MC-JM-xxx or JM-xxx



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	Launch 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.

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.

Examples

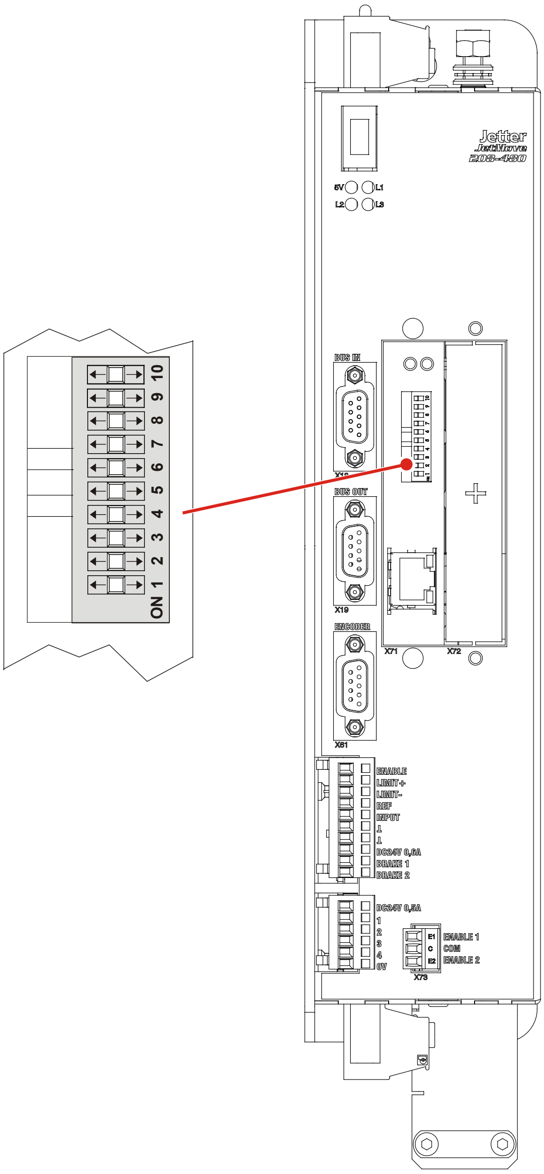
GNN = 4: Switch 3 is set to ON. All other DIP switch sliders are set to OFF.

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

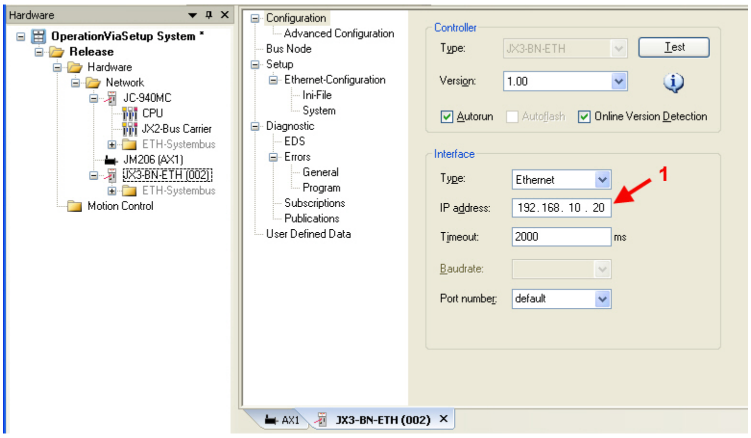
GNN = 8: Switch 4 is set to ON. All other DIP switch sliders are set to OFF.

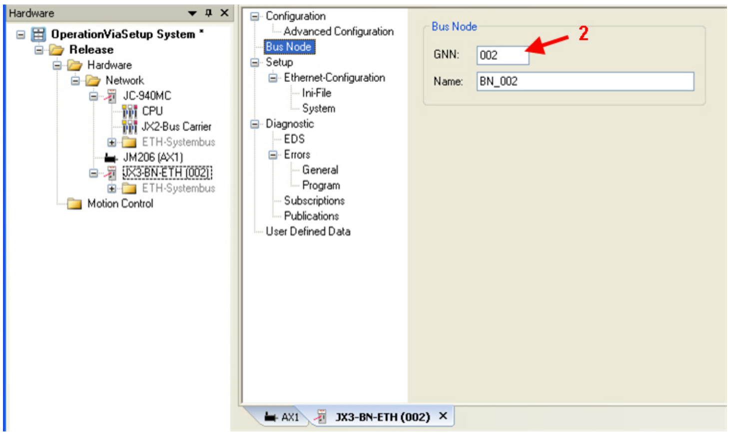
Position of the DIP switch sliders at the MC-JM-xxx or JM-xxx

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.



Assignment at the JX3-BN-ETH





Step	Action
1	Set the GNN at the DIP switch (DIP switch sliders 1 through 8) of the JX3-BN-ETH.
2	Set the operating mode GNN at the DIP switch (DIP switch sliders 9 through 12) of the JX3-BN-ETH.
3	Launch JetSym.
4	Select the device JX3-BN-ETH 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.

Setting the DIP switch sliders at the JX3-BN-ETH

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.

Examples

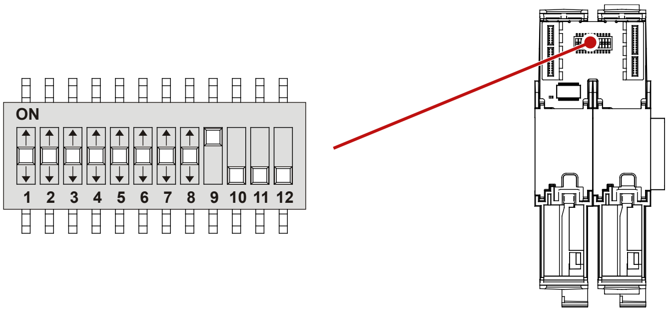
GNN = 4: Switch 3 is set to ON. All other DIP switch sliders are set to OFF.

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

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



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)

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	During the boot process, each network node, except for JetControl and JX3-BN-ETH, uses the settings of the DIP switch as a fixed IP address.
2	During the boot process of the JetControl, each network node is assigned a network configuration (IP address, subnet mask, gateway address) via JetIPScan at executing the NetConsistency function.
3	After the boot process of the JetControl, and thus, after executing the NetConsistency function, the network nodes can be reached by the network configurations set in the Hardware Manager.

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 is executed.

Register description - NetConsistency basic driver

Register overview

Register	Description
470000 ... 470008	Cookie
470009	Version number
470010	Status
470011	Command
470020	Maximum possible number 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	
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 evaluation at NetConsistency

Possibilities of error output

There are the following possibilities of error output:

- Onto a screen via VGA interface of the JC-940MC
- Via the logger of NetConsistency
- 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.

Error name	Error number
NoError	0
AllDiff	-1
CantSetIp	-2
Already	-3
CantSaveIp	-4
InvalidList	-10
InvalidId	-11
Internal	-12
NotRunning	-20
WrongType	-1000
NotFound	-2000
MultiFound	-3000

Module register properties	
Values	-2^{16} ... 0
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 249)

9.3.5 Administrating the connections of the JetIP/TCP and STX debug server

Introduction

This document covers the connection management enhancements of the JetIP/TCP server and of the STX debug server in a JetControl PLC.

Previous to this enhancement, nodes were only able to clear active connections with other nodes. If, for example, the Ethernet cable was unplugged or cut, the node was not able to clear the connection. The connection remained active.

The enhanced connection management allows for the server to clear connections according to criteria that can be set by the user.

Number of connections

The number of simultaneously established connections for the TCP server in a JetControl is limited to the following value:

Server	Connections
JetIP/TCP server	4
STX debug server	20

Contents

Automatic termination of connections

Introduction

If the maximum number of simultaneously established connections has been reached, any further connections cannot be established. If further connect requests are made, the user can set the response by the JetIP/TCP server and of the STX Debug server. There are the following possibilities:

- Reject new connection.
- Terminate one existing connection and establish the new one.
- Terminate all existing connections and establish the new one.

Default setting

By default, the server terminates the connection with the longest time of inactivity.

No automatic termination of connections

If the server is not to terminate any of the existing connections, proceed as follows:

Step	Action
1	Enter value 0 into MR 1.

Terminating the connection with the longest time of inactivity

If the server is to terminate the connection that has been inactive the longest time, proceed as follows:

Step	Action
1	Enter value -1 into MR 2.
2	Enter value 1 into MR 1.

Terminating the connection when the set minimum time has expired

If the server is to terminate a connection after a set minimum time of inactivity, proceed as follows:

Step	Action
1	Enter the minimum time [ms] into MR 2.
2	Enter value 1 into MR 1.

If the set minimum value has not been exceeded yet, the server rejects the new connection.

Terminating any connection

If the server is to terminate any of the existing connections, proceed as follows:

Step	Action
1	Enter value 2 into MR -1.
2	Enter value 1 into MR 2.

Terminating all connections which exceed the minimum time of inactivity

If the server is to terminate all existing connections which have exceeded the minimum time of inactivity proceed as follows:

Step	Action
1	Enter the minimum time [ms] into MR 2.
2	Enter value 1 into MR 2.

Register

Register numbers

The register numbers to be used are calculated by adding and the controller-dependent basic register number and the module register number.

Controller/server	Basic register number	Register numbers
JC-24x: JetIP/TCP	2755	2755 ... 2757
JC-340, JC-350, JC-360(MC), JC-940MC: JetIP/TCP	230000	230000 ... 230002
JC-340, JC-350, JC-360(MC), JC-940MC: STX-Debug	212000	212000 ... 212002

Number of connections

The number of currently established connections can be read from module register 0.

Module register properties	
Values	0 ... 4 (JetIP/TCP server) 0 ... 20 (STX debug server)

Mode

If the maximum number of connections is active, and if the server is to establish a new connection, module registers 1 and 2 determine the behavior.

Module register properties	
Values	0 ... 2
Value after reset	1

Minimum inactivity time

If the maximum number of connections is active, and if the server is to establish a new connection, module registers 1 and 2 determine the behavior.

Module register properties	
Values	-1 ... 2,147,483,647 [ms]
Value after reset	-1

9.4 Startup delay register

Introduction

The device JC-940MC 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 JC-940MC. For this, mode selector S11 must be in *RUN* position.
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.

Display

- The yellow LED will remain lit during the delay time.
- With advancing delay time, the flashing intervals of the green LED become shorter and shorter.

9.5 Real-time clock (RTC)

Introduction

The JC-940MC is equipped with a component which maintains time and date settings for a certain time even when it is not energized.

Usage by OS

The OS uses the real-time clock for the following functions:

- Storing file date and time to a file

Restrictions

When using the real-time clock, the following restrictions apply:

- When the device is de-energized the power reserve is limited.
- The real-time clock has no automatic daylight savings time function.

Contents

Technical specifications

Technical data - Real-time clock

Parameter	Description
Lifetime	Minimum: 10 years
Deviation	1 minute per month max.

Factory settings

At the end of the controller manufacturing process, the manufacturers set the real-time clock to the actual date and time.

Programming

Programming using STX

To program date and time it is advisable to use the functions provided by JetSym STX:

- DateTimeActual()
- DateTimeDecode()
- DateTimeEncode()
- DateTimeIsValid()
- DateTimeSet()

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

If you make use of the above functions, the minimum time interval is one second. If you need a time interval of one second, programming must be made by using the registers described below.

Programming using registers

Depending on the respective application, access to the real-time clock via registers might be required. For this, there are two register sets:

- Register set 1 is for directly accessing individual real-time clock values.
- Changes to values in register set 1 are immediately transferred to the real-time clock.
- Register set 2 operates within a buffer. In the buffer, all real-time clock values are consistently read out and written.
- Not before the trigger register is written to, the value changes made in or out of register set 2 are transferred.

Register overview

The following registers have been assigned to the real-time clock:

Register set 1: Direct access

Register	Description
R 102910	Milliseconds
R 102911	Seconds
R 102912	Minutes
R 102913	Hours
R 102914	Weekday (0 = Sunday)
R 102915	Day
R 102916	Month
R 102917	Year

Register set 2: Buffer access

Register	Description
R 102920	Milliseconds
R 102921	Seconds
R 102922	Minutes
R 102923	Hours
R 102924	Weekday (0 = Sunday)
R 102925	Day
R 102926	Month
R 102927	Year
R 102928	Read/write trigger

R 102910

Milliseconds

This register contains the millisecond of the actual time.

Register properties	
Values	0 ... 999
Value after reset	0

R 102911

Seconds

This register contains the seconds of the actual time.

Register properties		
Values	0 ... 59	
Value after reset	**If ...**	**... then ...**
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	0

R 102912

Minutes

This register contains the minutes of the actual time.

Register properties	
Values	0 ... 59

Value after reset	If ...	... then ...
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	0

R 102913

Hours

This register contains the hours of the actual time.

Register properties		
Values	0 ... 23	
Value after reset	**If ...**	**... then ...**
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	0

R 102914

Weekday

This register contains the weekday of the actual date.

Register properties		
Values	0 ... 6 (0 = Sunday)	
Value following a reset	**If ...**	**... then ...**
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	0

R 102915

Day

This register contains the day of the actual date.

Register properties		
Values	1 ... 31	
Value after reset	**If ...**	**... then ...**
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	1

R 102916

Month

This register contains the month of the actual date.

Register properties		
Values	1 ... 12	
Value after reset	**If ...**	**... then ...**
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	1

R 102917

Year

This register contains the year of the actual date.

Register properties		
Values	0 ... 99	
Value after reset	**If ...**	**... then ...**
	the power reserve has not elapsed,	actual time
	the power reserve has elapsed,	0

R 102920

Milliseconds

This register contains the milliseconds stored in the buffer.

Register properties	
Values	0 ... 999
Value after reset	0
Takes effect	After read/write access to register 102928

R 102921

Seconds

This register contains the seconds stored in the buffer.

Register properties	
Values	0 ... 59
Value after reset	0
Takes effect	After read/write access to register 102928

R 102922

Minutes

This register contains the minutes stored in the buffer.

Register properties	
Values	0 ... 59
Value after reset	0
Takes effect	After read/write access to register 102928

R 102923

Hours

This register contains the hours stored in the buffer.

Register properties	
Values	0 ... 23
Value after reset	0
Takes effect	After read/write access to register 102928

R 102924

Weekday

This register contains the weekday stored in the buffer.

Register properties	
Values	0 ... 6 (0 = Sunday)
Value following a reset	0
Takes effect	After read/write access to register 102928

R 102925

Day

This register contains the day stored in the buffer.

Register properties	
Values	0 ... 31
Value after reset	0
Takes effect	After read/write access to register 102928

R 102926

Month

This register contains the month stored in the buffer.

Register properties	
Values	0 ... 12
Value after reset	0
Takes effect	After read/write access to register 102928

R 102927

Year

This register contains the year stored in the buffer.

Register properties	
Values	0 ... 99
Value after reset	0
Takes effect	After read/write access to register 102928

R 102928

Read/write trigger

This register allows transferring values between buffer register and real-time clock.

Register properties	
Read	The actual date and time are transferred from real-time clock to buffer registers 102920 through 102927. The reading is undefined.
Write	The values contained in buffer registers 102920 ... 102927 are transferred to the real-time clock. The value written is ignored.

Sample program for real-time clock

Task

Read the actual time and date of the JC-940MC and have the values displayed.

Solution

An application program task reads out the real-time clock at regular intervals. Then it outputs the readings properly formatted as trace message.

When you activate the trace mode in JetSym, JetSym displays these readings.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

JetSym STX program

```
Type
    // Structure of the RTC buffer
    TimeAndDate:   Struct
                    Second:       Int;
                    Minute:       Int;
                    Hour:         Int;
                    DayOfWeek:    Int;
                    Day:          Int;
                    Month:        Int;
                    Year:         Int;
                    Trigger:      Int;
                   End_Struct;
End_Type;

Var
    RTCregs:     TimeAndDate At %VL 102921;
End_Var;

Task ShowTimeAndDate Autorun
    Var
        Dummy:    Int;
    End_Var;

    Loop
        // Wait for one second
        Delay(T#1s);
        // Copy current time and current date
        // to RTC buffer
        Dummy := RTCregs.Trigger;
```

```
        // Displaying day of the week
        Case RTCregs.DayOfWeek Of
            0:  Trace('Sunday');
                Break;
            1:  Trace('Monday');
                Break;
            2:  Trace('Tuesday');
                Break;
            3:  Trace('Wednesday');
                Break;
            4:  Trace('Thursday');
                Break;
            5:  Trace('Friday');
                Break;
            6:  Trace('Saturday');
                Break;
        End_Case;
        // Displaying date
        Trace(StrFormat(' , %2d.%02d.%4d , ',
                        RTCregs.Day,
                        RTCregs.Month,
                        RTCregs.Year + 2000));
        // Zeit anzeigen (plus cr/lf)
        Trace(StrFormat('%2d:%02d:%02d$n',
                        RTCregs.Hour,
                        RTCregs.Minute,
                        RTCregs.Second));
    End_Loop;
End_Task;
```

9.6 Runtime registers

Introduction

The JC-940MC provides several registers which are incremented by the operating system at regular intervals.

Application

These registers can be used to easily carry out time measurements in the application program.

Contents

Description of the runtime registers

Register overview

The device is equipped with the following runtime registers:

Register	Description
R 201000	Application time base in milliseconds
R 201001	Application time base in seconds
R 201002	Application time base in R 201003 * 10 ms
R 201003	Application time base units for R 201002
R 201004	System time base in milliseconds
R 201005	System time base in microseconds

R 201000

Application time base in milliseconds

Every millisecond this register is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)

R 201001

Application time base in seconds

Every second this register is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)

R 201002

Application time base in application time base units

Every [R 201003] * 10 ms this register value is incremented by one. Using the reset value 10 in register 201003, this register is incremented every 100 ms.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)

R 201003

Application time base units for R 201002

This register contains the multiplier for runtime register R 201002.

Register properties	
Values	1 ... 2,147,483,647 (* 10 ms)
Value after reset	10 (--> 100 ms)
Enabling conditions	After at least 10 ms

R 201004

System time base in milliseconds

Every millisecond this register value is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)
Type of access	Read

R 201005

System time base in microseconds

Every microsecond this register value is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)
Type of access	Read

Sample program - Runtime registers

Task

Measure how much time it takes to store variable values to a file.

Solution

Before storing the values, set register 201000 to 0.

Once the values have been stored, you can see from this register how much time it took to store the values [in milliseconds].

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

JetSym STX program

```
Var
    DataArray:     Array[2000] Of Int;
    File1:         File;
    WriteTime:     Int;
    WriteIt:       Bool;
    MilliSec:      Int At %VL 201000;
End_Var;

Task WriteToFile Autorun
    Loop
        // Resetting the start flag
        WriteIt := False;
        // Wait for user to set start flag
        When WriteIt Continue;
        // Opening the file in write mode
        // If there is no file available, a new file
        // is created
        If FileOpen(File1, 'Test.dat', fWrite) Then
            // Set the application time base register to null
            MilliSec := 0;

            // Write the data range into the file
            FileWrite(File1, DataArray, SizeOf(DataArray));
            // Register the run time
            WriteTime := MilliSec;
            FileClose(File1);
            // Display the run time
            Trace(StrFormat('Time : %d [ms]$n', WriteTime));
        Else
            // Display the error message
            Trace('Unable to open file!$n');
        End_If;
    End_Loop;
End_Task;
```

9.7 Monitoring the interface activity

Introduction

Several servers for variables have been integrated into the controller to make variables used within the controller accessible from outside. These servers support several protocols on different interfaces. The servers do not require any programming in the application program, but process requests from external clients on their own.

This chapter explains one possibility for detecting from within the application program whether communication with the servers takes place through these interfaces.

Monitored interface activities

The following interface activities can be monitored:

- JetIP server via Ethernet interface
- STX debug server via Ethernet interface

Application

The monitoring function for interface activities is used, amongst others, for the following scenarios:

- Plants requiring process visualization to ensure safe operation. They can be transferred into a save condition if communication fails.
- When the service technician connects an HMI, the application program automatically displays additional status information.

Contents

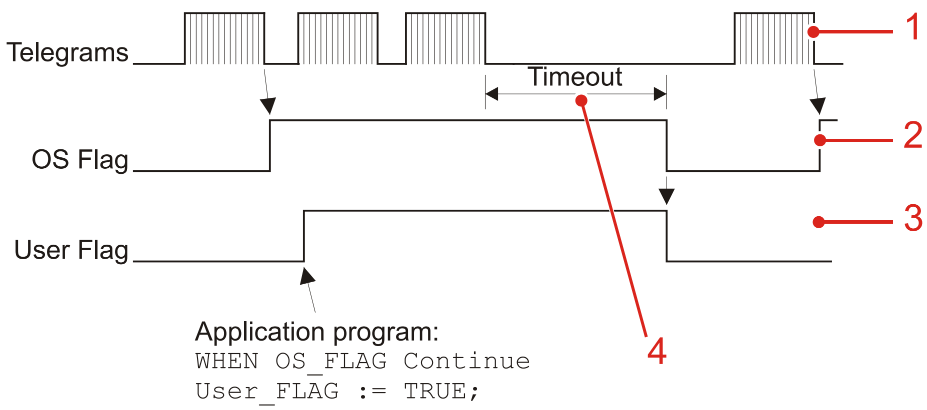
Operating principle

Introduction

The application program monitors the activity of a client communicating with a server of the device JC-940MC by means of two special flags and one special register per interface.

Overview

The illustration below shows the interdependence between interface activity and the two special flags, as well as the special register:



Number	Element	Description
1	Telegrams	The client places requests to the server.
2	OS flag	OS flag set by the device JC-940MC after receiving a request
3	User flag	You must set the user flag in the application program once the device has set the OS flag.This indicates that the connection has temporarily been disrupted even if the device resets the OS flag very quickly.
4	Timeout	Time of inactivity after which the OS resets both special flags. This time can be set in a special register.

Description

Interface activities are monitored as follows:

Step	Description
1	Enter the desired value into the timeout register of the application program. This way, the monitoring mode is activated as well.
2	When the controller receives the next telegram, the device JC-940MC sets the corresponding OS flag.
3	If the OS flag has been set, the application program also sets the respective user flag.
4	Each new telegram causes the timeout to restart.
5	If telegrams cease to arrive, both special flags are reset by the controller upon expiry of the timeout interval.

Step	Description
6	The application program detects that the device has reset the special flags and therefore takes appropriate action.
7	When further telegrams start arriving, the device sets the corresponding OS flag. The user flag, however, remains reset.

Programming

Registers/flags - Overview

For interface monitoring, the device provides the following registers and flags:

Timeout registers

Register	Interface	Application
R 203000	JetIP (Ethernet)	▪ Visualization ▪ Controller networking
R 203005	STX debugging (Ethernet)	▪ JetSym via Ethernet

Special flags

Flags	Interface	Application
F 2088	JetIP (Ethernet)	OS flag
F 2089		User flag
F 2098	STX debugging (Ethernet)	OS flag
F 2099		User flag

R 203000

Timeout in the case of JetIP (Ethernet)

This register contains the timeout for the JetIP server (Ethernet) in milliseconds.

Register properties	
Values	0 ... 2,147,483,647 [ms]
Value after reset	0 (monitoring disabled)

R 203005

Timeout in the case of STX debugging (Ethernet)

This register contains the timeout for the STX debug server (Ethernet) in milliseconds.

Register properties	
Values	0 ... 2,147,483,647 [ms]
Value after reset	0 (monitoring disabled)

Enabling the monitoring function

To enable monitoring of interface activities, proceed as follows:

Step	Action
1	Enter the desired value into the timeout register of this interface.
2	Wait until the controller has set the OS flag of this interface.
3	Set the corresponding user flag.

Detecting a timeout

To detect a timeout, proceed as follows:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Enable monitoring of interface activities (see above).</td></tr>
<tr><td>2</td><td colspan="2">Wait until the controller has reset the user flag of this interface.
Result: A timeout has occurred.</td></tr>
<tr><td rowspan="4">3</td><td colspan="2">Check the corresponding OS flag.</td></tr>
<tr><th>If ...</th><th>... then ...</th></tr>
<tr><td>... the OS flag is set,</td><td>... the connection was temporarily disrupted.</td></tr>
<tr><td>... the OS flag is reset,</td><td>... the connection is still disrupted.</td></tr>
</table>

9.8 Programming the local JX6-I/O submodules

Introduction

This chapter covers programming of the local JX6-I/O submodules at the PCI bus of the controller JC-940MC.

Contents

Submodule JX6-SB(-I)

Introduction

This chapter covers configuring of the JX2 system bus interface at the PCI bus of the controller JC-940MC.

Submodule JX6-SB(-I)

Submodules JX6-SB and JX6-SB-I take on converting the PCI bus to the JX2 system bus. The operating mode is Master - Slave.

Register

The JX2 system bus can be configured as follows:

- **Configuration and status registers of the JX6-SB(-I) submodule**

2	0	S	J	0	0	z	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zzz	Module register number	100 163

- **JX2 system bus registers**

2	0	S	J	0	2	z	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zzz	Module register number	000 999

Further literature

For more information on registers and engineering of the JX2 system bus, please turn to the JX6-SB user information.

Enhancements

Since revision 2.11.2 of the JX6-SB user information the following enhancements have been made:

- The value range I/O module numbers on the JX2 system bus
  - old: 0, 2 ... 32, 70 ... 79
  - new: 0, 2 ... 64, 70 ... 79
- The value range of the number of connected JX2-I/O and JX-SIO modules (register 2013)
  - old: 0 ... 41
  - new: 0 ... 64

- The module code was supplemented by the following items:
  - JX3 modules
  - JX-TP20-R
  - Festo CP-FB modules
  - LioN-S modules
  - Third-party I/O modules
  - Third-party slave modules

Digital JX6-I/O submodule JX6-IO16CB

Introduction

The digital input/output module is equipped with eight galvanically isolated 24 V inputs, and eight galvanically isolated 24 V outputs with a maximum current of 0.5 A each.

I/O numbers for local JX6-I/O submodules

I/O numbers for local JX6-I/O submodules connected to the JC-940MC consist of the following elements:

2	0	S	J	0	0	1	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zz	Module-specific I/O number	01 ... 08

Register numbers

In this chapter only the module register number is specified. To calculate the actually used register number, add the basic register number of the corresponding controller. The basic register number is made up of the module board number and the number of the submodule on the module board.

2	0	S	J	0	0	z	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zzz	Module register number	100 ... 999

Register overview

The local JX6-I/O module can be programmed using a set of module registers.

Register	Description
MR 100	State of the digital inputs
MR 101	Access to the digital outputs
MR 102	Error state of the digital outputs
MR 103	Control register
MR 104	Filter interval of inputs 1 to 4
MR 105	Filter interval of inputs 5 to 8

MR 100

Digital inputs

The input state can be read from MR 100. At a voltage greater than 12 V, the bit assigned to this input is set in this register (= 1). At lower voltages, the assigned bit is not set (= 0).

Meaning of the individual bits	
Bit 0	**Input 1**
Bit 1	**Input 2**
...	
Bit 7	**Input 8**

Module register properties	
Type of access	Read
Value after reset	State of the inputs

MR 101

Digital outputs

From MR 101, the output state can be read and the outputs can be energized and de-energized. When the bit is set, the assigned output is energized When the bit is reset, the assigned output is de-energized.

Meaning of the individual bits	
Bit 0	**Output 1**
Bit 1	**Output 2**
...	
Bit 7	**Output 8**

MR 102

Error state of the digital outputs

The error state of the digital outputs can be read from bit 7 of MR 102. The other bits of this register remain cleared (= 0). Even if the error is cleared, bit 7 remains set. Clear bit 7 by writing any value to this register.

Meaning of the individual bits	
Bit 7	**Collective error message of the digital outputs** 0 = No error 1 = Error in at least one output

Module register properties	
Values	0, 128
Value after reset	128

The following errors are recognized:

- The voltage of the 24 V output supply is too low.
- Short-circuit of the output
- Overloading of the output

MR 103

Control register

Via the control register, you can configure the JX6-I/O module.

Meaning of the individual bits		
Bit 0	**Bit 7 in output register MR 101**	
	0 =	The error state of the digital outputs is stored to bit 7 of MR 101.
	1 =	The state of output 8 is stored to bit 7 of MR 101 when reading.
Bit 1	**Behavior at errors of digital outputs**	
	0 =	In case of an error only the faulty output is switched off.
	1 =	In case of an error all outputs are switched off
Bit 3, 2	**Filter interval for the error signal of the outputs**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms

Module register properties	
Value after reset	1

MR 104

Filter interval of inputs 1 to 4

Write the filter interval for inputs 1 to 4 to this register. Two bits of this register have been assigned to each input.

Meaning of the individual bits		
Bit 1, 0	**Filter interval of input 1**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms
Bit 3, 2	**Filter interval of input 2**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms
Bit 5, 4	**Filter interval of input 3**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms
Bit 7, 6	**Filter interval of input 4**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms

MR 105

Filter interval of inputs 5 to 8

Write the filter interval for inputs 5 to 8 to this register. Two bits of this register have been assigned to each input.

Meaning of the individual bits		
Bit 1, 0	**Filter interval of input 5**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms
Bit 3, 2	**Filter interval of input 6**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms
Bit 5, 4	**Filter interval of input 7**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms
Bit 7, 6	**Filter interval of input 8**	
	00 =	1.5 ... 2.0 ms
	01 =	0.4 ... 0.5 ms
	10 =	approx. 0.1 ms
	11 =	approx. 0.02 ms

Related topics

- **Slot numbering of the module board and of the JX6-I/O module** (see page 172)

Combi module JX6-SV1

Introduction

The combi module is supplied by a potential-free relay contact, an analog voltage output and a counting input for incremental encoders or SSI absolute encoders.

Register numbers

In this chapter only the module register number is specified. To calculate the actually used register number, add the basic register number of the corresponding controller. The basic register number is made up of the module board number and the number of the submodule on the module board.

2	0	S	J	0	0	z	z	z

Element	Description	Value range
S	Module board number	1 ... 3
J	Number of the JX6-I/O submodule located on the module board	1 ... 2
zzz	Module register number	100 ... 999

Register overview

The local JX6-SV module can be programmed using a set of module registers. The functions of module registers 103 to 110 depends on whether an incremental encoder or an absolute encoder has been configured:

Register	Description
MR 100	Module ID
MR 101	Hardware configuration
MR 102	Analog output
MR 103	Strobe value/encoder value received
MR 104	Preset/pulse generator
MR 105	Count value
MR 106	Reference/offset
MR 107	Control register
MR 108	Status register
MR 110	Sampling timer

MR 100

Module ID

This register shows you, whether a JX6-SV1 has been connected.

Module register properties	
Type of access	Read
Value after reset	1 (with JX6-SV1)

MR 101

Hardware configuration

This register is for setting the module hardware.
Always write 0 to bits that have not been documented.

Meaning of the individual bits		
Bit 0	**Relay**	
	0 =	de-energized; pin 14 not connected with pin 15
	1 =	energized; pin 14 is connected with pin 15
Bit 2	**CHD signal on the counter chip**	
	0 =	Counter zeroing/reference has been disabled
	1 =	Counter zeroing/reference has been enabled
Bit 5	**Encoder type**	
	0 =	Incremental encoder
	1 =	SSI absolute encoder

MR 102

Analog output

After writing a value to this register, module JX6-SV1 outputs a voltage proportional to the written value at its analog output.

Module register properties	
Values	-32,767 ... 32,767 (-10 V ... +10 V)

MR 103

Incremental encoder: Strobe register

Module register properties	
Values	24 bits
Write	Writing the actual count value to this register
Read	Reading the count value written to the register last

Absolute encoder: Encoder value

This register shows the position value output by the SSI encoder.

Only if bit 0 is set in the status register (MR 108), there are valid values to be read.

Module register properties	
Type of access	Read
Values	32 bits

MR 104

Incremental encoder: Preset

When writing a value into register MR 105, this value is taken over as new count value.

Module register properties	
Values	24 bits

Absolute encoder: Clock generator

This register is for configuring the connected SSI encoder.

Meaning of the individual bits	
Bit 9 ... 0	**Frequency of the SSI pulse signal** = Module clock/(SSI clock * 2) - 1 Example: Module clock = 8 MHz; SSI clock = 500 kHz = 8,000,000 / (500,000 * 2) - 1 = 7
Bit 15 ... 10	**Data width of the SSI encoder in bits** = Pulse number * 2 + 1

Example:
Number of increments = 24
= 24 * 2 + 1 = 49

Module register properties	
Values	0 ... 65,535
Example	49 * 1,024 + 7 = 50,183

MR 105

Incremental encoder: Count value

This register lets you access the present count value of the incremental encoder.

Module register properties	
Values	24 bits
Read	Present count value
Write	The value of MR 104 is taken over into this register.

MR 106

Reference/offset

This register can be used both as reference or offset register, dependent on the settings in the control register.

Module register properties	
Values	24 bits

MR 107

Control register

This register lets you set the operating modes and input filters.

Meaning of the individual bits		
Bit 0	**Strobe**	
	0 =	Strobe not allowed
	1 =	Strobe allowed
Bit 1	**Reference strobe**	
	0 =	Reference strobe not allowed
	1 =	Reference strobe allowed (with /CHA * CHB * CHD)
Bit 2	**Comparison with reference value**	
	0 =	Comparison is not being executed
	1 =	The count value is compared with the reference value In case of equal values, bit 2 is set in MR 008.
Bit 3	**Counter zeroing**	
	0 =	No counter zeroing
	1 =	Once-only counter zeroing using CHC * CHD
Bit 4	**Repeated counter zeroing**	
	0 =	No counter zeroing
	1 =	Counter zeroing at each CHC * CHD
Bit 5	**Counting inputs**	
	0 =	Counter inputs are active
	1 =	Deactivating the counting inputs
Bit 6	**Set by reference**	
	0 =	Set-by reference deactivated
	1 =	The counter is loaded with the reference value with CHC * CHD
Bit 9, 8	**Pulse edge selection**	
	00 =	Pulse-direction mode
	01 =	Single evaluation
	10 =	Dual evaluation
	11 =	Four-fold evaluation
Bit 10	**Addition of offset**	
	0 =	Addition disabled
	1 =	Addition of an offset to a count and strobe value
Bit 11	**File format using an SSI encoder**	
	0 =	Binary
	1 =	Gray-format
Bit 12	**Parity using an SSI encoder**	
	0 =	No parity bit

	1 =	Parity check
Bit 13	**Polarity of the parity bit**	
	0 =	even parity
	1 =	odd parity
Bit 23 ... 14	**Input filters**	
	Minimum signal duration until a signal is detected: = (T_{sig} / T_{sys}) - 1	
	T_{sig}	Minimum signal duration
	T_{sys}	Period of a system cycle (125 ns at 8 MHz)

Module register properties	
Values	24 bits
Value after reset	0x300

MR 108

Status register

After read access to this register, all bits, except for bit 3, are cleared.

Meaning of the individual bits		
Bit 0	**Strobe**	
	Incremental encoder:	
	0 =	Strobe value not yet detected
	1 =	Strobing value detected
	Absolute encoder:	
	0 =	Invalid positioning value
	1 =	Valid positioning value
Bit 1	**Reference strobe**	
	1 =	Reference strobe detected
Bit 2	**Comparison with reference value**	
	1 =	Count value and reference value agree with each other
Bit 3	**Counter zeroing**	
	1 =	Counter zeroing has been executed
Bit 4	**Strobing value has been overwritten**	
	Incremental encoder:	
	1 =	The value in the strobe register has been overwritten
	Absolute encoder:	
	1 =	The position value has been overwritten

Bit 5	**Lost reference value**	
	1 =	A new reference value got lost, as the previous value has not been read out yet.
Bit 6	**Z track error**	
	1 =	Z track errors or maximum travel speed exceeded or illegal Z track change
Bit 7	**Parity error**	
	1 =	A parity error has occurred

MR 110

Sampling timer

This register value determines the intervals of reading the position value from the encoder. When the register content is "0", sampling will not be carried out. If you write a value other than "0" to the register, a sampling cycle is carried out when the set interval has elapsed.

Module register properties	
Values	0 ... 65,535
Calculation	= Module frequency/(sampling frequency * 2) - 1
Example	Module frequency = 8 MHz Sampling frequency = 1 kHz = 8,000,000 / (1,000 * 2) - 1 = 3,999

Related topics

- **Slot numbering of the module board and of the JX6-I/O module** (see page 172)

Sample program for local JX6-I/O submodules

Task

Cyclically read the position value of an absolute encoder with an SSI interface. Simultaneously physically establish moving lights.

Solution

For reading the position value, read module JX6-SV1. Physically establish the moving lights by using module JX6-IO16CB.

Sample configuration

This example is based on the following configuration:

- The module board is inserted in PCI slot 1 (S = 1).
- The JX6-IO16CB module is inserted in slot 1 of the module board (J = 1).
- The JX6-SV1 module is inserted in slot 2 of the module board (J = 2).

Program function

- In the task *Main*, the modules JX6-SV1 and JX6-IO16CB are initialized, and the cyclic task is started.
- Depending on a digital input the interval is set, at which each output of the moving lights is active.
- During the *While*-loop, the eight outputs for the moving lights are activated and deactivated successively.
- The cyclic task *Encode* stops the sampling timer. In the end, it reads the position value output by the encoder last.
- Then, the timer is started with an interval of 0.8 ms. After this, the JX6-SV1 transmits another position value from the SSI encoder so that at the next startup of the cyclic task, the result is ready for being fetched.
- Finally, the position value referring to the set format is masked.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

JetSym STX program

```
Type
    DIG_I8O8:
    Struct
        Inputs                  : Int;
        Outputs                 : Int;
        OutStatus               : Int;
        Control                 : Int;
        FilterI1to4             : Int;
        FilterI5to8             : Int;
    End_Struct;
```

```
    JX6_SV1:
    Struct
        ModuleIdentification : Int;
        Config               : Int;
        DAC                  : Int;
        CntValue             : Int;
        CntBitClock          : Int;
        CntDummy             : Int;
        CntRef               : Int;
        CntControl           : Int;
        CntStatus            : Int;
        IntMask              : Int;
        Timer                : Int;
    End_Struct;
End_Type;

Const
    // Gray-to-bin conversion; 2-MHz filter
    SSI_CONTROL         = 0x4800;
    // Delay 800 µs
    SSI_TIMER_1MS       = 3199;
End_Const;

Var
    DigitalIO         : DIG_I8O8    At %VL 201100100;
    Encoder           : JX6_SV1     At %VL 201200100;

    SSIdataMask       : Int;
    SSIsignBit        : Int;
    SSIsignExt        : Int;

    SSItimerValue     : Int;
    Position          : Int;

    OutputTime        : Int;
End_Var;


Function SetUpSSI(DataLen, Clock_kHz: Int): Int;
    Var
        i, j : Int;
    End_Var;

    // Configuration as an SSI encoder
    Encoder.Config := 32;

    i := 0xffffffff;
    j := 32 - DataLen;
    i >>= j;
```

```
        SSIdataMask := i;
        SSIsignExt := SSIdataMask Wxor 0xffffffff;
        SSIsignBit := DataLen - 1;

        // Data length of the SSI encoder in bits
        i := DataLen * 2 + 1;
        i <<= 10;

        // Frequency of the SSI pulse signal
        j := 4000 / Clock_kHz - 1;
        Encoder.CntBitClock := i Wor j;

        Encoder.CntControl := SSI_CONTROL;

        SSItimerValue := SSI_TIMER_1MS;

        // Start the timer
        Encoder.Timer := SSItimerValue;
    End_Function;


    Task Main Autorun
        Var
            OutMask:    Byte;
        End_Var;

        // 24-bit data length, 500 kHz
        SetUpSSI(24, 500);

        // Acknowledge the error state of the digital outputs
        DigitalIO.OutStatus := 0;

        TaskAllEnableCycle();

        Loop
            If BitSet(DigitalIO.Inputs, 7) Then
                OutputTime := T#200ms;
            Else
                OutputTime := T#100ms;
            End_If;

            OutMask := 1;
            While OutMask Do
                DigitalIO.Outputs := OutMask;
                Delay(OutputTime);
                OutMask <<= 1;
            End_While;
        End_Loop;
    End_Task;
```

```
Task Encode    Cycle T#1ms
    // Stop the sampling timer of the SSI encoder
    Encoder.Timer := 0;
    // Read the position value from the SSI encoder
    Position := Encoder.CntValue;
    // Restart timer
    Encoder.Timer := SSItimerValue;

    // Mask the data bits
    Position := Position Wand SSIdataMask;
    // Is a sign-extension required?
    If BitSet(Position, SSIsignBit) Then
        // Sign extension to 32 bits
        Position := Position Wor SSIsignExt;
    End_If;
End_Task;
```

9.9 E-mail

Introduction

The user creates template files for e-mails. Into these, the controller JC-940MC can enter variables for sending, if required. The controller sends e-mails to an e-mail server which will then forward the message.

This chapter gives a description on how to configure the e-mail feature in the JC-940MC controller, and on how the controller creates and sends e-mails.

Required programmer's skills

To be able to use the e-mail feature, the following skills are required:

- Since files are used to configure the e-mail feature, and e-mails as such are based on these files, the user must be familiar with the file system.
- The user must be familiar with IP networks.

Contents

9.9.1 Configuring the E-mail feature

Introduction

This chapter gives a description on how to configure the e-mail feature so as to allow sending e-mails from within the application program.

During the boot process, the JC-940MC reads out configuration data from the file **/EMAIL/email.ini**.

Prerequisites

For creating the configuration file, the following prerequisites must be fulfilled:

- The IP address of the e-mail server is known.
- If the IP address of the e-mail server is not known, name resolution through a DNS server must be possible - refer to *Using names for IP addresses*.
- The log-on and authentication parameters at the e-mail server are known.

To obtain this information contact your network administrator.

Contents

Configuration file "/EMAIL/email.ini"

Introduction

The configuration of the e-mail client in the controller is based on the contents of the file **/EMAIL/email.ini**. The JC-940MC reads the values during the boot process only.

File structure

This configuration file is a text file the entries of which are grouped into several sections.

- These sections are for entering values which are then used by the e-mail client.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Sections

The configuration file contains up to three sections. Section [SMTP] is mandatory. The user does not need to create the other sections unless they are actually required.

Section	Configuration values
[SMTP]	▪ IP address and port number of the SMTP server ▪ Log-on parameters
[POP3]	▪ IP address and port number of the POP3 server ▪ Log-on parameters
[DEFAULT]	▪ Name of an e-mail template file containing default values

Section [SMTP]

Introduction

In this section, specify the parameters for establishing a connection with the SMTP server.

Example:

```
[SMTP]
IP       = 192.168.40.1
PORT     = 25000
HELO     = JetControl_2
USER     = JetControl0815
PASSWORD = MyPassWord
```

Authentication

This type of authentication requires the JC-940MC to log on at the SMTP server before sending an e-mail. During the logon process USER and PASSWORD have to be entered. The JC-940MC supports the following authentication methods:

- LOG-ON
- PLAIN
- CRAM-MD5

Configuration values

IP	
In the given example	192.168.40.1
Description	IP address of the SMTP server; can also be specified as name.
Allowed values	■ > 1.0.0.0 ■ < 223.255.255.255
Illegal values	■ Network address ■ Broadcast address
In case of illegal value or missing entry	The e-mail feature is not available

PORT	
In the given example	25.000
Description	Port number of the SMTP server
Allowed values	■ > 0 ■ < 65.536
Illegal values	■ > 65.335
In case of missing entry	25

HELO	
In the given example	JetControl_2
Description	Name for logging on at the e-mail server
Allowed values	String of 63 characters max.

In case of missing entry	When sending the e-mail, the JC-940MC uses the entry contained in [FROM]

USER	
In the given example	JetControl0815
Description	Log-on name for SMTP authentication. If this entry exists, the entry PASSWORD is required, too.
Allowed values	String of 63 characters max.
In case of missing entry	SMTP authentification will not be carried out

PASSWORD	
In the given example	MyPassWord
Description	Log-on password for SMTP authentication. If this entry exists, the entry USER is required, too.
Allowed values	String of 63 characters max.
In case of missing entry	SMTP authentification will not be carried out

Section [POP3]

Introduction

In this section, specify the parameters for establishing a connection with the POP3 server.

Only in case the e-mail server requires authentication via POP3-before-SMTP, this section is required.

Example:

```
[POP3]
IP       = 192.168.40.1
PORT     = 25100
USER     = JetControl4711
PASSWORD = Pop3PassWord
```

Authentication

This type of authentication requires the JC-940MC to log on at the POP3 server. During the logon process USER and PASSWORD have to be entered. After that, the SMTP server allows to send e-mails for a given period of time (10 to 30 minutes).

Configuration values

IP	
In the given example	192.168.40.1
Description	IP address of POP3 server; can also be specified as name.
Allowed values	■ > 1.0.0.0 ■ < 223.255.255.255
Illegal values	■ Network address ■ Broadcast address
In case of illegal value or missing entry	POP3 log-in will not be carried out

PORT	
In the given example	25.100
Description	Port number of POP3 server
Allowed values	■ > 0 ■ < 65.536
Illegal values	■ > 65.335
In case of missing entry	110

USER	
In the given example	JetControl4711
Description	Log-on name for POP3 authentication. If this entry exists, the entry PASSWORD is required, too.
Allowed values	String of 63 characters max.
In case of missing entry	POP3 log-in will not be carried out

PASSWORD	
In the given example	Pop3PassWord
Description	Log-on password for POP3 authentication. If this entry exists, the entry USER is required, too.
Allowed values	String of 63 characters max.
In case of missing entry	POP3 log-in will not be carried out

Section [DEFAULT]

Introduction

In this section, specify the name of an e-mail template file which contains default settings for e-mails. If the respective section is not available in the respective e-mail template, the JC-940MC applies these default settings for sending an e-mail message.

Example

```
[DEFAULT]
MAILCFG = EmailDefaults.cfg
```

Related topics

- **Structure of template file** (see page 317)

Configuration file - Examples

Introduction

This section contains several examples of the e-mail configuration file **/EMAIL/email.ini**.

Minimum configuration

If no authentication is required and the default value is assigned to the IP port of the SMTP server, the configuration file must contain only the IP address of the SMTP server.

```
[SMTP]
IP       = 192.168.40.1
```

Authentication through POP3 Log-on

In case the e-mail server requires previous log-on through POP3 and an e-mail template containing default setting has been defined:

```
[SMTP]
IP       = 192.168.40.1

[POP3]
IP       = 192.168.40.1
USER     = JetControl4711
PASSWORD = Pop3PassWord

[DEFAULT]
MAILCFG = EmailDefaults.cfg
```

Authentification through SMTP

In case the e-mail server requires an encrypted authentication:

```
[SMTP]
IP       = 192.168.40.1
USER     = JetControl0815
PASSWORD = MyPassWord
```

9.9.2 Creating e-mails

Introduction

This chapter describes how to create an e-mail. Then, the application program sends these e-mails.

For each e-mail the user has to create an e-mail template file.

Contents

Name of the e-mail template file

Introduction

The name of an e-mail template file consists of a constant part of the name and a variable part. The variable part of the name allows the application program to choose various e-mails for sending.

File name

`email_#.cfg`

Part of the name	Description
email_	Name prefix which always remains constant
#	Number of the e-mail; value between 0 and 255
.cfg	Constant file extension

Storage location

E-mail template files have to be stored to the same directory on the internal flash disk as is the configuration file.

`/EMAIL`

Examples

```
email_0.cfg
email_37.cfg
email_255.cfg
```

Structure of the e-mail template file

Introduction

An e-mail template file is a text file which is divided into sections. When sending an e-mail, it is compiled based on the information contained in these sections.

E-mail template file

- Sections [FROM] and [TO] are mandatory. This information may be specified either in the e-mail to be sent or in the e-mail template file containing the default settings.
- All parameters in these sections can be tagged with real-time controller values (refer to *Inserting real-time controller values*).

[FROM]
Sender

[TO]
Addressee

[CC]
Additional addressee(s)

[SUBJECT]
Subject

[ATTACHMENT]
Complete path and file name

[MESSAGE]
E-mail message text

Sections

[FROM]	
Description	E-mail sender
Comments	Please check with your IT administrator which information has to be entered here.
Length	63 characters
Example	`[FROM]` `JetControl@jetter.de`

[TO]	
Description	E-mail addressee
Comments	Several addressees are separated by the semicolon “;”.
Length	255 characters
Example	`[TO]` `service@mydomain.com`

[CC]	
Description	Additional e-mail addressee(s)
Comments	Several addressees are separated by the semicolon ";".
Length	255 characters
Example	`[CC]` `service@mydomain.com;hotline@mydomain.com`

[SUBJECT]	
Description	Subject
Length	255 characters
Example	`[SUBJECT]` `Fatal Error`

[ATTACHMENT]	
Description	Complete name of the file to be attached
Comments	The attached file must be a text file.
Length	511 characters
Example	`[ATTACHMENT]` `/logfiles/error_report.log`

[MESSAGE]	
Description	E-mail message text
Comments	Text only message
Length	65,535 characters

Example	`[MESSAGE]`
	`Have a nice day !`
	`JetControl.`

Inserting real-time controller values

Introduction

Actual real-time controller values are integrated into parameter entries within the sections via tag functions. This way, the contents respectively states of registers, text registers, inputs, outputs and flags can be displayed.

Tag delimiters

All tags start and end with defined strings. Between these tag delimiters, the variables are defined.

Delimiter	String
Tag start	<JW:DTAG
Tag end	/>

Variable definition

The variable definition in a tag contains attributes which are used to set, for example, how the value of a variable is to be displayed:

name	
Description	Variable name
Comments	Code letter followed by the variable number
Example	`name="R1000023"`

type	
Description	Variable type of notation
Example	`type="REAL"`

format	
Description	Representation format
Comments	Refer to format definition
Example	`format="+0####.###"`

factor	
Description	Factor by which the real-time controller value is multiplied
Comments	Multiplication is executed prior to adding the offset.
Example	`factor="1.5"`

offset	
Description	Value which is added to the real-time controller value
Comments	Multiplication by the factor is executed prior to adding the value to the real-time controller value.
Example	`offset="1000"`

Format definition

You can define the representation of variables by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" sets the output of leading zeros. This applies to the register types INT, INTX and REAL.
- Prefix "+" sets the output of a sign. This applies to the register types INT and REAL.
- Prefixing a blank sets the output of a blank. This applies to the register types INT and REAL.

Registers/text registers

The variable name begins with a capital "R" followed by the register number.

The following types are possible:

Type	Notation
INT	Integer, decimal
INTX	Integer, hexadecimal
INTB	Integer, binary
BOOL	Register content = 0 --> Display: 0 Register content != 0 --> Display: 1
REAL	Floating point, decimal
STRING	Text register

Standard type: INT

Example:

```
JW:DTAG name="R1000250" type="REAL" format="+0####.###"
factor="3.25" offset="500" /
```

Result:

This instruction causes the contents of register 1000250 to be multiplied by 3.25 and then added to product 500. The result appears in the Web browser with sign and at least five integer positions before the decimal point. If need be, five leading zeros are added. Furthermore, three decimal positions are added.

Flags

The variable name begins with a capital "F" followed by the flag number.

The following types are possible:

Type	Notation
BOOL	Flag = 0 --> Display: 0 Flag = 1 --> Display: 1
STRING	Flag = 0 --> Display: FALSE Flag = 1 --> Display: TRUE

Standard type: BOOL

Example:

```
<JW:DTAG name="F100" type="STRING" format="#" />
```

Result:
The state of flag 100 is displayed as string "T" or "F".

Inputs

The variable name begins with a capital "I" followed by the input number.
The following types are possible:

Type	Notation
BOOL	Input = 0 --> Display: 0 Input = 1 --> Display: 1
STRING	Input = 0 --> Display: OFF Input = 1 --> Display: ON

Standard type: BOOL

Example:

```
<JW:DTAG name="I201200308" type="STRING" />
```

Result:
The state of input 201200308 on the CPU is displayed as string "ON" or "OFF".

Outputs

The variable name begins with a capital "O" followed by the output number.
The following types are possible:

Type	Notation
BOOL	Output = 0 --> Display: 0 Output = 1 --> Display: 1
STRING	Output = 0 --> Display: OFF Output = 1 --> Display: ON

Standard type: BOOL

Example:

```
<JW:DTAG name="O201100308" />
```

Result:
The state of output 201100308 is inserted as "1" or "0".

Access via pointer register

Access via pointer register is realized by inserting the capital letter "P" in front of the variable name. In each case the value of the variable is displayed whose number corresponds to the content of the register specified in the variable name.

Examples:

```
<JW:DTAG name="PR1000300" />
```

Result: The content of the register is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PF1000300" />
```

Result: The state of the flag is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PI1000300" />
```

Result: The state of the input is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PO1000300" />
```

Result: The state of the output is displayed whose number is contained in register 1000300.

Access via pointer register and offset

To specify the number of the variable to be displayed, it is also possible to add a constant value or another register content to the pointer register value

Examples:

```
<JW:DTAG name="PR1000300 + 100" />
```

Result: The content of the register is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PR1000300 + R1000100" />
```

Result: The content of the register is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PF1000300 + 100" />
```

Result: The state of the flag is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PF1000300 + R1000100" />
```

Result: The state of the flag is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PI1000300 + 100" />
```

Result: The state of the input is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PI1000300 + R1000100" />
```

Result: The state of the input is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PO1000300 + 100" />
```

Result: The state of the output is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PO1000300 + R1000100" />
```

Result: The state of the output is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

9.9.3 Sending an e-mail

Introduction	This chapter gives a description on how to send previously created e-mails from within the application program. When sending an e-mail from the application program, the device JC-940MC creates the e-mail based on the e-mail template file and inserts variable values if required.
Processing within the application program	Sending an e-mail may take considerable time. Therefore, other tasks of the application program are processed while an e-mail is being sent. Only a function call via e-mail is not possible. While an e-mail of a task is being sent, all other tasks which invoke the e-mail function are therefore blocked until this operation is completed.
System function 110	As of JetSym 5.0, system function 110 is outdated. Instead, apply JetSym STX function `EMailSend()`.
JetSym STX function EMailSend()	The JetSym STX function EMailSend() has been described in detail in the online help of JetSym. Declaration of functions: `Function EmailSend(Const Ref FileName: String): Int;`

9.9.4 Registers

Introduction

This chapter gives a description of those registers from which you can query the status of e-mail processing.

Contents

Overview of registers

Introduction

The device JC-940MC makes the registers available from which you can query the status of e-mail processing.

Register overview

Register	Description
202930	Web status
292932	IP address of the SMTP server
292933	IP address of the POP3 server
292934	Port number of the SMTP server
292935	Port number of POP3 server
292937	Status of e-mail processing
292938	ID of the task that is just sending an e-mail

Register description

R 202930

Web status

The Web status register displays all available functions in bit-coded mode.

Meaning of the individual bits		
Bit 0	**FTP server**	
	1 =	available
Bit 1	**HTTP server**	
	1 =	available
Bit 2	**E-mail**	
	1 =	available
Bit 3	**Data file function**	
	1 =	available
Bit 4	**Modbus/TCP**	
	1 =	existing
Bit 5	**Modbus/TCP**	
	1 =	available

Module register properties	
Type of access	Read
Value after reset	Depending on options purchased

R 292932

IP address of the SMTP server;

From this register the IP address of the SMTP server can be seen as it has been specified in the file **/EMAIL/email.ini**.

Module register properties	
Type of access	Read
Value after reset	Depending on configuration
Takes effect	Once R 202930.2 = 1

R 292933

IP address of POP3 server

From this register the IP address of the POP3 server can be seen as it has been specified in the file **/EMAIL/email.ini**.

Module register properties	
Type of access	Read
Value after reset	Depending on configuration
Takes effect	Once R 202930.2 = 1

R 292934

Port number of the SMTP server

From this register the port number of the SMTP server can be seen as it has been specified in the file **/EMAIL/email.ini**.

Module register properties	
Type of access	Read
Value after reset	Depending on configuration
Takes effect	Once R 202930.2 = 1

R 292935

Port number of POP3 server

From this register the port number of the POP3 server can be seen as it has been specified in the file **/EMAIL/email.ini**.

Module register properties	
Type of access	Read
Value after reset	Depending on configuration
Takes effect	Once R 202930.2 = 1

R 292937

Status of e-mail processing

With the help of this register the user can track the e-mail status.

Module register properties		
Values	0	No e-mail is being sent
	1	Parameters are being handed over to the e-mail client of the JC-940MC
	2	E-mail is being compiled and connection with the server is being established
	3	E-mail has been sent to the server
Type of access	Read	

R 292938

Task ID (e-mail)

The ID of the task that is just sending an e-mail can be seen from this register

Module register properties		
Values	0 ... 99	Task ID
	255	No task is sending an e-mail
Value after reset	255	
Type of access	Read	

9.10 Sorting data

Introduction

This chapter describes system function 50. This system function is used to trigger the sorting algorithm provided by the operating system.

Application

For sorting data in controller registers by their value.

The sort algorithm is provided by the operating system of the controller. The data to be sorted are indirectly addressed through a descriptor using parameter 1.

System function 50

As of JetSym 5.0, system function 50 is outdated. Instead, apply JetSym STX function `QSort()`.

JetSym STX function QSort()

The JetSym STX function QSort() has been described in detail in the online help of JetSym.

Declaration of functions:

```
Function QSort(DataPtr: Int, ElementCnt: Int, ElementSize: Int,
SortOffset: Int, SortType: STXBASETYPE, SortMode: QSORTMODE): Int;
```

9.11 Modbus/TCP

Introduction

This chapter describes the functions of the Modbus/TCP server and client integrated into JC-940MC.

Required programmer's skills

To be able to use the functions described in this chapter, the following skills are required:

- The user must be familiar with Modbus/TCP and the supported commands.
- The user must be familiar with IP networks.

Contents

9.11.1 Modbus/TCP server

Introduction

After successful start of the Modbus/TCP server, an external client can access registers, flags, inputs and outputs.

This chapter covers the addressing process and describes the commands supported by the Modbus/TCP server.

Number of possible connections

Four connections may be opened at the same time.

Restriction

Modbus/TCP only supports transmission of registers with a width of 16 bits. From this follows, that only the 16 least significant bits are transmitted when 32-bit registers are sent.

When assigning incoming register values to the internal 32-bit registers no sign extension will be carried out.

Contents

Addressing

Introduction

The addresses which have been received via Modbus/TCP can be modified locally in the server. For this purpose, three registers have been provided. The basic addresses for accessing registers, inputs and outputs are entered into these registers. Then, the address contained in the Modbus/TCP frame specifies the address with reference to the basic address.

R 272702

Register offset

The basic address for accessing registers via Modbus/TCP is entered into R 272702.

Module register properties	
Value after reset	1000000

R 272704

Input offset

The basic address for accessing inputs via Modbus/TCP is entered into R 272704.

Module register properties	
Value after reset	100000000

R 272705

Output offset

The basic address for accessing outputs via Modbus/TCP is entered into R 272705.

Module register properties	
Value after reset	100000000

Example 1

The Modbus/TCP server on the JetControl receives from a Modbus/TCP client the command **read multiple registers** starting from register number 100. The number of registers to be read is 5. Register 272702 *Register Offset* contains the value 1000000.

Hence, registers 1000100 through 1000104 will be read.

Example 2

The Modbus/TCP server on the JetControl receives from a Modbus/TCP client the command **read input discretes** specifying input number 205 and the instruction to read this input. Register 272704 *Input Offset* contains the value 201100000.

Hence, input 201100205, for example of a peripheral module JX2-IO16, will be read.

Example 3

The Modbus/TCP server on the JetControl receives from a Modbus/TCP client the command **write coils** specifying output number 205 and the instruction to set this output. Register 272705 *Output Offset* contains value 201200000.

Hence, output 201100205, for example of a peripheral module JX2-IO16, will be activated.

Supported commands - Class 0

fc 3

read multiple registers

Reading register sets

The starting register number within JC-940MC is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*.

fc 16

write multiple registers

Writing register sets

The starting register number within JC-940MC is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*.

Supported commands - Class 1

fc 1 — **read coils**

Reading outputs

The output number within the JC-940MC is calculated as follows: Output number specified in the command plus the content of register 272705 *Output Offset*.

fc 2 — **read input discretes**

Reading inputs

The input number within JC-940MC is calculated as follows: Input number specified in the command plus the content of register 272704 *Input Offset*.

fc 4 — **read input registers**

Reading inputs blockwise in 16-bit words.

The starting register number within JC-940MC is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*.

fc 5 — **write coil**

Enabling/disabling an individual output

The output number within the JC-940MC is calculated as follows: Output number specified in the command plus the content of register 272705 *Output Offset*.

fc 6 — **write single register**

Entering values into the 16 least significant bits of a register

The starting register number within JC-940MC is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*.

Supported commands - Class 2

fc 15

force multiple coils

Enabling/disabling several outputs

The output number within the JC-940MC is calculated as follows: Output number specified in the command plus the content of register 272705 *Output Offset*.

fc 23

read/write registers

Reading/writing registers simultaneously

The starting register number within the JC-940MC is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*.

9.11.2 Modbus/TCP client

Introduction

The Modbus/TCP client included in JC-940MC supports only Class 0 Conformance.

In this class, commands for reading and writing multiple registers are used. Up to 125 registers with a width of 16 bits can be transmitted in one frame.

As protocol ID "0" is used. Assignment of sent and received frames is carried out using the transaction ID.

This chapter describes how to carry out noncyclical or cyclical transmission to a Modbus/TCP server using system functions.

Number of possible connections

Connections to eleven different Modbus/TCP servers may be opened at the same time.

Noncyclical data transmission

System functions 65 and 67 *reading registers*, as well as 66 and 68 *writing registers* are used to establish a noncyclical transmission channel to a Modbus/TCP server.

These system functions establish a connection to the specified Modbus/TCP server, transmit the desired data and clear down the connection.

If RemoteScan has already established a connection for cyclical data transmission, this connection will be used. Setting up and clearing down the connection is, therefore, not required.

Cyclical data transmission

Cyclical data transmission is made through the configurable function `RemoteScan`. The inputs and outputs 20001 through 36000 that are combined in the 16-bit registers 278000 through 278999 are cyclically transmitted from and to the Modbus/TCP servers.

Only one connection is established to each Modbus/TCP server (IP address and port) irrespective of the number of communication units which have been configured on this server.

If several communication units are configured on one Modbus/TCP server, accesses are serialized since servers often do not support **command pipelining**. If several servers have been configured, communication is carried out in parallel.

Combined inputs and outputs

Register	Inputs and outputs
278000	20001 ... 20016
278001	20017 ... 20032
278002	20033 ... 20048
...	...
278999	35985 ... 36000

These registers and the inputs and outputs mapped to them are merely storage cells within the RAM. There is no direct mapping to the hardware. Therefore, it is not defined whether inputs or outputs are mapped to a register. Assignment is made not until configuration in the communication units takes place.

Unit ID

The instruction header of a Modbus/TCP telegram contains a *Unit ID*. The Unit ID is not evaluated by Modbus/TCP devices, as they can be addressed without ambiguity by their IP address. Therefore, in the case of system functions 65, 66 and 80 always value "1" is sent.

Converters from Modbus/TCP to Modbus RTU use the *Unit ID* for addressing the Modbus RTU servers. Therefore, the corresponding system functions for reading and writing registers (system functions 67 and 68), as well as for initializing RemoteScan (system function 85) have been provided. These system functions can be used to set the Unit ID.

Restriction

Modbus/TCP only supports transmission of registers with a width of 16 bits. From this follows, that only the 16 least significant bits are transmitted when 32-bit registers are sent.

When assigning incoming register values to the internal 32-bit registers no sign extension will be carried out.

Outdated system functions

As of JetSym 5.0, the system functions are outdated. Instead, apply the corresponding JetSym STX functions.

JetSym STX functions

This is a comparison between the system functions and the corresponding JetSym STX functions.

System function	Corresponding JetSym STX function
60	Function ModbusCRCgen(FramePtr: Int, Length: Int): Int;
61	Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int;
65/67	Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int;
66/68	Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int;
80/85	Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int;
81	Function RemoteScanStart(Protocol: Int): Int;
82	Function RemoteScanStop(Protocol: Int): Int;

9.11.3 Modbus/TCP client with STX variables

Introduction

The Modbus/TCP client included in JC-940MC supports only Class 0 Conformance.

In this class, commands for reading and writing multiple registers are used. One frame transmits up to 125 registers of 16 bits width.

As protocol ID, "0" is used. Assignment of transmitted and received frames is carried out using the transaction ID.

This chapter describes how to carry out noncyclical or cyclical transmission to a Modbus/TCP server using STX functions.

Number of possible connections

Connections to eleven different Modbus/TCP servers may be opened at the same time.

Noncyclical data transmission

Functions `ModbusReadReg()` and `ModbusWriteReg()` are used to establish a noncyclical transmission channel to a Modbus/TCP server.

These functions copy data between registers of a Modbus/TCP server and STX variables. They establish a connection to the specified Modbus/TCP server, transmit the desired data and clear down the connection again.

If RemoteScan has already established a connection for cyclical data transmission, this connection will be used. Setting up and clearing down the connection is, therefore, not required.

Cyclical data transmission

Cyclical data transmission is made through the configurable function `RemoteScanConfig()`. The data are cyclically transmitted from and to the Modbus/TCP servers by means of STX variables.

Only one connection is established to each Modbus/TCP server (IP address and port) irrespective of the number of communication units which have been configured on this server.

If several communication units are configured on one Modbus/TCP server, accesses are serialized since servers often do not support **command pipelining**. If several servers have been configured, communication is carried out in parallel.

Unit ID

Converters from Modbus/TCP to Modbus RTU use the *Unit ID* for addressing the Modbus RTU servers. For this reason, the Unit ID can be set.

JetSym STX functions

The JetSym STX functions have been described in detail in the online help of JetSym.

System function	Corresponding JetSym STX function
60	Function ModbusCRCgen(FramePtr: Int, Length: Int): Int;
61	Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int;
65/67	Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int;
66/68	Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int;
80/85	Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int;
81	Function RemoteScanStart(Protocol: Int): Int;
82	Function RemoteScanStop(Protocol: Int): Int;

9.12 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 JC-940MC 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 of the controller. 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

9.12.1 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 JC-940MC. 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 data

Technical data of the user-programmable IP interface:

Feature	Description
Number of connections	20
Maximum data size	4,000 byte

Restrictions

While the controller JC-940MC 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

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 JC-940MC processes this function in the following steps:

Step	Description
1	The device JC-940MC closes all open connections of the user-programmable IP interface.
2	The device JC-940MC initializes all OS-internal data structures of the user-programmable IP interface.

Related topics

- **Establishing a connection** (see page 346)
- **Terminating a connection** (see page 355)
- **Sending data** (see page 350)
- **Receiving data** (see page 352)

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	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:

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">The device JC-940MC tries to establish a TCP/IP connection via port 46000 to the network client with IP address 192.168.75.123.</td></tr>
<tr><td rowspan="4">2</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>the network client has accepted the connection,</td><td>the function is terminated and a positive value is returned as handle for further access to the connection.</td></tr>
<tr><td>the connection could not be established and the timeout of 10 seconds has not elapsed yet,</td><td>step 1 is carried out.</td></tr>
<tr><td>an error has occurred or the timeout has elapsed,</td><td>the function is terminated and a negative value is returned.</td></tr>
</table>

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 JC-940MC 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 JC-940MC 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 JC-940MC 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 JC-940MC 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 JC-940MC 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 355)
- **Sending data** (see page 350)
- **Receiving data** (see page 352)
- **Initializing the user-programmable IP interface** (see page 345)

Sending data

Introduction

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

Via UDP/IP connection of a server data cannot be sent, but only received.

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	Outcome of the function `ConnectionCreate()`
`IPAddr`	Valid IP address	Required only for UDP/IP client
`IPPort`	Valid IP port	Required only for UDP/IP client
`SendData`	Address of the data block to be sent	
	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 is not required anymore and can be ignored in the function.

In the following situations, the task is not processed further after issuing the command:

- Until the data are sent and receiving them has been confirmed.
- Until an error has occurred.

Using this function with a UDP/IP client

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

Functioning principle with a UDP/IP client

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 the command until the data are sent.

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

Related topics

- **Initializing the user-programmable IP interface** (see page 345)
- **Establishing a connection** (see page 346)
- **Terminating a connection** (see page 355)
- **Receiving data** (see page 352)

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 cannot be received, but only sent.

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	Outcome 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 = infinitely

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 is not required any more and can be ignored in the function.

In the following situations, the task is not processed further after issuing the command:

- Until the data have been received
- Until an error has occurred

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

The device JC-940MC processes this function in the following steps:

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">The device JC-940MC waits until data have been received, but no longer than the specified timeout.</td></tr>
<tr><td rowspan="3">2</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>the timeout has elapsed or the connection has been terminated,</td><td>the function is exited and an error message is issued.</td></tr>
<tr><td>data have been received,</td><td>they are copied to the receiving buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3.</td></tr>
<tr><td rowspan="2">3</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>more data have been received than could have been copied into the receiving buffer,</td><td>these are buffered by the JC-940MC to be fetched by further function calls.</td></tr>
<tr><td>4</td><td colspan="2">The function is exited and the number of data, which have been copied into the receiving buffer, is returned.</td></tr>
</table>

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 the command:

- Until all data have been received.
- Until an error has occurred.

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

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">The device JC-940MC waits until all data of a datagram have been received, but no longer than the specified timeout.</td></tr>
<tr><td rowspan="3">2</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>the timeout has elapsed or the connection has been terminated,</td><td>the function is exited and an error message is issued.</td></tr>
<tr><td>data have been received,</td><td>they are copied to the receiving buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3.</td></tr>
<tr><td rowspan="2">3</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>... more data have been received than could be copied into the receiving buffer - that is, if the sent datagram is too large,</td><td>... these data are discarded.</td></tr>
<tr><td>4</td><td colspan="2">The sender's IP address and IP port are transferred into the variables which are given along with the data.</td></tr>
<tr><td>5</td><td colspan="2">The function is exited and the number of data, which have been copied into the receiving buffer, is returned.</td></tr>
</table>

Related topics

- **Initializing the user-programmable IP interface** (see page 345)
- **Establishing a connection** (see page 346)
- **Terminating a connection** (see page 355)
- **Sending data** (see page 350)

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	Outcome 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 346)
- **Sending data** (see page 350)
- **Receiving data** (see page 352)
- **Initializing the user-programmable IP interface** (see page 345)

9.12.2 Registers

Introduction

This chapter describes the registers of the JC-940MC from which the current connection list of the user-programmable IP interface can be read out. 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

Controller	Basic register number	Register numbers
JC-24x	10290	10290 ... 10297
JM-D203-JC24x	10290	10290 ... 10297
JC-340, JC-350, JC-360, JC-360MC	350000	350000 ... 350007
JC-940MC	350000	350000 ... 350007
JVM-407, JVM-407B	350000	350000 ... 350007
JVM-104	350000	350000 ... 350007

Determining the register number

In this chapter only the last figure of a register number is specified. To calculate the actually used module register number, add the basic register number of the corresponding device.

Register 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
MR 5	State
MR 6	Number of sent bytes
MR 7	Number of received bytes

Register 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. Read access to this register is used to display whether the following registers contain connection details.

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

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

Module register properties	
Values	0 ... 65,535

MR 5

State

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

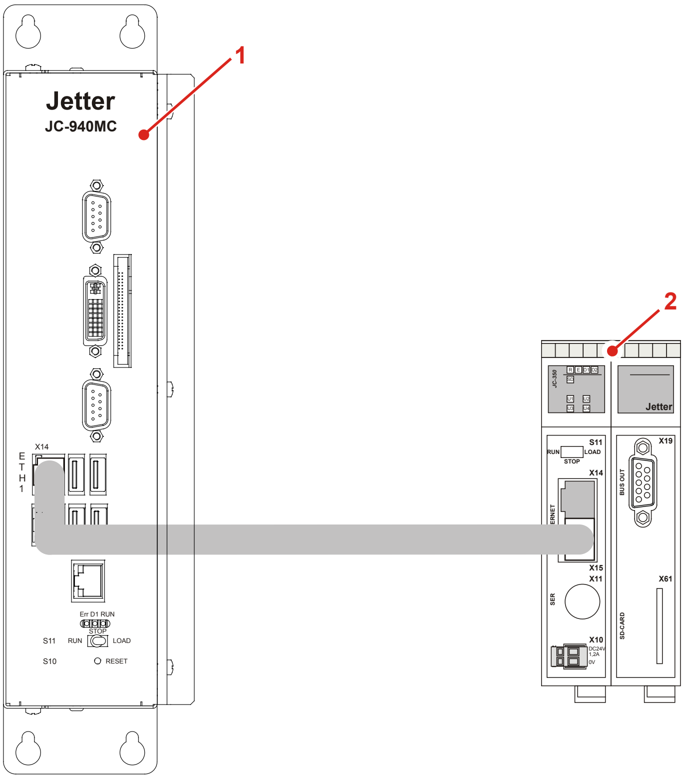
9.12.3 Sample programs

Introduction

This chapter contains sample programs for implementing a server and a corresponding client which will use TCP/IP for communication.

Sample configuration

The examples in this chapter are based on the following configuration:



Number	Part	Description
1	JC-940MC	Controller
2	JC-350	Controller

Due to the platform-independent implementation of the user-programmable IP interface these sample programs can be used for other configurations without modification.

Use case

The use case is a Jetter AG controller or HMI communicating with a device from a third-party manufacturer. The user-programmable IP interface lets you make protocol adjustments in the application program of the Jetter AG device.

Contents

Server

Task

A server is to receive a data block with a given number of characters and to return the received data to the client.

Solution

Programming a server for the user-programmable IP interface. The server communicates via TCP/IP.

Sample configuration

This example is based on the configuration described under *Sample configuration* (see page 361).

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

JetSym STX program

```
Const
  TCP_PORT = 52100;
  MSG_LEN  = 4000;
End_Const;

Var
  // Handle as a return value of the function "Establishing
  // a connection"
  ConnHandle      : Int;

  // Transmit buffer
  SendBuf         : Array[MSG_LEN] Of Char;
  // Receive buffer
  RecvBuf         : Array[MSG_LEN] Of Char;

  ResConnInit     : Int;
  ResConnCreate   : Int;
  ResConnReceive  : Int;
  ResConnSend     : Int;
  ConnTimeOut     : Int;
  RecvTimeOut     : Int;

  // Amount of receive errors
  RecvErrors      : Int;
  // Amount of transmit errors
  SendErrors      : Int;
  // Counter of correct communication transmissions
  CommCnt         : Int;
  AmountToReceive : Int;
```

```
  // Dummy
  NotUsed          : Int;
End_Var;


Task TCPserver Autorun

  Var
    RecvTimer      : Timer;
    ReceiveCnt     : Int;
  End_Var;

  // Timeout for connection
  ConnTimeOut := T#5s;
  // Timeout when receiving a data packet
  RecvTimeOut := T#5s;

  // Amount of data to be received
  AmountToReceive := MSG_LEN;

  // Terminate all connections, initialize the data structures
  ResConnInit := ConnectionInitialize();

  If ResConnInit >= 0 Then

    Trace('Server running.$n');

    While (True) Do

      // Connection attempt
      ResConnCreate := ConnectionCreate
                         ( CONNTYPE_SERVER,
                           IPTYPE_TCP,
                           0,
                           TCP_PORT,
                           ConnTimeOut );

      If ResConnCreate > 0 Then

        Trace('Connection established.');
        // Save the return value (handle)
        ConnHandle := ResConnCreate;

        // Repeat, while the connection exists
        Loop

// Set up a timeout for completely transmitting the data packet
          TimerStart(RecvTimer, RecvTimeOut * 5);
// Initialize the counter for the received data
          ReceiveCnt := 0;
```

```
    // Repeat, until all expected data have been received
    // Otherwise, retry until timeout
            While ReceiveCnt < AmountToReceive
                And Not TimerEnd(RecvTimer) Do

              ResConnReceive := ConnectionReceiveData
                                    ( ConnHandle,
                                      NotUsed,
                                      NotUsed,
                                      RecvBuf[ReceiveCnt],
                                      SizeOf(RecvBuf),
                                      RecvTimeOut );

              If ResConnReceive > 0 Then
                // A value has been received, increment the counter
                ReceiveCnt := ReceiveCnt + ResConnReceive;
              Else
                // Error during reception
                ResConnReceive := -1;
                // Increment the error counter value
                Inc(RecvErrors);
                // Exit the loop
                Exit;
              End_If;
            End_While;

  // At this point, implement the server function
  // In this example, the received data
  // are returned
            If ReceiveCnt Then
              // Copy the data from the receiving buffer to the
transmit buffer
              MemCopy(SendBuf,RecvBuf,SizeOf(SendBuf));
              ResConnSend := ConnectionSendData
                                ( ConnHandle,
                                  0,
                                  0,
                                  SendBuf,
                                  ReceiveCnt );
              If ResConnSend < 0 Then
                // Increment the error counter value
                Inc(SendErrors);
              End_If;
            End_If;

            If ResConnSend >= 0 And ResConnReceive >= 0 Then
              // No error --> Counter OK to increment
              Inc(CommCnt);
            Else
              // Exit the loop
              Exit;
            End_If;
```

```
        End_Loop;

        If ConnHandle > 0 Then
          // Terminate the connection
          ConnectionDelete(ConnHandle);
          // Handle is no longer valid
          ConnHandle := 0;
          Trace('Connection closed.$n');
        End_If;

      End_If;

  // Wait shortly for another connection to be tried
      Delay(T#3s);

    End_While;

  Else
    Trace('ConnectionInitialize() failed, server stopped !$n');
  End_If;

End_Task;
```

Related topics

- **Client** (see page 367)

Client

Task

A client is to send a data block with a given number of characters and to return the data received from the server.

Solution

Programming a client for the user-programmable IP interface. The client communicates via TCP/IP.

Sample configuration

This example is based on the configuration described under *Sample configuration* (see page 361).

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-350, OS version 1.18

For other sample programs, refer to JetSym online help.

JetSym STX program

```
Const
  TCP_ADDR = IP#192.168.10.210;
  TCP_PORT = 52100;
  MSG_LEN  = 4000;
End_Const;

Var
  // Handle as a return value of the function "Establishing
  // a connection"
  ConnHandle       : Int;
  // Transmit buffer
  SendBuf          : Array[MSG_LEN] Of Char;
  // Receive buffer
  RecvBuf          : Array[MSG_LEN] Of Char;
  ResConnInit      : Int;
  ResConnCreate    : Int;
  ResConnReceive   : Int;
  ResConnSend      : Int;
  ConnTimeOut      : Int;
  RecvTimeOut      : Int;

  // Amount of receive errors
  RecvErrors       : Int;
  // Amount of transmit errors
  SendErrors       : Int;
  // Counter of correct communication transmissions
  CommCnt          : Int;
  AmountToReceive : Int;
  SendDelay        : Int;

  // Dummy
```

```
  NotUsed         : Int;
End_Var;


Task TCPclient Autorun

  Var
    RecvTimer     : Timer;
    ReceiveCnt    : Int;
  End_Var;

  // Timeout for connection
  ConnTimeOut := T#5s;
  // Timeout when receiving a data packet
  RecvTimeOut := T#5s;

  // Amount of data to be received
  AmountToReceive := MSG_LEN;
  SendDelay       := T#500ms;

  // Terminate all connections, initialize the data structures
  ResConnInit := ConnectionInitialize();

  If ResConnInit >= 0 Then

    Trace('Client running.$n');

    While (True) Do

      // Connection attempt
      ResConnCreate   := ConnectionCreate
                         ( CONNTYPE_CLIENT,
                           IPTYPE_TCP,
                           TCP_ADDR,
                           TCP_PORT,
                           ConnTimeOut );

      If ResConnCreate > 0 Then

        Trace('Connection established.');
        // Save the return value (handle)
        ConnHandle := ResConnCreate;

        // Repeat, while the connection exists
        Loop

          ResConnSend := ConnectionSendData
                         ( ConnHandle,
                           0,
                           0,
```

```
                                  SendBuf,
                                  AmountToReceive );
                If ResConnSend < 0 Then
                  // Increment the error counter value
                  Inc(SendErrors);
                End_If;

        // Timeout for completely transmitting the data packet
        // to be set up
                TimerStart(RecvTimer, RecvTimeOut * 5);
                // Initialize the counter for the received data
                ReceiveCnt := 0;

        // Repeat, until all expected data have been received
        // Otherwise, retry until timeout
                While ReceiveCnt < AmountToReceive
                    And Not TimerEnd(RecvTimer) Do

                  ResConnReceive := ConnectionReceiveData
                                        ( ConnHandle,
                                          NotUsed,
                                          NotUsed,
                                          RecvBuf[ReceiveCnt],
                                          SizeOf(RecvBuf),
                                          RecvTimeOut );

                  If ResConnReceive > 0 Then
                    // A value has been received, increment the counter
                    ReceiveCnt := ReceiveCnt + ResConnReceive;
                  Else
                    // Error during reception
                    ResConnReceive := -1;
                    // Increment the error counter value
                    Inc(RecvErrors);
                    // Exit the loop
                    Exit;
                  End_If;
                End_While;

                If ResConnSend >= 0 And ResConnReceive >= 0 Then
                  // No error --> Counter OK to increment
                  Inc(CommCnt);
                  Delay(SendDelay);
                Else
                  // Exit the loop
                  Exit;
                End_If;

              End_Loop;
```

```
        If ConnHandle > 0 Then
          // Terminate the connection
          ConnectionDelete(ConnHandle);
          // Handle is no longer valid
          ConnHandle := 0;
          Trace('Connection closed.$n');
        End_If;

      End_If;

  // Wait shortly for another connection to be tried
      Delay(T#3s);

    End_While;

  Else
    Trace('ConnectionInitialize() failed, client stopped !$n');
  End_If;

End_Task;
```

Related topics

- **Server** (see page 363)

10 Automatic copying of controller data

Introduction

This chapter describes the AutoCopy function which allows to copy data within the controller and/or between controller, the connected expansion modules and a controller within the network. To this end, you can create a command file which is then stored along with the data to a USB flash drive, for example. This command file is automatically processed by the controller during the boot process.

Functions

The AutoCopy function executes the following functions:

- Storing registers and flags to a file
- Restoring registers and flags from a file
- Creating directories
- Deleting directories
- Copying files
- Deleting files

Areas of application

There are the following application scopes for the AutoCopy function:

- Where remote control is not possible
- Where there is no PC on site
- If the operator is not able or should not be allowed to make modifications to the plant

The following actions can be taken using the AutoCopy function:

- Modification to the application program
- Modification to the application data
- Modification to the controller configuration
- Operating system update (controller and modules on the system bus)
- Duplication of a control system

Prerequisites

For automatic copying of controller data, the following prerequisites must be fulfilled:

- The programmer is familiar with the file system.
- In the configuration file **config.ini** of the controller, the path and file name of the command file in the section *[FILES]* must be entered in **AutoCopyIni**. For access to the configuration file, at least system rights are required.
- The command file and other required files are available.

Example: config.ini

This is an example of a configuration file config.ini with an entry AutoCopyIni.

```
;JC-940MC System Configuration
;Copyright (c) 2009 by Jetter AG, Ludwigsburg, Germany

[IP]
Address     = 192.168. 10.209
SubnetMask = 255.255.255.  0
DefGateway  =   0.  0.  0.  0
DNSServer   = 192.168. 10.244

[HOSTNAME]
SuffixType = 0
Name       = JetControl940MC

[PORTS]
JetIPBase = 50000
JVMDebug  = 52000

[FILES]
AutoCopyIni = /USB1/autocopy.ini
```

Designation

In this description *Complete Name* means the name of the file or directory including the complete path.

Contents

10.1 Operating principle

Introduction

This chapter describes how to start and execute the AutoCopy function.

Contents

Activating the AutoCopy feature

Introduction

The AutoCopy function can only be executed when the controller is booting (i.e. after startup).

Prerequisites

You have created the command file and stored it to the respective directory.

	Value	Remarks
File name	autocopy.ini	All lower case letters
Directory	/USB1/	Root directory on the USB flash drive

Activating the AutoCopy feature

To start the AutoCopy function, proceed as follows:

Step	Action
1	Switch the controller off.
2	Set the mode selector S11 to *LOAD* position.
3	Switch the controller on.
4	Wait for the yellow LED **D1** to be lit and for the green LED **RUN** to flash slowly by approximately 1 Hz.
⇨	**Result:** The controller executes the AutoCopy function.
5	Wait for the yellow LED **D1** to be lit and for the green LED **RUN** to flash fast by approximately 4 Hz.
⇨	**Result:** The AutoCopy function is set.

Executing AutoCopy commands

Introduction

During the boot process in AutoCopy mode the controller executes the commands contained in the command file.

Restrictions

In AutoCopy mode the following restrictions of controller functions apply:

- The controller does not execute the application program.
- Communication with the controller is not possible.
- After terminating the AutoCopy function, restart of the controller is required.

Executing AutoCopy commands

The OS of the controller processes the AutoCopy function in the following steps:

Step	Description
1	The controller opens, for example, the file **/USB1/autocopy.ini**, which is specified within the entry **AutoCopyIni** that belong to the configuration file **/System/config.ini**.
2	The controller reads the values from section *[OPTIONS]*.
3	The controller reads the command and its parameters from the section *[COMMAND_1]*, processes it and writes the results, if any, into the log file.
4 ... n	The controller processes the other commands in ascending order up to the number given in section *[OPTIONS]*.
n+1	The controller calculates the statistic values for all command results and writes them into the log file.

LEDs of the JC-940MC in AutoCopy mode

During the boot process of the controller, the OS status LEDs indicate the following:

Step	Description			
1	**ERR**	**D1**	**RUN**	**State**
	ON	OFF	OFF	Reset
2	**ERR**	**D1**	**RUN**	**State**
	OFF	ON	OFF	Initialization
3	**ERR**	**D1**	**RUN**	**State**
	OFF	ON	1Hz	The AutoCopy function is executed.
4a	**ERR**	**D1**	**RUN**	**State**
	OFF	ON	4Hz	AutoCopy function is completed; no errors occurred

Step	Description			
4b	ERR	D1	RUN	State
	ON	ON	4Hz	AutoCopy function is completed; errors occurred

Terminating AutoCopy mode

Introduction

Only a restart of the controller terminates the AutoCopy mode.

Prerequisites

Processing the AutoCopy command is completed.

Terminating AutoCopy mode

To terminate the AutoCopy mode, proceed as follows:

Step	Action
1	Switch the controller off.
2	Set the mode selector to *RUN* or STOP position.
3	Switch the controller on.

Result: The controller is rebooting.

10.2 The file "autocopy.ini"

Introduction

This chapter covers the structure of the file **autocopy.ini** and the available commands.

File structure

This command file of the AutoCopy function is a text file the entries of which are grouped into several sections.

- In these sections you can set values then used by the AutoCopy function.
- You can insert blank lines as required.
- Introduce comment marks by "!", "#" oder ";".

Sections

The command file has two section types:

- In the *[OPTIONS]* section, you can make default settings. This file is unique.
- In the *[COMMAND_#]* section, you can specify the commands that are to be executed. The number of command section is limited to a value of 128.

Contents

Section [OPTIONS]

Introduction

In the [OPTIONS] section, you can make default settings. This section exists only once, preferably at the beginning of the file.

Example

```
[OPTIONS]
CommandCount = 14
LogFile = /USB1/autocopy.log
LogAppend    = 1
```

Elements of this section

The section consists of the following elements:

CommandCount	
In the given example	14
Description	Number of command sections that follow
Allowed values	> = 0
Illegal values	< 0
In case of illegal value or missing entry	0

LogFile	
In the given example	/USB1/autocopy.log
Description	Complete name of the log file
Allowed values	■ All allowed file names ■ Directory exists
Illegal values	■ Incorrect filename ■ Non-existent directory
In case of illegal value or missing entry	The device does not create a log file.

LogAppend	
In the given example	1
Description	Defines whether a new log file is to be created or it is to be appended to an existing one
Allowed values	■ 0 = Delete file which may exist and create a new one ■ 1 = Append file to an existing one. If no file exists, the device creates a new log file
Illegal values	■ < 0 ■ > 1
In case of illegal value or missing entry	The device re-creates a log file.

Command sections

Introduction

In these sections you can specify commands which are then executed by the AutoCopy function.

Example

```
[COMMAND_1]
Command        = DirCreate
Path           = /Homepage
ErrorAsWarning = 1

[COMMAND_2]
Command        = FileCopy
Source         = /USB1/Index.htm
Destination    = /Homepage/index.htm

[COMMAND_3]
Command        = FtpConnect
ServerAddr     = 192.168.123.45
UserName       = admin
Password       = admin
```

Section names

The names of the sections consist of the `COMMAND_` string followed by a value. The value is between one and the value of the `CommandCount` entry from section `[OPTIONS]`.

Processing commands

The AutoCopy function processes the commands in order of their section names:

- Starting with the command under section `[COMMAND_1]`
- Ending with the command under the section with the value of entry `CommandCount` from section `[OPTIONS]`
- Each command section may only contain one command. Thus, you have to create an individual section for each command.

Troubleshooting

When an error occurs while a command is being processed, the device makes a corresponding entry in the log file. For each command the user can set, whether the device is to enter the error into the log file as `Error` or as `Warning`. Make this setting by the optional parameter `ErrorAsWarning`.

ErrorAsWarning	Entry into the log file
Parameter does not exist	Error
ErrorAsWarning = 0	Error
ErrorAsWarning = 1	Warning

File names

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system supports this, the function parameter for the file located on the FTP server can also contain the path to this file. If this feature is not supported, the corresponding directory must be set beforehand using the command `FtpDirChange(...)`.
- The file system of a JC-940MC supports both options.

Available commands in the local file system

The following commands are available for access to the local file system:

Command = DirCreate

Description	Creates a subdirectory
Parameter name	Path
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ Higher-level directories are available
Illegal values	■ Invalid directory name ■ Non-existent higher-level directory ■ Name of an already existing directory
In the event of an illegal value	The device does not generate the directory. It enters the error into the log file.

Example	`[COMMAND_1]`
	`Command = DirCreate`
	`Path     = /sub1`
	
	`[COMMAND_2]`
	`Command = DirCreate`
	`Path     = /sub1/sub2`

Command = DirRemove

Description	Removes a subdirectory
Parameter name	Path
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ The directory is empty
Illegal values	■ Invalid directory name ■ Directory is not empty
In the event of an illegal value	The device does not remove the directory. It enters the error into the log file.

Example	`[COMMAND_8]`
	`Command = DirRemove`
	`Path     = /sub1/sub2`

Command = FileCopy

Description	This command is for copying a file
Parameter name 1	Source
Parameter value 1	Complete name of the source file

Parameter name 2	Destination
Parameter value 2	Complete name of the destination file
Allowed values	■ All allowed file names ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent source file ■ Non-existent destination directory
In the event of an illegal value	The device does not copy the file. It enters the error into the log file.

Example	`[COMMAND_1]`
	`Command     = FileCopy`
	`Source      =`
	`/USB1/OS/JC-940MC_1.03.0.20.os`
	`Destination = /System/OS/op_system.os`
	
	`[COMMAND_2]`
	`Command     = FileCopy`
	`Source      = /USB1/Manual.pdf`
	`Destination = /sub1/Manual.pdf`

Command = FileRemove

Description	Delete file
Parameter name	Path
Parameter value	Complete name of the file
Allowed values	All allowed file names
Illegal values	Incorrect filename
In the event of an illegal value	The device does not delete the file. It enters the error into the log file.

Example	`[COMMAND_5]`
	`Command = FileRemove`
	`Path    = /sub1/Manual.pdf`

Command = DaFileRead

Description	Transferring register values and flag states from a data file to the JC-940MC
Parameter name	DaFile
Parameter value	Complete name of the data file
Allowed values	All allowed file names for data files
Illegal values	■ Incorrect filename ■ Nonexistent data file
In the event of an illegal value	The data are not transmitted to the device JC-940MC. The JC-940MC enters the error into the log file.

Example	`[COMMAND_12]`
	`Command     = DaFileRead`
	`DaFile      = /USB1/Data/MyTestData.da`

Command = DaFileWrite

Description	This command is for storing register values and flag states to a data file

Parameter name 1	DaFile
Parameter value 1	Complete name of the data file
Allowed values	■ All allowed file names for data files ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent destination directory
In the event of an illegal value	The device does not generate the data file. It enters the error into the log file.
Parameter name 2	Append
Parameter value 2	Defines whether a new data file is to be created or it is to be appended to an existing one.
Allowed values	■ 0 = Delete the data file which may exist and create a new data file ■ 1 = Append the file to an existing one. If no file exists, the device creates a new data file
Illegal values	■ < 0 ■ > 1
In the event of an illegal value	A new data file will be created.
Parameter name 3	Type
Parameter value 3	Defines whether registers or flags are to be stored.
Allowed values	■ Register ■ Flag
Illegal values	Values other than "Register" or "Flag"
In the event of an illegal value	The device does not generate the data file. It enters the error into the log file.
Parameter name 4	First
Parameter value 4	Number of the first register or flag
Allowed values	All valid numbers from the memory area of the corresponding JC-940MC
Illegal values	Invalid numbers
In the event of an illegal value	The device does not generate the data file. It enters the error into the log file.
Parameter name 5	Last
Parameter value 5	Number of the last register or flag
Allowed values	All valid numbers from the memory area of the corresponding JC-940MC which are equal to or greater than the value for "First".
Illegal values	■ Invalid numbers ■ Numbers less than "First"
In the event of an illegal value	The device stores only one value (First).

Example

```
[COMMAND_11]
Command     = DaFileWrite
DaFile      = /USB1/MyTestData2.da
Append      = 0
Type        = Register
First       = 1000000
Last        = 1000000

[COMMAND_12]
Command     = DaFileWrite
DaFile      = /USB1/MyTestData2.da
Append      = 1
Type        = Flag
First       = 10
Last        = 20

[COMMAND_13]
Command     = DaFileWrite
DaFile      = /USB1/MyTestData2.da
Append      = 1
Type        = Register
First       = 1000001
Last        = 1000999
```

Available commands for access via FTP

The following commands are available for access via network using FTP:

Command = FtpConnect

Description	Establishing a connection to an FTP server
Parameter name 1	ServerAddr
Parameter value 1	IP address or name of FTP server
Allowed values	■ IP address of the FTP server ■ Name which can be resolved through DNS
Illegal values	■ IP address other than that of the FTP server ■ Name which cannot be resolved
Parameter name 2	UserName
Parameter value 2	User name for logging on at the FTP server
Parameter name 3	Password
Parameter value 3	Password for logging on at the FTP server
In the case of a illegal values	The device does not establish the connection. It enters the error into the log file.

Example	`[COMMAND_1]`
	`Command = FtpConnect`
	`ServerAddr = 192.168.123.45`
	`UserName   = admin`
	`Password   = admin`

Restriction	Only one connection with an FTP server can be established at a time. The device terminates the existing connection, before a connection to another FTP server is established.

Command = FtpFileRead

Description	Copying file from FTP server into the local file system
Parameter name 1	ServerFile
Parameter value 1	Complete name of the source file in the FTP server
Parameter name 2	ClientFile
Parameter value 2	Complete name of the destination file in the local file system
Allowed values	■ All allowed file names ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent source file ■ Non-existent destination directory
In the event of an illegal value	The device does not copy the file. It enters the error into the log file.

Example	`[COMMAND_8]`
	`Command      = FtpFileRead`
	`ServerFile   = /app/cantest/cantest.es3`
	`ClientFile   = /USB1/cantest3.es`

Command = FtpFileWrite

Description	Copying the file from the local file system into the file system of the FTP server
Parameter name 1	ServerFile
Parameter value 1	Complete name of the destination file in the FTP server
Parameter name 2	ClientFile
Parameter value 2	Complete name of the source file in the local file system
Allowed values	■ All allowed file names ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent source file ■ Non-existent destination directory
In the event of an illegal value	The device does not copy the file. It enters the error into the log file.

Example	`[COMMAND_5]`
	`Command     = FtpFileWrite`
	`ServerFile  = /System/OS/op_system.os`
	`ClientFile  =`
	`/USB1/OS/JC-940_1.03.0.20.os`

Command = FtpFileRemove

Description	Deleting a file from the FTP server
Parameter name	ServerFile
Parameter value	Complete filename
Allowed values	All allowed file names
Illegal values	Incorrect filename
In the event of an illegal value	The device does not delete the file. It enters the error into the log file.

Example	`[COMMAND_9]`
	`Command = FtpFileRemove`
	`ServerFile = /sub1/Manual.pdf`

Command = FtpDirChange	
Description	Changing the working directory in FTP server
Parameter name	ServerDir
Parameter value	Complete directory name
Allowed values	All valid directory names
Illegal values	Invalid directory name
In the event of an illegal value	The device does not switch the directory. It enters the error into the log file.

Example	`[COMMAND_12]`
	`Command = FtpDirChange`
	`ServerDir   = /Data/MyTestData`

Command = FtpDirCreate	
Description	Create a subdirectory in the FTP server
Parameter name	ServerDir
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ Higher-level directories are available
Illegal values	■ Invalid directory name ■ Non-existent higher-level directory ■ Name of an already existing directory
In the event of an illegal value	The device does not generate the directory. It enters the error into the log file.

Example	`[COMMAND_6]`
	`Command     = FtpDirCreate`
	`ServerDir   = /Data/MyTestData`

Restriction	If a directory with the corresponding path is specified as function parameter, all directories up to the directory to be created must exist. Recursive creation of several directories is not supported.

Command = FtpDirRemove	
Description	Clear the subdirectory in the FTP server
Parameter name	ServerDir
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ The directory is empty
Illegal values	■ Invalid directory name ■ Directory is not empty
In the event of an illegal value	The device does not remove the directory. It enters the error into the log file.

Example	`[COMMAND_8]`
	`Command     = FtpDirRemove`
	`ServerDir   = /Data/MyTestData`

Example of a command file

Task

The controller JC-940MC controls an already existing plant. In this plant, you want to enhance the functions.

To this end, the following modifications are required:

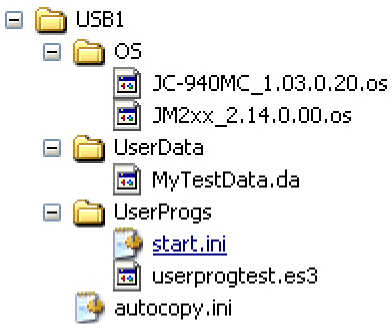
- Operating system update for the controller
- Operating system update for a connected JetMove 2xx
- New application program
- New values for some of the registers

Solution

You copy the required files to a USB flash drive and create a command file for the AutoCopy function. Then you send this USB flash drive along with a short instruction sheet to the plant operator. Once the update is completed, the operator returns the card.

USB flash drive - Content

The following illustration shows the directory structure and the files on the USB flash drive from the controller's point of view before the AutoCopy function is executed:



Following execution the log file **autocopy.log** has been added.

Command file

```
[OPTIONS]
CommandCount = 6
LogFile      = /USB1/autocopy.log
LogAppend    = 0

# update operating system of controller
[COMMAND_1]
Command     = FileCopy
Source      = /USB1/OS/JC-940MC_1.03.0.20.os
Destination = /System/OS/op_system.os

# update operating system of controller
[COMMAND_2]
Command     = FileCopy
Source      = /USB1/OS/JM2xx_2.14.0.00.os
Destination =
/System/PCI-Slot1/SB-Module2/JX2-Slave02/OS/system.os
```

```
# create user program directories
# probably already present - but to be sure ...
[COMMAND_3]
Command        = DirCreate
Path           = /app
ErrorAsWarning = 1

[COMMAND_4]
Command     = DirCreate
Path        = /app/userprogtest

# copy user program start file
[COMMAND_5]
Command     = FileCopy
Source      = /USB1/UserProgs/start.ini
Destination = /app/start.ini

# copy user program
[COMMAND_6]
Command     = FileCopy
Source      = /USB1/UserProgs/userprogtest.es3
Destination = /app/userprogtest/userprogtest.es3

# set registers and flags
[COMMAND_7]
Command     = DaFileRead
DaFile      = /USB1/UserData/MyTestData.da
```

10.3 Log file

Introduction

This chapter covers the structure and contents of the log file into which the device enters the outcome of the respective commands.

Contents

File contents

Introduction

The log file is a plain text file. By making an entry into the command file, you define whether a log file is to be created or whether the device is to append the entries to an existing log file.

Example

```
JetControl AutoCopy log file 07.11.2012 09:14:09

 1: Ok      - FileCopy   /USB1/OS/JC-940MC_1.03.0.20.os
                         /System/OS/op_system.os (187400 byte)
 2: Ok      - FileCopy   /USB1/OS/JM2xx_2.14.0.00.os

/System/PCI-Slot1/SB-Module2/JX2-Slave02/OS/system.os
(567000 byte)
 3: Warning - DirCreate  /app
 4: Ok      - DirCreate  /app/userprogtest
 5: Ok      - FileCopy   /USB1/UserProgs/start.ini
                         /app/start.ini (63 byte)
 6: Ok      - FileCopy   /USB1/UserProgs/userprogtest.es3
                         /app/userprogtest/userprogtest.es3
                         (169 byte)
 7: Error   - DaFileRead /USB1/UserData/MyTestData.da

Command statistics:
  Total  : 7
  Ok     : 5
  Warning: 1
  Error  : 1
```

Description

When for each executed AutoCopy function a section is appended to an existing log file, the log file consists of three elements:

- The header contains date and time.
- The following block contains information on the executed commands.
- Finally, it contains short statistics on command processing.

In the above example an error occurs when trying to create the directory */app* as this directory already exists. This error enters the device as a warning. When the device reads the DA file, an error also occurs. This error enters the device as an error into the log file.

10.4 Data files

Introduction

This chapter covers data files where register and flag values are stored.

Contents

File format

Format

The data file consists of the following elements:

- Pure text file
- Each entry must be in a separate line of text
- Each line must be terminated by carriage return/line feed
- Comment lines must be preceded by ";"
- Each data file is to start with the entry *SD1001*.

Data lines

A data line consists of the following elements:

- ID of the variable at the beginning of the line
- Now follows the number of the variable separated by a blank or tab
- Then follows the value of the variable separated by a blank or tab

Variable ID	Variable type
FS	Flags
RS	Integer register
QA	Floating-point registers

Example

```
SD1001
; Data File - Jetter AG
;
; Registers 1000000 ... 1000005
RS     1000000     12345
RS     1000001     2
RS     1000002     -1062729008
RS     1000003     502
RS     1000004     50
RS     1000005     3
QS     1009000     3.14
;
; Flags 10 ... 13
FS     10     0
FS     11     1
FS     12     1
FS     13     0
```

11 Operating system update

Introduction

Jetter AG are continuously striving to enhance the operating systems for their controllers and peripheral modules. Enhancing means adding new features, upgrading existing functions and fixing bugs.

This chapter describes how to perform an operating system update for a system equipped with a JC-940MC controller.

Downloading an operating system

You can download operating systems from the Jetter AG **homepage http://www.jetter.de**. You get to the OS files for download at *Industrial Automation - Support - Downloads* or by clicking on the quick link *Operating System Download* on the website of the corresponding controller or module.

JC-9xx system - Devices

The following devices within a system equipped with the controller JC-940MC allow an OS update:

- Controller JC-940MC
- JX2 slave modules on a JX2 system bus
- Bus nodes on a JX2 system bus
- Bus nodes on a JX3 system bus
- Analog modules on a JX3 system bus
- Digital inputs and outputs on a JX3 system bus

Contents

11.1 Updating the operating system of the controller

Introduction

This chapter describes how to update the OS of the controller JC-940MC. There are several options to transfer the OS file to the controller:

- From within the programming tool JetSym
- Via FTP connection
- From the application program

Contents

OS update by means of JetSym

Introduction

The programming tool JetSym offers an easy way to transfer an OS file to the JC-940MC.

Prerequisites

- An OS file for the JC-940MC is available.
- A UDP/IP and a TCP/IP connection between programming tool and IP port of the JC-940MC is possible.
  The number of the device in the configuration memory is set as IP basic port number for JetIP communication.
- The OS is running.
- The controller must not be de-energized during the OS update process.

Updating the OS

To update the OS, proceed as follows:

Step	Action
1	Select in the JetSym menu **Build** the menu item **Update OS.** In the **Advanced Configuration** dialog of the Hardware Manager, click on the button **Update OS**. **Result:** The file selection dialog opens.
2	Select the new OS file here. **Result:** In JetSym, a confirmation dialog opens.
3	Launch the OS upload by clicking the button **Yes**.
4	Wait until the update process is completed.
5	To activate the transferred OS, re-boot the device.

Operating system update via FTP

Introduction

Using an FTP client an OS file can be transferred to the device JC-940MC.

Prerequisites

- An OS file for the JC-940MC is available.
- An FTP connection to the device must be possible.
- The login parameters for a user with administrator or system rights are at hand.
- The operating system of the JC-940MC is running.
- The controller must not be de-energized during the OS update process.

Updating the OS

To update the OS, proceed as follows:

Step	Action
1	Establish an FTP connection to the JC-940MC.
2	Log in with administrator or system rights.
3	Navigate to the directory */System/OS*.
4	Transfer the OS file.
5	Wait until the update process is completed.
6	Close the FTP connection.
7	To activate the transferred OS, re-boot the device.

Automatic OS update from USB flash drive

Reference: An automatic OS update of the controller from the USB flash drive can be carried out using the AutoCopy function. For a detailed description, turn to the chapter *AutoCopy* (see page 371).

Operating system update from within the application program

Introduction

The file functions included in the STX language allow to carry out a program-controlled OS update of a JC-940MC from within an OS file.

Prerequisites

- An OS file must be available in the file system of the JC-940MC.
- The operating system of the JC-940MC and the application program are running.
- The controller must not be de-energized during the OS update process.

Updating the OS

To start an OS update out of the application program, proceed as follows:

Step	Action
1	Open the OS file in read-only mode.
2	Open a file with any name and the extension ***.os** in the directory */System/OS* in write mode.
3	Read the data out of the OS file.
4	Write these data to the target file.
5	Close both files.
6	To activate the transferred OS, for example by writing to the system command register, re-boot the device.

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

Sample program

```
Var
    SourceName:             String;
    DestinationName:        String;
    UpdateIt:               Bool;
End_Var;

//*****************************************************
// Name:      FileCopy
// param[in]  SrcName          name of source file
// param[in]  DstName          name of destination file
// return     >= 0             size of source file
// return     < 0              error
// brief      copies a file
//*****************************************************
Function FileCopy(Ref SrcName: String,
                  Ref DstName: String):Int;

    Var
```

```
    SrcFile, DstFile:    File;
    FileBuffer:          Array[1000] Of Byte;
    Result:              Int;
    ReadSize:            Int;
    WriteSize:           Int;
    FileSize:            Int;
End_Var;

Result := 0;
FileSize := 0;
// open source file for reading
If FileOpen(SrcFile, SrcName, 'r') Then
    // open destination file for writing
    If FileOpen(DstFile, DstName, 'w') Then
        // read first block of data
        ReadSize := FileRead(SrcFile,
                             FileBuffer,
                             SizeOf(FileBuffer));
        While ReadSize <> 0 Do
            // write read data to destination file
            WriteSize := FileWrite(DstFile,
                                   FileBuffer,
                                   ReadSize);
            If WriteSize <> ReadSize Then
                // write error
                Result := -3;
                Exit;
            End_If;
            Inc(FileSize, WriteSize);
            // read next block of data
            ReadSize := FileRead(SrcFile,
                                 FileBuffer,
                                 SizeOf(FileBuffer));
        End_While;
        // close both files
        FileClose(SrcFile);
        FileClose(DstFile);
    Else
        // can't open destination file
        FileClose(SrcFile);
        Result := -2;
    End_If;
Else
    // can't open source file
    Result := -1;
End_If;
```

```
    If Result < 0 Then
        FileCopy := Result;
    Else
        FileCopy := FileSize;
    End_If;
End_Function;

//*****************************************************
// 1. Enable Tracing in JetSym
// 2. Put source file name into 'SourceName'
// 3. Set flag 'UpdateIt'
//*****************************************************
Task OSupdate Autorun
    Var
        ResCopy:    Int;
    End_Var;

    DestinationName := '/System/OS/OperatingSystem.os';
    Loop
        UpdateIt := False;
        When UpdateIt Continue;
        ResCopy := FileCopy(SourceName,
                            DestinationName);
        Trace('Result : ' + IntToStr(ResCopy) + '$n');
    End_Loop;
End_Task;
```

11.2 OS update of a module

Introduction

This chapter describes how an OS update of a module connected to the system bus of the controller JC-940MC is carried out. There are several options to transfer the OS file to the module:

- From within the programming tool JetSym
- Via FTP connection
- From a USB flash drive
- From the application program

Contents

OS update by means of JetSym

Introduction

The programming tool JetSym offers an easy way to transfer an OS file to a module connected to the system bus of the controller.

Prerequisites

- An OS file for the module is available.
- A UDP/IP and a TCP/IP connection between programming tool and IP port of the controller is possible (timeout 4,000 ms min.).
  The number of the controller in the configuration memory is set as IP basic port number for JetIP communication.
- The operating system is running.
- The controller has put the system bus into operation without any errors.
- The boot loader or the operating system of the module is running.
- The controller must remain energized during the OS update process.

Updating the OS

To update the OS, proceed as follows:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">In JetSym, click on the menu Build and select item Update OS..., or click in the Advanced Configuration window of the Hardware Manager on OS Update.
Result: The file selection dialog opens.</td></tr>
<tr><td>2</td><td colspan="2">Select the new OS file here.
Result: In JetSym, a confirmation dialog opens.</td></tr>
<tr><td>3</td><td colspan="2">Confirm by clicking Yes.
Result: JetSym opens an input box for entering the interface type and module number.</td></tr>
<tr><td>4</td><td colspan="2">Enter the number of the module board (1), the number of the submodule (1 or 2) and the module number (2 ... 63).
Launch the OS upload by clicking the button Update.</td></tr>
<tr><td>5</td><td colspan="2">Wait until the update process is completed.</td></tr>
<tr><td rowspan="3">6</td><td>If ...</td><td>... then ...</td></tr>
<tr><td>you wish to update other modules,</td><td>proceed with step 1.</td></tr>
<tr><td>... do not you wish to update other modules,</td><td>reboot the controller to launch the new operating system.</td></tr>
</table>

Operating system update via FTP

Introduction

Using an FTP client an OS file can be transferred to a module connected to the system bus of the controller.

Prerequisites

- An OS file for the module is available.
- An FTP connection to the controller must be possible.
- The login parameters for a user with administrator or system rights are at hand.
- The operating system is running.
- The controller has put the system bus into operation without any errors.
- The boot loader or the operating system of the module is running.
- The controller must remain energized during the OS update process.

Updating the OS

To update the OS, proceed as follows:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Establish an FTP connection to the controller.</td></tr>
<tr><td>2</td><td colspan="2">Log in with administrator or system rights.</td></tr>
<tr><td>3</td><td colspan="2">Navigate to the OS directory of the module.
Example: /System/PCI-Slot1/SB-Module1/JX2-Slave02/OS oder /System/PCI-Slot1/SB-Module2/JX3-Module05/OS</td></tr>
<tr><td>4</td><td colspan="2">Transfer the OS file into this directory.</td></tr>
<tr><td>5</td><td colspan="2">Wait until the update process is completed.</td></tr>
<tr><td rowspan="3">6</td><th>If ...</th><th>... then ...</th></tr>
<tr><td>you wish to update other modules,</td><td>proceed with step 3.</td></tr>
<tr><td>... do not you wish to update other modules,</td><td>close the FTP connection. Then reboot the controller to launch the new operating system.</td></tr>
</table>

Automatic OS update from USB flash drive

Reference:	An automatic OS update of the a JX2 or JX3 module from the USB flash drive can be carried out using the AutoCopy function. For a detailed description, turn to the chapter *AutoCopy* (see page 371).

Operating system update from within the application program

Introduction

The file functions included in the STX language allow to transfer an OS file to a module on the JX2 or JX3 system bus of the controller.

Prerequisites

- An OS file for the module is available in the file system of the controller.
- The operating system of the controller and the application program are running.
- The controller has put the system bus into operation without any errors.
- The boot loader or the operating system of the module is running.
- The controller must remain energized during the OS update process.

Updating the OS

To update the OS of a module, proceed as follows:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Open the OS file in read-only mode.</td></tr>
<tr><td>2</td><td colspan="2">Open a file with any name and the extension *.os in the OS directory of the module in write mode.
Example: /System/PCI-Slot1/SB-Module1/JX2-Slave02/OS/system.os
oder /System/PCI-Slot1/SB-Module2JX3-Module05/OS/Irgendwas.os</td></tr>
<tr><td>3</td><td colspan="2">Read the data out of the OS file.</td></tr>
<tr><td>4</td><td colspan="2">Write these data to the target file for the module.</td></tr>
<tr><td>5</td><td colspan="2">Close both files.</td></tr>
<tr><td rowspan="3">6</td><td>If ...</td><td>... then ...</td></tr>
<tr><td>you wish to update other modules,</td><td>proceed with step 1.</td></tr>
<tr><td>... do not you wish to update other modules,</td><td>reboot the controller to launch the new operating system.</td></tr>
</table>

Software versions

The sample program has been tested for compliance with the following software versions:

- JetSym version 5.1
- Controller JC-940MC, OS version 1.05

For other sample programs, refer to JetSym online help.

Sample program

```
Var
    SourceName:             String;
    DestinationName:        String;
    UpdateIt:               Bool;
End_Var;
```

```
//*************************************************
// Name:        FileCopy
//! \param[in]  SrcName        name of source file
//! \param[in]  DstName        name of destination file
//! \return     >= 0           size of source file
//! \return     < 0            error
//! \brief      copies a file
//*************************************************
Function FileCopy(Ref SrcName: String,
                  Ref DstName: String):Int;
    Var
        SrcFile, DstFile:    File;
        FileBuffer:          Array[1000] Of Byte;
        Result:              Int;
        ReadSize:            Int;
        WriteSize:           Int;
        FileSize:            Int;
    End_Var;

    Result := 0;
    FileSize := 0;
    // open source file for reading
    If FileOpen(SrcFile, SrcName, 'r') Then
        // open destination file for writing
        If FileOpen(DstFile, DstName, 'w') Then
            // read first block of data
            ReadSize := FileRead(SrcFile,
                                 FileBuffer,
                                 SizeOf(FileBuffer));
            While ReadSize <> 0 Do
                // write read data to destination file
                WriteSize := FileWrite(DstFile,
                                       FileBuffer,
                                       ReadSize);
                If WriteSize <> ReadSize Then
                    // write error
                    Result := -3;
                    Exit;
                End_If;
                Inc(FileSize, WriteSize);
                // read next block of data
                ReadSize := FileRead(SrcFile,
                                     FileBuffer,
                                     SizeOf(FileBuffer));
            End_While;
            // close both files
            FileClose(SrcFile);
            FileClose(DstFile);
```

```
        Else
            // can't open destination file
            FileClose(SrcFile);
            Result := -2;
        End_If;
    Else
        // can't open source file
        Result := -1;
    End_If;
    If Result < 0 Then
        FileCopy := Result;
    Else
        FileCopy := FileSize;
    End_If;
End_Function;

//******************************************************
// 1. Enable Tracing in JetSym
// 2. Put source file name into 'SourceName'
// 3. Put destination file name into 'DestinationName'
// 4. Set flag 'UpdateIt'
//******************************************************
Task OSupdate Autorun
    Var
        ResCopy:    Int;
    End_Var;

    Loop
        UpdateIt := False;
        When UpdateIt Continue;
        ResCopy := FileCopy(SourceName,
                            DestinationName);
        Trace('Result : ' + IntToStr(ResCopy) + '$n');
    End_Loop;
End_Task;
```

12 Application program

Introduction

This chapter describes how to store the application program in JC-940MC. The user determines the program that is to be executed.

Required programmer's skills

This chapter requires knowledge on how to create application programs in JetSym and how to transmit them via the file system of the JC-940MC.

Contents

Application program - Default path

Introduction

When uploading the application program from JetSym to the controller, this program is stored as a file to the internal flash disk. The device enters the path and file name into the file **/app/start.ini**.

Path and file name

In the directory */app*, JetSym, by default, creates a subdirectory and assigns the project name to it. Then, JetSym stores the application program to this subdirectory assigning the extension ***.es9** to it. Path and file names are always converted into lower case letters.

The file */app/start.ini*

This file is a text file with one section holding two entries:

Element	Description
[Startup]	Section name
Project	Path to the application program file, relative to */app*
Program	Name of the application program file

Example:

```
[Startup]
Project = test_program
Program = test_program.es9
```

Result: The application program is loaded from the file **/app/test_program/test_program.es9**.

Related topics

- **Storing the application program to the USB flash drive** (see page 411)

The application program is stored to the USB flash drive

Introduction

When uploading the application program from JetSym to the controller, the default storage for application programs is used.

If you want the device to read the application program from the USB flash drive, you have to configure the file path.

If you want to store the application program to another directory of the internal flash disk, proceed the same way.

Prerequisites

Since the controller's file system is case sensitive, make sure that path and file names, as well as file entries are spelled correctly.

Storing the application program to the USB flash drive

If you want to store the application program to the USB flash drive, configure the device as follows:

Step	Action
1	Create an application program file by JetSym.
2	Create the desired directory on the USB flash drive.
3	Store the application program file to the desired directory.
4	Enter the path to the application program file and the program name into the file **/app/start.ini** on the internal flash disk of the device.

Result: On re-boot, the device uploads the application program from the USB flash drive.

The file */app/start.ini*

This file is a text file with one section holding two entries:

Element	Description
[Startup]	Section name
Project	Path leading to the application program file
Program	Name of the application program file

Example:

```
[Startup]
Project = /USB1/TestProgram
Program = Test1.es9
```

Result: The application program is loaded from the file **Test1.es9** located in the folder **TestProgram** on the USB flash drive (**/USB1/TestProgram/Test1.esx9**).

Related topics

- **Application program - default path** (see page 410)

Loading an application program

Introduction

At reboot of the application program via JetSym or booting the JC-940MC, the application program is loaded and executed via the file system. For this, mode selector S11 must be in *RUN* position.

The loading process

The application program is loaded by the controller's OS as follows:

Step	Description
1	The OS reads the file **/app/start.ini** from the internal flash disk.
2	The OS evaluates the **Project** entry. It contains the path leading to the application program file.
3	The OS evaluates the **Program** entry. It contains the program name.
4	The OS loads the application program from the file **<Project>/<Program>**.

13 Motion Control

Programming

Please refer to the JetSym help for a description on functions and programming of the Motion Control.

The Motion Control is programmed in the Motion API interface in the programming language STX.

14 Quick reference JC-9xx

Corresponding OS version

This quick reference summarizes in brief the registers and flags of the controller JC-940MC with OS version 1.05.0.13

General overview - Registers

100000 ... 100999	Electronic Data Sheet (EDS)
101000 … 101999	Configuration
102000 … 102999	Real-time clock
104000 ... 104999	Ethernet
108000 ... 108999	CPU
200000 ... 209999	General system registers
210000 ... 219999	Application program
230000 ... 239999	Networking via JetIP
240000 ... 249999	JetSync
250000 ... 259999	Ethernet System Bus
260000 ... 269999	RemoteScan
270000 ... 279999	Modbus/TCP
290000 ... 299999	E-mail
310000 ... 319999	File system/data files
320000 ... 359999	FTP client
350000 ... 359999	User-programmable IP interface
380000 ... 389999	Error history
390000 ... 399999	I/O networking
470000 ... 470999	NetConsistency: Basic drivers
471000 ... 471999	NetConsistency: First instance
1000000 ... 1119999	Application registers (remanent; Int/Float)
20SJ00000 ... 20SJ19999	JX2 modules; JetMove 2xx JX6 submodules
50000000 ... 59999999	Motion Control
	Networking via Jetter Ethernet system bus GNN: nnn = 000 ... 199
1nnn020000 ... 1nnn179999	JX3 module registers
1nnn202000 ... 1nnn227999	JX2 module registers
1nnn810000 ... 1nnn819999	JetMove registers
1nnn980000 ... 1nnn980199	Indirect access via local R 236xxx
1nnn990000 ... 1nnn999999	Indirect access with variable target window

I/Os - General overview

20001 ... 36000	Virtual I/Os for RemoteScan
20SJ0xx01 ... 20SJ0xx16	JX2 modules (xx: 02 ... 32); JX3 modules via JX3-BN-CAN (xx: 02 ... 17)
1nnn01xx01 ... 1nnn01xx16	JX3 modules via JX3-BN-ETH GNN: nnn = 000 ... 199 xx: 02 ... 17

Electronic Data Sheet (EDS)

[Identification]	
100600	Internal version number
100601	Module ID
100602 ... 100612	Module name (register string)
100613	PCB revision
100614	PCB options
[Production]	
100700	Internal version number
100701 ... 100707	Serial number (register string)
100708	Day
100709	Month
100710	Year
100711	TestNum
100712	TestRev
[Features]	**JC-9xx**
100800	Internal version number
100804	Switch
100805	STX
100806	Remanent registers
100810	Motion control
100812	HTTP/e-mail
100813	Modbus/TCP
100817	RTC

Configuration

	From file /system/config.ini
101100	IP address
101101	Subnet mask
101102	Default gateway
101103	DNS server
101132	HOSTNAME suffix type
101133 ... 101151	Host name (register string)
101164	JetIP port number
101165	Port number for STX debugger
	Used by the system
101200	ETH 1: IP address
101201	ETH 1: Subnet mask
101202	ETH 1: Default Gateway
101203	DNS server
101210	ETH 2: IP address
101211	ETH 2: Subnet mask
101212	ETH 2: Default Gateway
101213	ETH 3: IP address
101214	ETH 3: Subnet mask
101215	ETH 3: Default Gateway
101216	ETH 1: Static route of the IP address
101217	ETH 1: Static route of the subnet mask
101218	ETH 1: Static route of the gateway
101219	ETH 2: Static route of the IP address
101220	ETH 2: Static route of the subnet mask
101221	ETH 2: Static route of the gateway
101222	ETH 3: Static route of the IP address
101223	ETH 3: Static route of the subnet mask
101224	ETH 3: Static route of the gateway
101232	HOSTNAME suffix type
101233 … 101251	Host name (register string)
101264	JetIP port number
101265	Port number for STX debugger
101299	Write trigger (0x77566152)

Real-time clock

	Direct access
102910	Milliseconds
102911	Seconds
102912	Minutes
102913	Hours
102914	Weekday (0 = Sunday)
102915	Day
102916	Month
102917	Year

	Buffer access
102920	Milliseconds
102921	Seconds
102922	Minutes
102923	Hours
102924	Day of the week (0 = Sunday)
102925	Day
102926	Month
102927	Year
102928	Read/write trigger

Ethernet

	Link status	
104002	Link status ETH1	
	0	No link
	10	10 MBit/s half-duplex
	20	10 MBit/s full duplex
	100	100 MBit/s half-duplex
	200	100 MBit/s full duplex
	1000	1000 MBit/s half-duplex
	2000	1000 MBit/s full duplex
104004	Link status ETH2	
	0	No link
	10	10 MBit/s half-duplex
	20	10 MBit/s full duplex
	100	100 MBit/s half-duplex
	200	100 MBit/s full duplex
	1000	1000 MBit/s half-duplex
	2000	1000 MBit/s full duplex
104006	Link status ETH3	
	0	No link
	10	10 MBit/s half-duplex
	20	10 MBit/s full duplex
	100	100 MBit/s half-duplex
	200	100 MBit/s full duplex
	IP	
104531	ETH 1: Current IP address (rw)	
104532	ETH 1: Current subnet mask (rw)	
104533	ETH 1: Current Default Gateway (rw)	
104540	ETH 2: Current IP address (rw)	
104541	ETH 2: Current subnet mask (rw)	
104542	ETH 3: Current IP address (rw)	
104543	ETH 3: Current subnet mask (rw)	
104544	ETH 2: Current Default Gateway (rw)	
104545	ETH 3: Current Default Gateway (rw)	
	Routing table	
104550	Status	
	0	No error
	-1	Routing table is full
	-2	Entry not found
	-3	Port is not active
	-4	TCP/IP stack not initialized
104551	Command	
	1	Add route
	0	Delete route
104552	Port number	
	1	ETH 1 (X14)
	2	ETH 2 (X15)
	3	ETH 3 (X16)
104553	IP address	
104554	Subnet mask	
104555	Gateway	

CPU

108002	All LEDs on/off (bit-coded)
	Bit 0: LED **RUN**
	Bit 1: LED **ERR**
	Bit 2: LED **D1**
108003	LED **RUN**
	0 = OFF
	1 = Flashing slowly
	2 = Flashing fast
	3 = ON
108004	LED **ERR**
	0 = OFF
	3 = ON
108005	LED **D1**
	0 = OFF
	3 = ON
108015	Mode selector
	1 = LOAD
	2 = RUN
	3 = STOP

General system registers

200000	OS version (major * 100 + minor)	
200001	Application program is running (bit 0 = 1)	
200008	Error register (identical with 210004)	
	Bit 0:	Error on flash disk
	Bit 2:	Error on JX2 system bus
	Bit 3:	Error on Ethernet system bus
	Bit 4:	Error on application register
	Bit 7:	Error in expanded error register
	Bit 8:	Illegal jump
	Bit 9:	Illegal call
	Bit 10:	Illegal index
	Bit 11:	Illegal opcode
	Bit 12:	Division by 0
	Bit 13:	Stack overflow
	Bit 14:	Stack underflow
	Bit 15:	Illegal stack
	Bit 16:	Error when loading the application program
	Bit 17:	Memory protection violated
	Bit 24:	Timeout - cycle time
	Bit 25:	Timeout - task lock
	Bit 31:	Unknown error
200009	Enhanced error register (bit-coded)	
	Bit 3:	Error ModConfig.de
	Bit 10:	A bus node (publish/subscribe client) has reported an error to the controller

	Bit 12:	Error message by JetIPScan
	Bit 16:	Error message by NetConsistency.
200010	Enhanced error register (bit-coded)	
	Bit 1:	Error in the MC object
200051	Error number register of JetIPScan	
200061	Error number register of NetConsistency	
200169	OS version (IP format)	
200170	Controller type (940/970)	
201000	Runtime register in milliseconds (rw)	
201001	Runtime register in seconds (rw)	
201002	Runtime register in R 201003 Units (rw)	
201003	10 ms units for R 201002 (rw)	
201004	Runtime registers in milliseconds (ro)	
201005	Runtime registers in microseconds (ro)	
202930	Web status (bit-coded)	
	Bit 0 = 1:	FTP server available
	Bit 1 = 1:	HTTP-server available
	Bit 2 = 1:	E-mail available
	Bit 3 = 1:	Data file function available
	Bit 4 = 1:	Modbus/TCP exists
	Bit 5 = 1:	Modbus/TCP available
202936	Control register - File system	
	0xc4697a4b: Formatting the flash disk	
202960	Password for system command register (0x424f6f74)	
202961	System command register	
	102	Controller restart (reboot)
	103	Application register test
	104	Resetting the configuration register
	122	Waiting for communication - OFF
	123	Waiting for communication - ON
	160	Task switch on I/O access - OFF
	161	Task switch on I/O access - ON
	301	Saving the flash disk drive
	310	Loading the configuration data
	311	Load ModConfig.de
	312	Load configuration for Ethernet system bus
	313	Stop Ethernet system bus
	330	JetIPScan client - OFF
	331	JetIPScan client - ON
202962	System status register (bit-coded)	
	Bit 0 = 1:	Task switch on I/O access
	Bit 1 = 1:	Without waiting for communication
	Bit 2 = 1:	JetIPScan client is ON
202970	Password for start delay (0x424f6f74)	
202971	Delay time in 100 ms	
202980	Error history: Status	
202981	Error history: Command	
202984	Error history: Number of entries	
202985	Error history: Index	
202986	Error history: Entry	
202987	Error history: Stop error code	
202988	Error history: Stop error code - Threshold	
203000	Interface monitoring: JetIP	
203005	Interface monitoring: Debug server	
203100 ... 203107	32-bit overlaying - Flag 0 ... 255	
203108 ... 203123	16-bit overlaying - Flag 0 ... 255	
203124 ... 203131	32-bit overlaying - Flag 2048 ... 2303	
203132 ... 203147	16-bit overlaying - Flag 2048 ... 2303	
209700	System logger: Global enable	
209701 ... 209739	Enabling system components	

Application program

210000	Application program is running (bit 0 = 1)	
210001	JetVM version	
210004	Error register (bit-coded)	
	Bit 0:	Error on flash disk
	Bit 2:	Error on JX2 system bus
	Bit 3:	Error on JetEthernetBus
	Bit 4:	Error on application register
	Bit 7:	Error in expanded error register
	Bit 8:	Illegal jump
	Bit 9:	Illegal call
	Bit 10:	Illegal index
	Bit 11:	Illegal opcode
	Bit 12:	Division by 0
	Bit 13:	Stack overflow
	Bit 14:	Stack underflow
	Bit 15:	Illegal stack
	Bit 16:	Error when loading the application program
	Bit 17:	Memory protection violated
	Bit 24:	Timeout - cycle time
	Bit 25:	Timeout - task lock
	Bit 31:	Unknown error
210006	Highest task number	
210007	Minimum program cycle time	
210008	Maximum program cycle time	
210009	Current program cycle time	
210011	Current task number	
210050	Current program position within the presently ongoing runtime	
210051	ID of the presently ongoing runtime	
210056	Desired total cycle time in µs	
210057	Calculated total cycle time in µs	
210058	Maximum time slice per task in µs	
210060	Task ID (for R210061)	
210061	Priority for task [R210060]	
210063	Length of scheduler table	
210064	Index in scheduler table	
210065	Task ID in scheduler table	
210070	Task ID (for R210071)	
210071	Timer number (0 ... 31)	
210072	Manual triggering of a timer event (bit-coded)	
210073	End of cyclic task (task ID)	
210074	Command for cyclic tasks	
210075	Number of timers	
210076	Timer number (for R210077)	
210077	Timer value in milliseconds	

210100 ... 210199	Task state Please use the STX function `TaskGetInfo()` as described in the JetSym online help.
210400 ... 210499	Task - Program address
210600	Task ID of a cyclical task (for R210601)
210601	Processing time of a cyclical task in per mil figure
210609	Task lock timeout in ms -1: Monitoring disabled
210610	Timeout (bit-coded) Bit 0 -> Timer 0, etc.

TCP auto-close for the STX debug server

212000	Number of open connections
212001	Mode
212002	Time

Networking via JetIP

TCP auto-close for the JetIP/TCP server

230000	Number of open connections
230001	Mode
230002	Time

Other registers for networking via JetIP

232708	Timeout in milliseconds
232709	Response time in milliseconds
232710	Amount of network errors
232711	Error code of last access 0 = No error 1 = Timeout 3 = Error message from the remote station 5 = Invalid network address 6 = Invalid amount of registers 7 = Invalid interface number
232717	Max. number of retries
232718	Number of retries

Network registers

235000 ... 235399	IP addresses
235400 ... 235799	Port numbers
236000 ... 236199	Indirect register numbers
	GNN: nnn = 000 ... 199
1nnn020000 ... 1nnn179999	JX3 module registers
1nnn202000 ... 1nnn227999	JX2 module registers
1nnn810000 ... 1nnn819999	JetMove registers
1nnn980000 ... 1nnn980199	Indirect access via local register 236xxx
1nnn990000 ... 1nnn999999	Indirect access with variable target window

Ethernet system bus

Subscriber

250000	Status (bit-coded) Bit 0 = 1: No CRC Bit 1 = 1: Error in connection with a subscription Bit 7 = 1: Subscriber is running
250001	Command 102: Restart 105: STOPP 110: Acknowledge error
250002	Subscription ID of the last error
250003	Number of subscriptions
250004	CRC of configuration file
250010	Selection via command
250011	Selection via ID Subscription
250020	Status
250021	Mode
250022	Number of elements
250023	Multicast group
250024	Hash
250025	Current sequence number
250026	Size (bytes)
250027	Timeout
250028	Number of received publications
250029	Number of timeout errors
250030	Amount of sequence number errors
250100 ... 250999	9 more subscriber register blocks

Address of the bus node (or controller) exceeding the timeout time

254001	GNN
254002	IP address
254003	Port number

Publisher

255000	Status (bit-coded) Bit 0 = 1: No CRC Bit 1 = 1: Error in connection with a publication Bit 7 = 1: Subscriber is running
255001	Command 102: Restart 105: STOPP 110: Acknowledge error
255002	Publication ID of the last error
255003	Number of publications
255004	CRC of configuration file
255010	Selection via command
255011	Selection via ID Publication
255020	Status
255021	Mode
255022	Number of elements
255023	Multicast group
255024	Hash
255025	Current sequence number
255026	Size (bytes)
255027	Cycle time
255028	Number of publications sent
255029	Number of retries
255030	Number of transmit errors
255100 ... 255999	9 more publisher register blocks

RemoteScan

262965	Protocol type
262966	Amount of configuration blocks
262967	Status

Modbus/TCP

272702	Register offset
272704	Input offset
272705	Output offset
278000 ... 278999	16-bit I/O registers overlaid by virtual I/Os 20001 ... 36000

E-mail

292932	IP address of the SMTP server
292933	IP address of the POP3 server
292934	Port number of the SMTP server
292935	Port number of POP3 server
292937	Status of e-mail processing
292938	Task ID - E-mail

File system/data file function

312977	Status of file operation
312978	Task ID

FTP client

320000	Number of open connections
320001	Command
320002	Timeout
320003	Server port
320004	Selection via number
320005	Selection via handle
320006	Server socket: IP address
320007	Server socket: Port
320008	Client socket: IP address
320009	Client socket: Port
320100	Access status
320101	Task ID

User-programmable IP interface

Reading out the connection list

350000	Last result (-1 = no connection selected)
350001	1 = Client; 2 = Server
350002	1 = UDP; 2 = TCP
350003	IP address
350004	Port number
350005	Connection state
350006	Number of sent bytes
350007	Number of received bytes

Error history

380000 Status

- Bit 0 = 1: Recording
- Bit 1 = 1: Stop if buffer is full
- Bit 2 = 1: Stop on error code
- Bit 3 = 1: Remanent memory

380001 Command

- 1: Clear error log
- 2: Start error log
- 3: Stop error log
- 4: Stop if error buffer is full
- 5: Circular buffer
- 6: Stop on error code ON
- 7: Stop on error code OFF
- 10: Remanent memory
- 11: Dynamic memory

380002	Buffer length
380003	Maximum buffer length
380004	Number of error entries
380005	Index to error list
380006	Error entry
380007	Error stop code
380008	Number of codes until stop

I/O networking

Status registers

390000 + node * 10	Error register
390001 + node * 10	Enhanced error register 1
390002 + node * 10	Enhanced error register 2
390003 + node * 10	JetSync status
390004 + node * 10	Subscriber status
390005 + node * 10	Subscription ID of the last error

Address of a bus node (not of a controller) having reported an error

394001	GNN
394002	IP address
394003	Port number

Control register

395000 + node * 10	Command

NetConsistency function

Basic drivers

470000 ... 470008	Cookie
470009	Version
470010	Status

- Bit 0 = 1: Error
- Bit 1 = 1: Alarms
- Bit 2 = 1: Basic driver initialized

470011 Command

0: There are no commands

470020	Maximum possible number of instances
470021	Number of instances ready for operation
470030	Max. number of error messages for the logger
470031	Number of error messages transmitted to the logger
470032	Max. number of warnings for the logger
470033	Number of warnings forwarded to the logger
470034	Max. possible number of error history entries
470035	Number of entries in the error history
470040	Error numbers
470041	Time of the error in ms
470042	Instance, at which the error occurred
470043	Number of error parameters
470044 ... 470048	Error parameters 1 through 5

470049	Number of characters of the error message
470050 ... 470157	Text of the error message
	First instance
471010	Status Bit 0 = 1: Error Bit 1 = 1: Alarms Bit 2 = 1: An instance has been initialized Bit 3 = 1: Execution in process
471011	Command 0: There are no commands

Application registers

1000000 ... 1119999	32-bit integer or floating point number (remanent)

Basic register for initializing the PCI bus

20SJ00000	Global control register Bit 30: Initialization completed Bit 31: Error when initializing
20SJ00001	Command register 1: Initializing the bus
20SJ00002	Number of detected module boards
20SJ00005	Bus initialization status 0: Initialization is running 1: Initialization OK -1: Error when initializing
20SJ00006	Number of JX2 modules
20SJ00007	Types of the JX6-I/O submodules 73: JX6-SB and JX6-SB-I 5: JX6-SV1 16: JX6-IO16CB
20SJ00010	Timeout while waiting for resetting the command
20SJ00011	Timeout while waiting for semaphore
20SJ00012	Timeout in the interpreter for total access

Access to controller modules (JX6-SB(I))

Flag 2105 = 1 reports an error

20SJ00050	Access error controller module Slot number - 1
20SJ00051	Access error controller module Axis number - 1
20SJ00052	Access error - controller module Register number

Digital I/O module JX6-IO16CB

20SJ00100	State of the digital inputs
20SJ00101	State of digital outputs
20SJ00102	Error state of the digital outputs
20SJ00103	Control register
20SJ00104	Filter time of the inputs 1 through 4
20SJ00105	Filter time of the inputs 5 through 8

Combi module JX6-SV1

20SJ00100	Module ID
20SJ00101	Hardware configuration
20SJ00102	Analog output
20SJ00103	Strobe value/encoder value received
20SJ00104	Preset/pulse generator
20SJ00105	Count value
20SJ00106	Reference/offset
20SJ00107	Control register
20SJ00108	Status register
20SJ00110	Sampling timer

Submodule JX6-SB(-I)

	Operating mode "Master-Slave"
20SJ00100	Status register
20SJ00101	Command register
20SJ00109	Firmware version number
20SJ00201	Register index of the JX-SIO
20SJ00202	Register date of the JX-SIO

JX2 system bus registers

20SJ02000	Version of JX2 system bus driver (IP)
20SJ02008	Error (bit-coded) Bit 3: I/O or CANopen® module timeout Bit 4: JX2 slave module timeout Bit 9: Error of I/O module periphery Bit 13: Error during JX2 system bus initialization Bit 14: Timeout of system registers
20SJ02011	I/O module number at timeout
20SJ02012	JX2 slave module number at timeout
20SJ02013	Amount of connected I/O modules
20SJ02014	Amount of connected JX2 slave modules
20SJ02015	Index to module array
20SJ02016	Module array
20SJ02023	Dummy I/O module
20SJ02024	JX2 slave dummy modules
20SJ02028	Monitoring interval for I/O modules [10 ms]
20SJ02029	Baud rate of JX2 system bus
20SJ02032	ON delay
20SJ02039	I/O module where a peripheral fault has occurred (bit-coded)
20SJ02070	Number of CANopen® modules
20SJ02071	Actual I/O sum of modules on the JX2 system bus
20SJ02072	Version of JX2 system bus driver (IP)
20SJ02073	Timeout for register access to CANopen® modules
20SJ02074	CANopen® sync intervall [ms]
20SJ02077	Enabling JX2 system bus special functions Bit 2: CAN-PRIM Bit 3: CAN-PRIM only
20SJ02080	CANopen® module index for JX2 system bus application registers
20SJ02085	SysBus application regs: Register number (65-89)
20SJ02086	SysBus application reg: Object ID
20SJ02087	SysBus application reg: Subindex
20SJ02088	SysBus application reg: Length
20SJ02638	Special flag, overlaid Bit 0: Flag 2048 Timeout during an access to a JX2-I/O module Bit 1: Flag 2049 Timeout during an access to a JX2 slave module Bit 2: Flag 2050 Timeout during register access to a JX2-I/O module
20SJ02639	Special flag, overlaid Bit 1: Flag 2065 Activated error signalization for output driver errors Bit 3: Fatal system bus error
20SJ02651	Special flag, overlaid

	Bit 14:	Flag 2270 Timeout during an access to an inactive JX-SIO module
20SJ02652	Special flag, overlaid	
	Bit 0:	Flag 2272 Access to an unknown JX-SIO-register
	Bit 1:	Flag 2273 Access to a JX-SIO-register that is not supported
	Bit 2:	Flag 2274 Timeout when monitoring a JX-SIO module
	Bit 3:	Flag 2275 JX-SIO module is not ready for operation
	Bit 4:	Flag 2276 Overflow during a read access to a 32-bit register
	Bit 5:	Flag 2277 Selecting the upper 8 bits for 32-bit write access to JX-SIO-registers

20SJ02760	Max. number of I/O update retries
20SJ02761	Index of array of I/O retry counters
20SJ02762	Array of I/O retry counters
20SJ02763	Timeout for I/O update of I/O modules [ms]
20SJ02764	Timeout for register access to I/O modules [ms]
20SJ02765	Timeout for register access to JX2 slave modules [ms]
20SJ02995	Bootloader version of JX2 system bus interface
20SJ03xx0 ... 20SJ03xx9	Registers on I/O modules xx: I/O module number - 2 (00...22)
20SJ04000 ... 20SJ04367	Inputs/outputs mapped to registers (see below)
20SJ05x00 ... 20SJ06x99	I/O registers: CANopen®/JX-SIO x: I/O module number - 70 (0...9)
20SJ07x00 ... 20SJ07x99	Configuration registers: CANopen®/JX-SIO x: I/O module number - 70 (0...9)
20SJxx100 ... 20SJxx999	JX2 slave registers xx: JX2 slave number + 10

CAN-PRIM registers

20SJ10500	Status register Bit 1 = 1: CAN message received Bit 2 = 0: 11-bit CAN ID Bit 2 = 1: 29-bit CAN ID
20SJ10501	Command register 7 = clear FIFO 8 = Set CAN ID to 11 bits 9 = Set CAN ID to 29 bits 10 = Check boxes for received messages
20SJ10503	FIFO buffer occupancy
20SJ10504	FIFO data
20SJ10506	Global receiving mask
20SJ10507	Global receive ID
20SJ10530 + Box * 20	Status register of the box
20SJ10531 + Box * 20	Configuration register of the box
20SJ10532 + Box * 20	CAN ID
20SJ10533 + Box * 20	Number of data bytes
20SJ10534 ... 20SJ10541 + Box * 20	Data bytes
20SJ10542 + Box * 20	CAN ID mask
20SJ10543 + Box * 20	Command register of the box
20SJ10544 + Box * 20	Received CAN ID

Inputs/outputs

20001 ... 36000	Virtual I/Os for RemoteScan
20SJ00101 ... 20SJ00108	JX6-IO16CB
20SJ0xx01 ... 20SJ0xx16 1nnn01xx01 ... 1nnn01xx16	JX2 modules (xx: 02 ... 32); JX3 modules via JX3-BN-CAN (xx: 02 ... 17) JX3 modules via JX3-BN-ETH GNN: nnn = 000 ... 199 xx: 02 ... 17

32 combined inputs

JX2 system bus: + 20SJ00000
Network: + 1nnn910000

4000	101..108	109..116	201..208	209..216
4001	109..116	201..208	209..216	301..308
4002	201..208	209..216	301..308	309..316
4003	209..216	301..308	309..316	401..408
4004	301..308	309..316	401..408	409..416
4005	309..316	401..408	409..416	501..508
4006	401..408	409..416	501..508	509..516
4007	409..416	501..508	509..516	601..608
4008	501..508	509..516	601..608	609..616
4009	509..516	601..608	609..616	701..708
4010	601..608	609..616	701..708	709..716
4011	609..616	701..708	709..716	801..808
4012	701..708	709..716	801..808	809..816
4013	709..716	801..808	809..816	901..908
4014	801..808	809..816	901..908	909..916
4015	809..816	901..908	909..916	1001..1008
4016	901..908	909..916	1001..1008	1009..1016
4017	909..916	1001..1008	1009..1016	1101..1108
4018	1001..1008	1009..1016	1101..1108	1109..1116
4019	1009..1016	1101..1108	1109..1116	1201..1208
4020	1101..1108	1109..1116	1201..1208	1209..1216
4021	1109..1116	1201..1208	1209..1216	1301..1308
4022	1201..1208	1209..1216	1301..1308	1309..1316
4023	1209..1216	1301..1308	1309..1316	1401..1408
4024	1301..1308	1309..1316	1401..1408	1409..1416
4025	1309..1316	1401..1408	1409..1416	1501..1508
4026	1401..1408	1409..1416	1501..1508	1509..1516
4027	1409..1416	1501..1508	1509..1516	1601..1608
4028	1501..1508	1509..1516	1601..1608	1609..1616
4029	1509..1516	1601..1608	1609..1616	1701..1708
4030	1601..1608	1609..1616	1701..1708	1709..1716
4031	1609..1616	1701..1708	1709..1716	1801..1808
4032	1701..1708	1709..1716	1801..1808	1809..1816
4033	1709..1716	1801..1808	1809..1816	1901..1908
4034	1801..1808	1809..1816	1901..1908	1909..1916
4035	1809..1816	1901..1908	1909..1916	2001..2008
4036	1901..1908	1909..1916	2001..2008	2009..2016
4037	1909..1916	2001..2008	2009..2016	2101..2108
4038	2001..2008	2009..2016	2101..2108	2109..2116
4039	2009..2016	2101..2108	2109..2116	2201..2208
4040	2101..2108	2109..2116	2201..2208	2209..2216
4041	2109..2116	2201..2208	2209..2216	2301..2308
4042	2201..2208	2209..2216	2301..2308	2309..2316
4043	2209..2216	2301..2308	2309..2316	2401..2408
4044	2301..2308	2309..2316	2401..2408	2409..2416

16 combined inputs

JX2 system bus: + 20SJ00000
Network: + 1nnn910000

4060	101..108	109..116
4061	109..116	201..208
4062	201..208	209..216
4063	209..216	301..308
4064	301..308	309..316
4065	309..316	401..408

4066	401..408	409..416
4067	409..416	501..508
4068	501..508	509..516
4069	509..516	601..608
4070	601..608	609..616
4071	609..616	701..708
4072	701..708	709..716
4073	709..716	801..808
4074	801..808	809..816
4075	809..816	901..908
4076	901..908	909..916
4077	909..916	1001..1008
4078	1001..1008	1009..1016
4079	1009..1016	1101..1108
4080	1101..1108	1109..1116
4081	1109..1116	1201..1208
4082	1201..1208	1209..1216
4083	1209..1216	1301..1308
4084	1301..1308	1309..1316
4085	1309..1316	1401..1408
4086	1401..1408	1409..1416
4087	1409..1416	1501..1508
4088	1501..1508	1509..1516
4089	1509..1516	1601..1608
4090	1601..1608	1609..1616
4091	1609..1616	1701..1708
4092	1701..1708	1709..1716
4093	1709..1716	1801..1808
4094	1801..1808	1809..1816
4095	1809..1816	1901..1908
4096	1901..1908	1909..1916
4097	1909..1916	2001..2008
4098	2001..2008	2009..2016
4099	2009..2016	2101..2108
4100	2101..2108	2109..2116
4101	2109..2116	2201..2208
4102	2201..2208	2209..2216
4103	2209..2216	2301..2308
4104	2301..2308	2309..2316
4105	2309..2316	2401..2408
4106	2401..2408	2409..2416

8 combined inputs

JX2 system bus: + 20SJ00000
Network: + 1nnn910000

4120	101..108
4121	109..116
4122	201..208
4123	209..216
4124	301..308
4125	309..316
4126	401..408
4127	409..416
4128	501..508
4129	509..516
4130	601..608
4131	609..616
4132	701..708
4133	709..716
4134	801..808
4135	809..816
4136	901..908
4137	909..916
4138	1001..1008
4139	1009..1016
4140	1101..1108
4141	1109..1116
4142	1201..1208
4143	1209..1216
4144	1301..1308
4145	1309..1316
4146	1401..1408
4147	1409..1416
4148	1501..1508
4149	1509..1516
4150	1601..1608
4151	1609..1616
4152	1701..1708
4153	1709..1716
4154	1801..1808
4155	1809..1816
4156	1901..1908
4157	1909..1916
4158	2001..2008
4159	2009..2016
4160	2101..2108
4161	2109..2116
4162	2201..2208
4163	2209..2216
4164	2301..2308
4165	2309..2316
4166	2401..2408
4167	2409..2416

32 combined outputs

JX2 system bus + 20SJ00000
Network: + 1nnn910000

4200	101..108	109..116	201..208	209..216
4201	109..116	201..208	209..216	301..308
4202	201..208	209..216	301..308	309..316
4203	209..216	301..308	309..316	401..408
4204	301..308	309..316	401..408	409..416
4205	309..316	401..408	409..416	501..508
4206	401..408	409..416	501..508	509..516
4207	409..416	501..508	509..516	601..608
4208	501..508	509..516	601..608	609..616
4209	509..516	601..608	609..616	701..708
4210	601..608	609..616	701..708	709..716
4211	609..616	701..708	709..716	801..808
4212	701..708	709..716	801..808	809..816
4213	709..716	801..808	809..816	901..908
4214	801..808	809..816	901..908	909..916
4215	809..816	901..908	909..916	1001..1008
4216	901..908	909..916	1001..1008	1009..1016
4217	909..916	1001..1008	1009..1016	1101..1108
4218	1001..1008	1009..1016	1101..1108	1109..1116
4219	1009..1016	1101..1108	1109..1116	1201..1208
4220	1101..1108	1109..1116	1201..1208	1209..1216
4221	1109..1116	1201..1208	1209..1216	1301..1308
4222	1201..1208	1209..1216	1301..1308	1309..1316
4223	1209..1216	1301..1308	1309..1316	1401..1408
4224	1301..1308	1309..1316	1401..1408	1409..1416
4225	1309..1316	1401..1408	1409..1416	1501..1508
4226	1401..1408	1409..1416	1501..1508	1509..1516
4227	1409..1416	1501..1508	1509..1516	1601..1608
4228	1501..1508	1509..1516	1601..1608	1609..1616
4229	1509..1516	1601..1608	1609..1616	1701..1708
4230	1601..1608	1609..1616	1701..1708	1709..1716
4231	1609..1616	1701..1708	1709..1716	1801..1808
4232	1701..1708	1709..1716	1801..1808	1809..1816
4233	1709..1716	1801..1808	1809..1816	1901..1908
4234	1801..1808	1809..1816	1901..1908	1909..1916
4235	1809..1816	1901..1908	1909..1916	2001..2008
4236	1901..1908	1909..1916	2001..2008	2009..2016
4237	1909..1916	2001..2008	2009..2016	2101..2108
4238	2001..2008	2009..2016	2101..2108	2109..2116
4239	2009..2016	2101..2108	2109..2116	2201..2208
4240	2101..2108	2109..2116	2201..2208	2209..2216
4241	2109..2116	2201..2208	2209..2216	2301..2308
4242	2201..2208	2209..2216	2301..2308	2309..2316
4243	2209..2216	2301..2308	2309..2316	2401..2408
4244	2301..2308	2309..2316	2401..2408	2409..2416

16 combined outputs

JX2 system bus + 20SJ00000
Network: + 1nnn910000

4260	101..108	109..116
4261	109..116	201..208
4262	201..208	209..216
4263	209..216	301..308
4264	301..308	309..316
4265	309..316	401..408
4266	401..408	409..416
4267	409..416	501..508
4268	501..508	509..516
4269	509..516	601..608

4270	601..608	609..616
4271	609..616	701..708
4272	701..708	709..716
4273	709..716	801..808
4274	801..808	809..816
4275	809..816	901..908
4276	901..908	909..916
4277	909..916	1001..1008
4278	1001..1008	1009..1016
4279	1009..1016	1101..1108
4280	1101..1108	1109..1116
4281	1109..1116	1201..1208
4282	1201..1208	1209..1216
4283	1209..1216	1301..1308
4284	1301..1308	1309..1316
4285	1309..1316	1401..1408
4286	1401..1408	1409..1416
4287	1409..1416	1501..1508
4288	1501..1508	1509..1516
4289	1509..1516	1601..1608
4290	1601..1608	1609..1616
4291	1609..1616	1701..1708
4292	1701..1708	1709..1716
4293	1709..1716	1801..1808
4294	1801..1808	1809..1816
4295	1809..1816	1901..1908
4296	1901..1908	1909..1916
4297	1909..1916	2001..2008
4298	2001..2008	2009..2016
4299	2009..2016	2101..2108
4300	2101..2108	2109..2116
4301	2109..2116	2201..2208
4302	2201..2208	2209..2216
4303	2209..2216	2301..2308
4304	2301..2308	2309..2316
4305	2309..2316	2401..2408
4306	2401..2408	2409..2416

8 combined outputs

JX2 system bus + 20SJ00000
Network: + 1nnn910000

4320	101..108
4321	109..116
4322	201..208
4323	209..216
4324	301..308
4325	309..316
4326	401..408
4327	409..416
4328	501..508
4329	509..516
4330	601..608
4331	609..616
4332	701..708
4333	709..716
4334	801..808
4335	809..816
4336	901..908
4337	909..916
4338	1001..1008
4339	1009..1016
4340	1101..1108
4341	1109..1116
4342	1201..1208
4343	1209..1216
4344	1301..1308
4345	1309..1316
4346	1401..1408
4347	1409..1416
4348	1501..1508
4349	1509..1516
4350	1601..1608
4351	1609..1616
4352	1701..1708
4353	1709..1716
4354	1801..1808
4355	1809..1816
4356	1901..1908
4357	1909..1916
4358	2001..2008
4359	2009..2016
4360	2101..2108
4361	2109..2116
4362	2201..2208
4363	2209..2216
4364	2301..2308
4365	2309..2316
4366	2401..2408
4367	2409..2416

Special flags for networks

2075	Error in networking via JetIP

Special flags - Publish/subscribe

2080	Enable for publishing an error
2081	Error collection of the subscriber

Special flags - Interface monitoring

2088	OS flag - JetIP
2089	User flag - JetIP
2098	OS flag - Debug server
2099	User flag - Debug server

32 combined flags

203100	0 ... 31
203101	32 ... 63
203102	64 ... 95
203103	96 ... 127
203104	128 ... 159
203105	160 ... 191
203106	192 ... 223
203107	224 ... 255

16 combined flags

203108	0 ... 15
203109	16 ... 31
203110	32 ... 47
203111	48 ... 63
203112	64 ... 79
203113	80 ... 95
203114	96 ... 111
203115	112 ... 127
203116	128 ... 143
203117	144 ... 159
203118	160 ... 175
203119	176 ... 191
203120	192 ... 207
203121	208 ... 223
203122	224 ... 239
203123	240 ... 255

32 combined special flags

203124	2048 ... 2079
203125	2080 ... 2111
203126	2112 ... 2143
203127	2144 ... 2175
203128	2176 ... 2207
203129	2208 ... 2239
203130	2240 ... 2271
203131	2272 ... 2303

16 combined special flags

203132	2048 ... 2063
203133	2064 ... 2079
203134	2080 ... 2095
203135	2096 ... 2111

203136	2112 ... 2127
203137	2128 ... 2143
203138	2144 ... 2159
203139	2160 ... 2175
203140	2176 ... 2191
203141	2192 ... 2207
203142	2208 ... 2223
203143	2224 ... 2239
203144	2240 ... 2255
203145	2256 ... 2271
203146	2272 ... 2287
203147	2288 ... 2303

Overlaid application registers/flags

1000000	256 ... 287
1000001	288 ... 319
1000002	320 ... 351
1000003	352 ... 383
1000004	384 ... 415
1000005	416 ... 447
1000006	448 ... 479
1000007	480 ... 511
1000008	512 ... 543
1000009	544 ... 575
1000010	576 ... 607
1000011	608 ... 639
1000012	640 ... 671
1000013	672 ... 703
1000014	704 ... 735
1000015	736 ... 767
1000016	768 ... 799
1000017	800 ... 831
1000018	832 ... 863
1000019	864 ... 895
1000020	896 ... 927
1000021	928 ... 959
1000022	960 ... 991
1000023	992 ... 1023
1000024	1024 ... 1055
1000025	1056 ... 1087
1000026	1088 ... 1119
1000027	1120 ... 1151
1000028	1152 ... 1183
1000029	1184 ... 1215
1000030	1216 ... 1247
1000031	1248 ... 1279
1000032	1280 ... 1311
1000033	1312 ... 1343
1000034	1344 ... 1375
1000035	1376 ... 1407
1000036	1408 ... 1439
1000037	1440 ... 1471
1000038	1472 ... 1503
1000039	1504 ... 1535
1000040	1536 ... 1567
1000041	1568 ... 1599
1000042	1600 ... 1631
1000043	1632 ... 1663
1000044	1664 ... 1695
1000045	1696 ... 1727
1000046	1728 ... 1759
1000047	1760 ... 1791
1000048	1792 ... 1823
1000049	1824 ... 1855
1000050	1856 ... 1887
1000051	1888 ... 1919
1000052	1920 ... 1951
1000053	1952 ... 1983
1000054	1984 ... 2015
1000055	2016 ... 2047

System function

For reasons of compatibility, the system functions are listed below. In JetSym STX, use the corresponding JetSym STX functions instead of system functions.

4	Conversion from BCD to HEX
5	Conversion from HEX to BCD
20	Square root
21	Sine
22	Cosine
23	Tangent
24	Arc sine
25	Arc cosine
26	Arc tangent
27	Exponential function
28	Natural logarithm
29	Absolute value
30	Separation of digits before and after the decimal point
50	Sorting register values
60	CRC generation for Modbus RTU
61	CRC check for Modbus RTU
65/67	Reading register block via Modbus/TCP
66/68	Writing register block via Modbus/TCP
80/85	Initializing RemoteScan
81	Starting RemoteScan
82	Stopping RemoteScan
90	Writing a data file
91	Appending a data file
92	Reading a data file
96	Deleting a data file
110	Sending an e-mail
150	Configuring NetCopyList
151	Deleting NetCopyList
152	Sending NetCopyList

JetSym STX functions

System function	Corresponding JetSym STX function
4	Function Bcd2Hex(Bcd: Int): Int;
5	Function Hex2Bcd(Hex: Int): Int;
50	Function QSort(DataPtr: Int, ElementCnt: Int, ElementSize: Int, SortOffset: Int, SortType: STXBASETYPE, SortMode: QSORTMODE): Int;
60	Function ModbusCRCgen(FramePtr: Int, Length: Int): Int;
61	Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int;
65/67	Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int;
66/68	Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int;
80/85	Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int;
81	Function RemoteScanStart(Protocol: Int): Int;
82	Function RemoteScanStop(Protocol: Int): Int;
90/91	Function FileDAWrite(Const Ref FileName: String, Const Ref Mode: String, VarType: DAWRITE_TYPE, First: Int, Last: Int): Int;
92	Function FileDARead(Const Ref FileName: String): Int;
110	Function EmailSend(Const Ref FileName: String): Int;
150	Function NetCopyListConfig(IPAddr: Int, IPPort: Int, Const Ref List: TNetCopyLinstL): Int;
151	Function NetCopyListSend(Handle: Int): Int;
152	Function NetCopyListDelete(Handle: Int): Int;

Appendix

Introduction

This appendix contains electrical and mechanical data, as well as operating data.

Contents

A: Technical specifications

Introduction

This chapter contains information on electrical and mechanical data, as well as on operating data of the JC-940MC.

Contents

Technical data

Electrical data: Power supply

Parameter	Description
Rated voltage	DC 24 V
Permissible voltage range	-15 % ... +20 %
Input current	3.125 A max.
Power consumption	75 W max.

Memory configurations

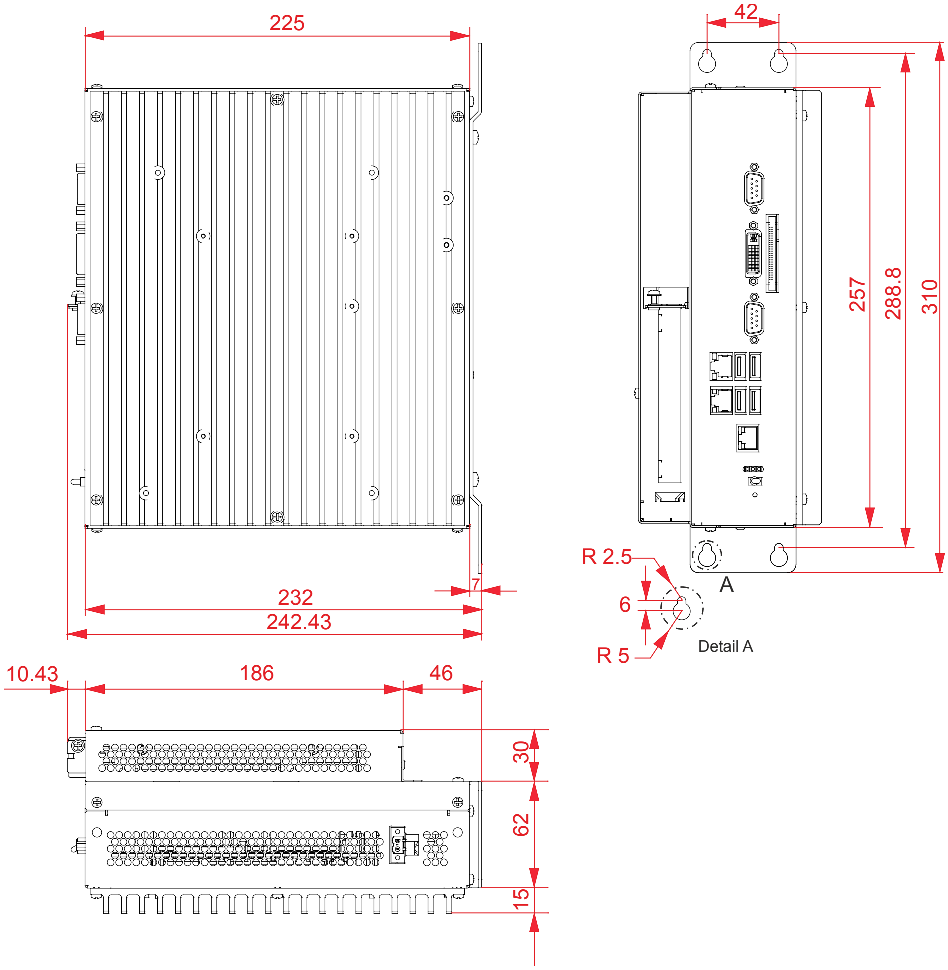
Parameter	Description
Remanent memory for variables	120,000 registers
Flash disk	8 MBytes

Technical data - Real-time clock

Parameter	Description
Lifetime	Minimum: 10 years
Deviation	1 minute per month max.

Physical dimensions

Physical dimensions



Minimum clearances	At mounting the controller JC-940MC, a minimum clearance above, below and on the right-hand side (side where the heat sink is positioned) must be maintained. ▪ Minimum clearance, above: 100 mm ▪ Minimum clearance, below: 100 mm ▪ Minimum clearance, right-hand side: 50 mm
Module width	The width of the controller JC-940MC is 77 mm. The corresponding module boards (PCI slots) have a width of 30 mm each. In case of option E, they widen the dimensions of the controller. The width of the controller JC-940MC-E01 is 107 mm, for example.
Mounting orientation	The orientation of the controller JC-940MC is vertical.

Operating parameters - Environment and mechanics

Environment

Parameter	Value	Standard
Operating temperature range	5 ... +55 °C	
Storage temperature range	-40 ... +70 °C	DIN EN 61131-2 DIN EN 60068-2-1 DIN EN 60068-2-2
Air humidity	10 ... 95 %, Non-condensing	DIN EN 61131-2
Pollution degree	2	DIN EN 61131-2
Corrosion/ chemical resistance	No special protection against corrosion. Ambient air must be free from higher concentrations of acids, alkaline solutions, corrosive agents, salts, metal vapors, or other corrosive or electroconductive contaminants.	
Maximum operating altitude	3,000 m above sea level	DIN EN 61131-2

Mechanical parameters

Parameter	Value	Standard
Free falls withstanding test	For weight < 10 kg: Height of fall (units within packing): 1 m Product packaging: 0.3 m	DIN EN 61131-2 DIN EN 60068-2-31
Vibration resistance	5 ... 9 Hz: Amplitude: 3.5 mm 9 ... 150 Hz: 1 g acceleration: 1 octave/minute, 10 frequency sweeps (sinusoidal), all three spatial axes	DIN EN 61131-2 DIN EN 60068-2-6
Shock resistance	15 g occasionally, 11 ms, sinusoidal half-wave, 3 shocks in the directions of all three spatial axes	DIN EN 61131-2 DIN EN 60068-2-27
Degree of protection	IP20	DIN EN 60529
Mounting orientation	Vertical	

Operating parameters: Enclosure

Electrical safety

Parameter	Value	Standard
Protection class	III	DIN EN 61131-2
Dielectric test voltage	Functional ground is connected to chassis ground internally.	DIN EN 61131-2
Protective connection	0	DIN EN 61131-2
Overvoltage category	II	DIN EN 61131-2

EMC - Emitted interference

Parameter	Value	Standard
Enclosure	Frequency band 30 ... 230 MHz, limit 30 dB (µV/m) in 10 m Frequency band 230 ... 1,000 MHz, limit 37 dB (µV/m) at 10 m distance (Class B)	DIN EN 61000-6-3 DIN EN 61131-2 DIN EN 55011

EMC - Interference immunity

Parameter	Value	Standard
Magnetic field with mains frequency	50 Hz 30 A/m	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-8
RF field, amplitude-modulated	Frequency band 80 MHz ... 1 GHz Test field strength: 10 V/m AM 80 % mit 1 kHz Criterion A	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-3
ESD	Discharge through air: Test peak voltage 8 kV Contact discharge: Test peak voltage 4 kV Criterion A	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-2

DC power supply inputs and outputs

EMC - Immunity to interference

Parameter	Value	Standard
RF, asymmetric	Frequency band 0.15 ... 80 MHz Test voltage 10 V AM 80 % with 1 kHz Source impedance 150 Ω Criterion A	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-6
Bursts	Test voltage 2 kV tr/tn 5/50 ns Repetition rate 5 kHz Criterion A	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-4
Surge voltages asymmetric (line to earth), symmetrical (line to earth)	tr/th 1.2/50 µs Common-mode interference voltage 1 kV Series-mode interference voltage 0.5 kV	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-5

Shielded data and I/O lines

EMC - Immunity to interference

Parameter	Value	Standard
Asymmetric RF, amplitude-modulated	Frequency band 0.15 ... 80 MHz Test voltage 10 V AM 80 % with 1 kHz Source impedance 150 Ω Criterion A	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-6
Bursts	Test voltage 1 kV tr/tn 5/50 ns Repetition rate 5 kHz Criterion A	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-4
Voltage surges, asymmetric (line to earth)	tr/th 1.2/50 µs Common-mode interference voltage 1 kV	DIN EN 61131-2 DIN EN 61000-6-2 DIN EN 61000-4-5

B: Index

P

R

S

T

U

V

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.