# JVM-407

## HMI


User Manual

Jetter

Variant: Jetter

Item # 60877112

Revision 1.17.4

November 2011 / Printed in Germany

**Address**

How To Contact us:

Jetter AG
Graeterstrasse 2
D-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 JVM-407:

Type: ______________________

Serial #: ______________________

Year of construction: ______________________

Order #: ______________________

To be entered by the customer:

Inventory #: ______________________

Place of operation: ______________________

**Significance**

Significance of this user manual

The user manual is an integral part of JVM-407:

- It must be kept in a way that it is always at hand, until the JVM-407 will be disposed of.
- If the JVM-407 is sold or loaned/leased out, the user manual has to be passed on.

In any case you encounter difficulties to clearly understand this user manual, please contact the manufacturer.

We would appreciate any suggestions and contributions on your part and would ask you to contact us by our e-mail address info@jetter.de. This will help us to produce manuals that are more user-friendly and to address your wishes and requirements.

This user manual contains important information on how to transport, erect, install, operate, maintain and repair the JVM-407.

Therefore, the persons carrying out these jobs must carefully read, understand and observe this user manual, and especially the safety instructions.

Missing or inadequate knowledge of the user manual results in the loss of any claim of liability on part of Jetter AG. Therefore, the operating company is recommended to have the instruction of the persons concerned confirmed in writing.

# Table of Contents

# 1 Safety Instructions

**Introduction**

This chapter informs the user of general safety instructions and warns of residual dangers, if applicable. Furthermore, it contains information on EMC.

**Contents**

## Basic Safety Instructions

**Introduction**

This device complies with the valid safety regulations and standards. Special emphasis was given 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 device has been designed for use in commercial vehicles and mobile machines. The device JVM-407 is an HMI with integrated controller for exchange of data with peripheral devices.

The HMI JVM-407 meets the requirement of the European Automotive EMC Directive for electric/electronic subassemblies.

The HMI JVM-407 must be operated within the limits and conditions established in the technical specifications. The operating voltage of the HMI JVM-407 is classified as SELV (Safety Extra Low Voltage). Therefore, the HMI JVM-407 is not subject to the EU Low Voltage Directive.

**Usage Other Than Intended**

This device must not be used in technical systems which to a high degree have to be fail-safe, e.g. ropeways and aeroplanes.

The JVM-407 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 the device is to be run under ambient conditions which differ from the allowed operating conditions, Jetter AG is to be contacted 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/automotive engineering, such as automotive mechatronics fitters. |
| **Commissioning / Programming:** | Trained and instructed experts with profound knowledge of, and experience with, automotive / automation technology, such as automotive engineers for mobile machinery. |
| **Operation:** | Trained, instructed and assigned personnel with knowledge in operating electronic devices for mobile machinery. |
| **Decommissioning:** | Specialized personnel with training in electrical/automotive engineering, such as automotive mechatronics fitters. |

**Modifications and Alterations to the Device**

**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 are not tested on our part, 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 JVM-407 contains electrostatic sensitive components which can be damaged if not handled properly.

To exclude damages to the JVM-407 during transport it should only be shipped in its original packaging or in packaging protecting against electrostatic discharge. This is particularly true for transport via mail.

- Use an appropriate outer packaging to protect the JVM-407 against impact or shock.
- In case of damaged packaging inspect the device for any visible damage. Inform your freight forwarder and the manufacturer, if applicable.

**Storing**

When storing the JVM-407 observe the environmental conditions given in the technical specification.

**Repair and Maintenance**

This device must not be repaired by the operators themselves. The device does not contain any parts that could be repaired by the operator.

The device must be sent to Jetter AG for repair.

**Disposal**

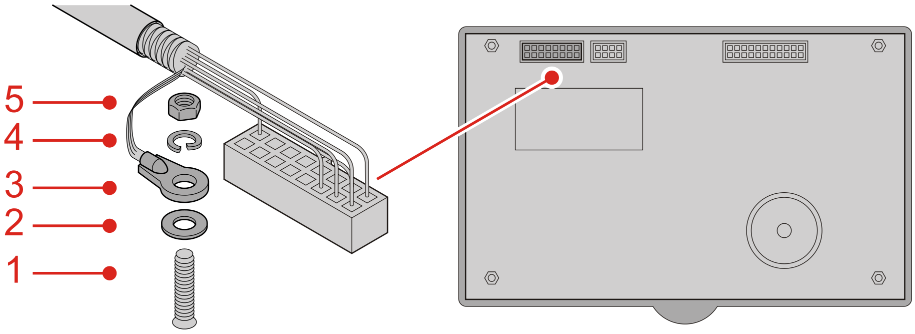
When disposing of devices, the local environmental regulations must be complied with.

## Instructions on EMI

**Wiring Instructions - CAN Cable**

To meet the requirements with regard to EMI the shielding of the CAN cable must be connected to the housing of the device. If you connect only pin 16 (shield), effective shielding is not ensured.

Connect the shielding of the CAN cable to the stud bolt of the device housing:



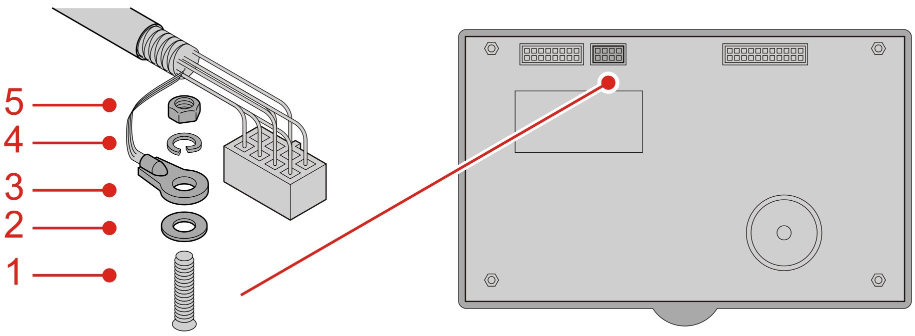

Caption:

| Number | Element |
|---|---|
| **1** | Threaded stud of the device housing |
| **2** | Plain washer |
| **3** | Wire lug |
| **4** | Lock washer |
| **5** | Nut |

**Wiring Instructions - Video Cable**

To meet the requirements with regard to EMI the shielding of the Video cable must be connected to the housing of the device. If you connect only ground connections (pin 4 and pin 7), effective shielding is not ensured.

Connect the shielding of the video cable to the stud bolt of the device housing:



Caption:

| Number | Element |
|---|---|
| **1** | Threaded stud of the device housing |
| **2** | Plain washer |
| **3** | Wire lug |
| **4** | Lock washer |
| **5** | Nut |

# 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**

## Product Description - JVM-407

**HMI JVM-407**

The HMI JVM-407 is extremely versatile thanks to its compact design and the integrated controller.

The JVM-407 can replace a complete instrument cluster.

The JVM-407 has been specially designed for use in the harsh environment of commercial vehicles and mobile machines.

**Product Features**

The features of this product are listed below:

- Display: 7" TFT with LED backlight
- Resolution: WVGA (800 x 480 pixels)
- 4 function keys
- 1 digipot
- 10 status LEDs available for selection via 10 digital inputs, rated for a power supply of 12 V and 24 V on the vehicle
- Adjustable night-lighting
- Buzzer (93 dB)
- Powerful programming language JetSym STX
- Non-volatile registers: 6,000
- RAM memory: 16 MBytes
- Flash memory: 64 MBytes
- 1 Ethernet interface
- 3 CAN-2.0B interfaces
- An additional 5 digital inputs freely available, rated for a power supply of 12 V and 24 V on the vehicle
- 1 protected digital output, 3 A
- 1 composite color signal (FBAS) video input for rearview camera
- 1 USB port
- SD card slot for SD cards up to 8 GBytes
- Real-time clock with battery backup
- Integrated Web server / e-mail feature
- Modbus/TCP

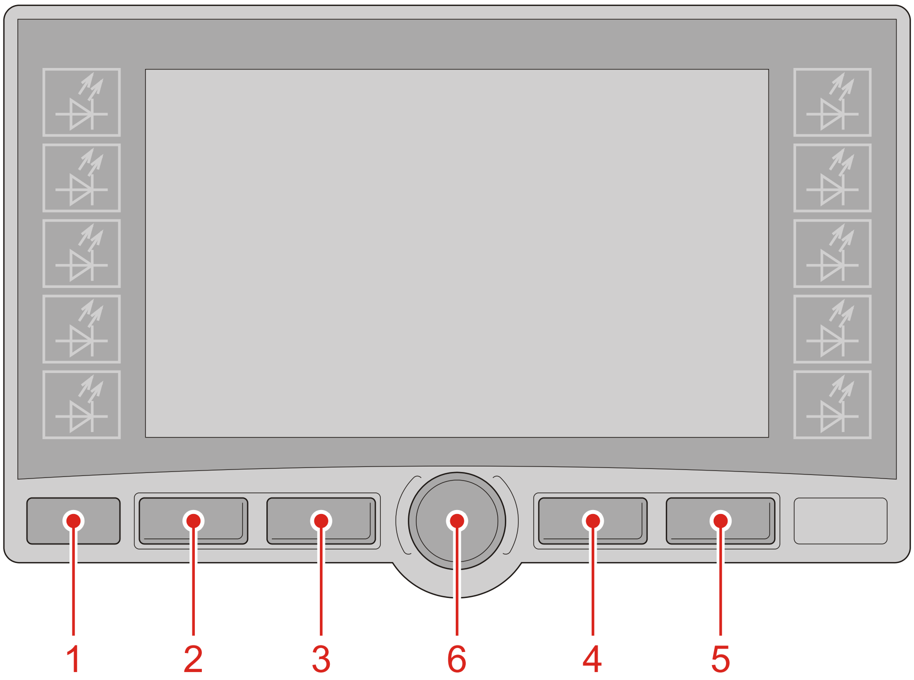
## Parts and Interfaces

**Introduction**

This chapter describes the parts and interfaces for the JVM-407.

**Controls**

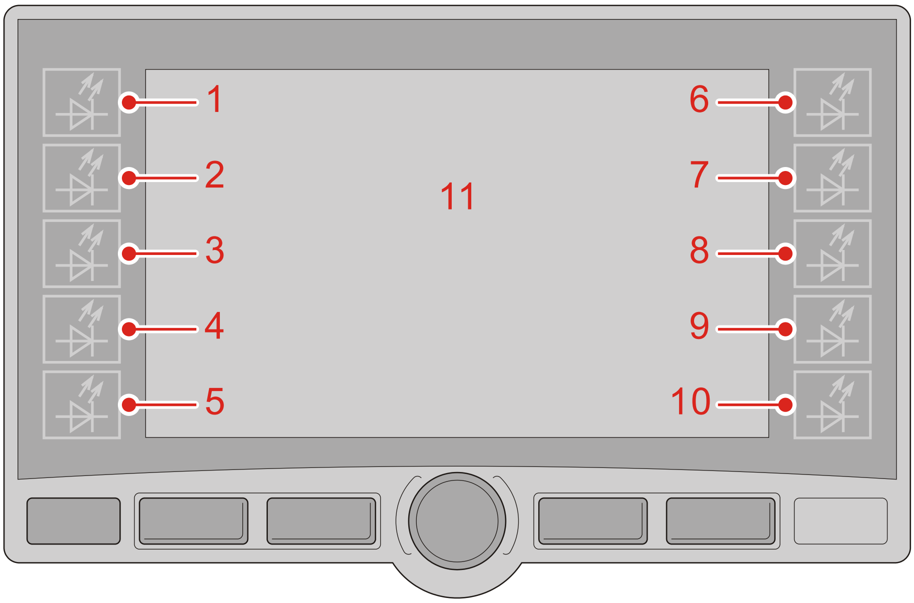
The diagram shows the controls on the front panel.




<table>
<tr><th>Number</th><th colspan="2">Control</th></tr>
<tr><td>1</td><td colspan="2">USB port behind protective cover</td></tr>
<tr><td>2</td><td>Function key F1</td><td rowspan="5">The digipot functions and the function keys are defined by the customer in the program.</td></tr>
<tr><td>3</td><td>Function key F2</td></tr>
<tr><td>4</td><td>Function key F3</td></tr>
<tr><td>5</td><td>Function key F4</td></tr>
<tr><td>6</td><td>Digipot (control dial with pushbutton feature)</td></tr>
</table>

**Displays**

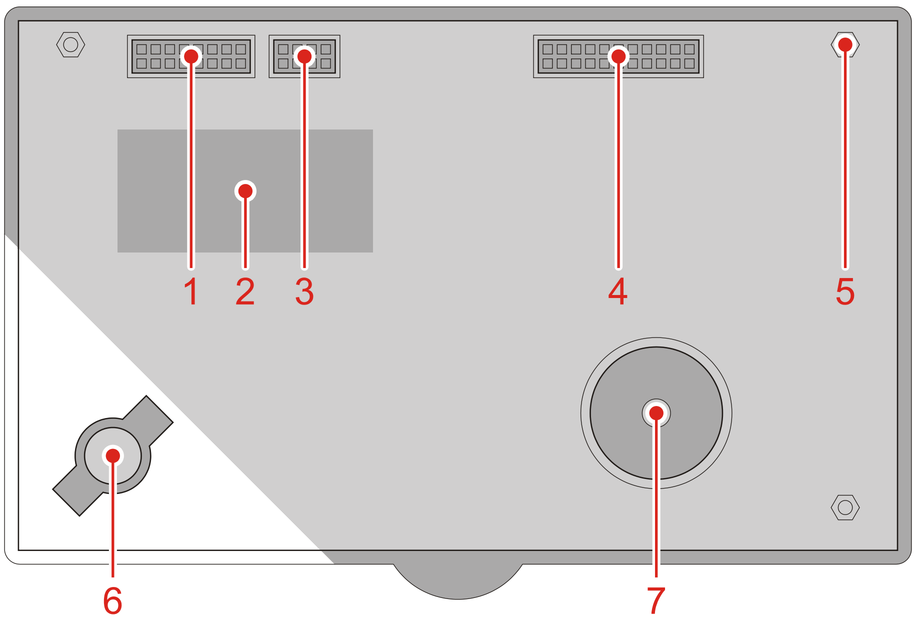
The diagram shows the display elements on the front panel.




<table>
<tr><th>Number</th><th colspan="2">Control or display element</th></tr>
<tr><td>1</td><td>LED 1</td><td rowspan="10">The LEDs illuminate the pictograms on the display.</td></tr>
<tr><td>2</td><td>LED 2</td></tr>
<tr><td>3</td><td>LED 3</td></tr>
<tr><td>4</td><td>LED 4</td></tr>
<tr><td>5</td><td>LED 5</td></tr>
<tr><td>6</td><td>LED 6</td></tr>
<tr><td>7</td><td>LED 7</td></tr>
<tr><td>8</td><td>LED 8</td></tr>
<tr><td>9</td><td>LED 9</td></tr>
<tr><td>10</td><td>LED 10</td></tr>
<tr><td>11</td><td colspan="2">Display screen</td></tr>
</table>

**Connectors and Parts on the Rear Panel**

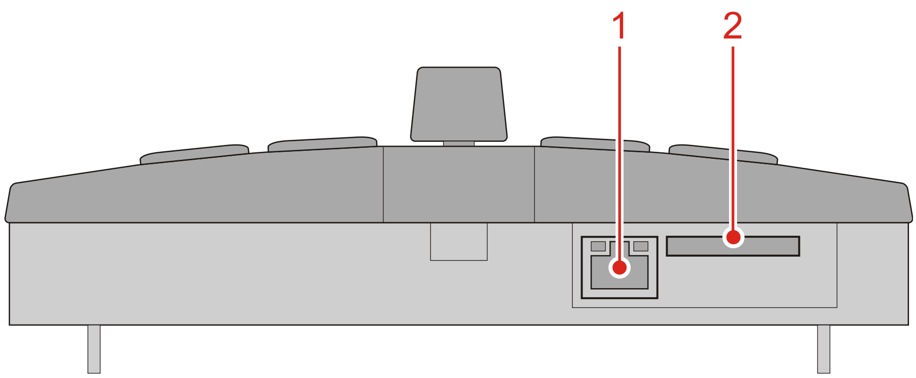
The diagram shows the connectors and parts on the rear panel.




| Number | Connector or part |
|---|---|
| 1 | CANopen® connector |
| 2 | Name plate |
| 3 | Video connector |
| 4 | Power supply connector, inputs and outputs |
| 5 | 4 threaded pins for installation panel |
| 6 | Backup battery on the circuit board |
| 7 | Buzzer |

**Connectors and Parts on the Underneath Panel**

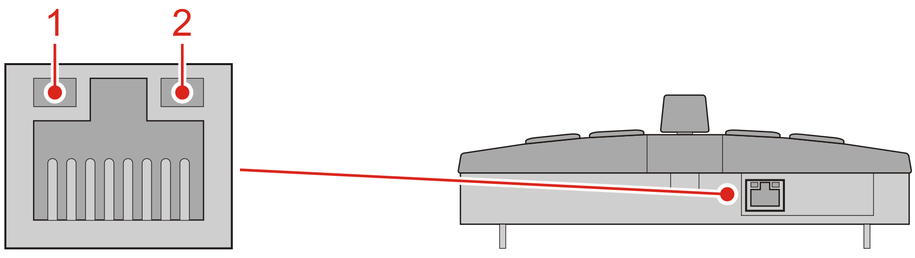
The diagram shows the connectors and parts on the underneath panel.



| Number | Connector or part |
|---|---|
| 1 | Connector jack for the ethernet cable |
| 2 | SD memory card slot |

**LEDs on the Underneath Panel**

The diagram shows the LEDs for the connector jack for the Ethernet cable.




| Number | Color | Description |
|---|---|---|
| 1 | amber | blinks when active (data transfer) |
| 2 | green | lights up when connection established |

## Order Reference / Options

**Order Reference**

The following variants exist for the JVM-407. They can be ordered from Jetter AG using the following part numbers.

| Part Number | Order Reference | Name |
|---|---|---|
| 10000821 | JVM-407-K00-O01 | HMI with support arm |
| 10000822 | JVM-407-K00-O12 | HMI for panel mounting |

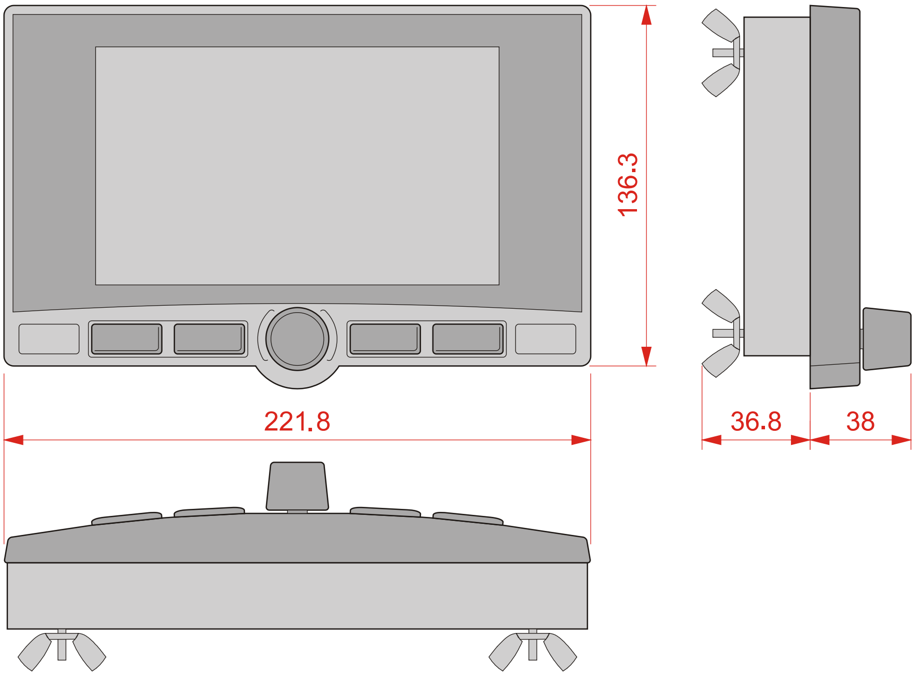
## Physical Dimensions

**Introduction**

This chapter details the physical dimensions of the JVM-407 and the conditions for installation.

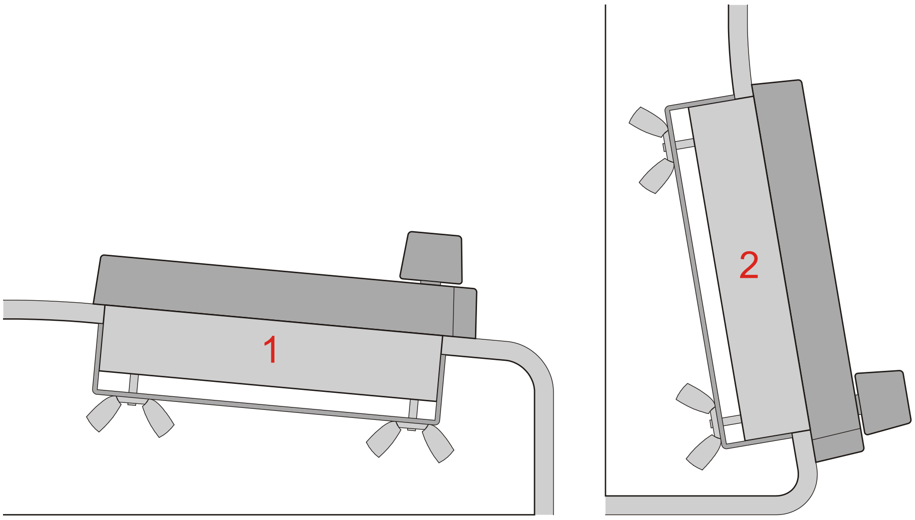
**Physical Dimensions**

The diagram shows the dimensions of the JVM-407.




**Permissible Installation Positions**

The diagram shows the positions permitted for installation.

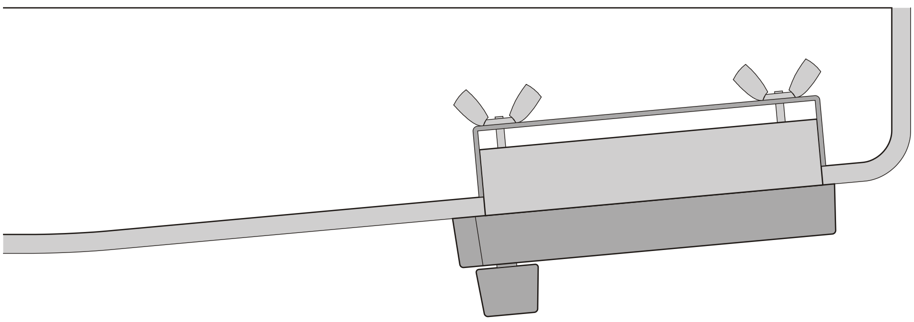


Explanations are as follows:

| Number | Permissible Installation Positions |
|---|---|
| 1 | horizontally or tilted |
| 2 | vertical or tilted |

**Prohibited Installation Positions**

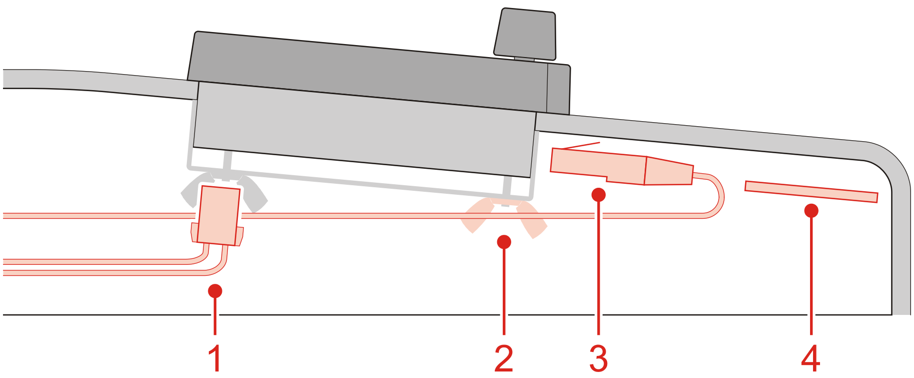
The diagram shows the positions prohibited for installation.



The rear panel of the HMI JVM-407 has no moisture protection, particularly against spray or water droplets. If the installation location cannot be guaranteed to be moisture-free, this method of installation (see diagram above) is prohibited. The accumulation of moisture and water droplets in the device can lead to current leakages and corrosion.

**Space Required for Installation and Service**

The diagram shows the space required for the HMI JVM-407.




Ensure there is enough space around the housing for servicing requirements.

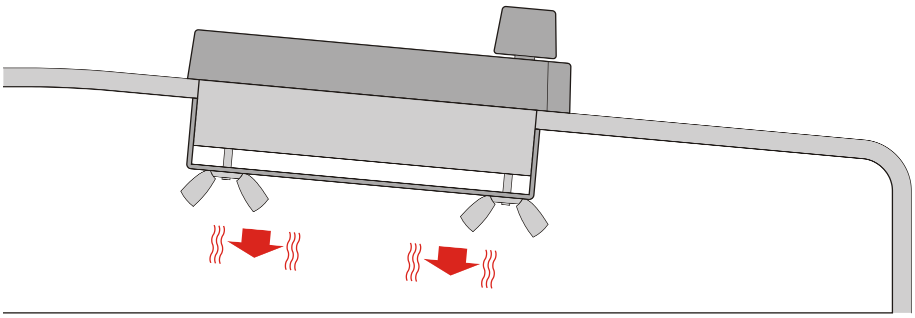
- It should be possible to disconnect the connector at any time.
- It should be possible to exchange the SD card at any time.
- It must be possible to easily loosen the wing nut on the SD card locking device.

Explanations are as follows:

| Number | Description |
|---|---|
| 1 | Connectors for CANopen®, video, power supply, inputs and outputs |
| 2 | Wing nut to secure the SD card |
| 3 | Network connector |
| 4 | SD memory card |

**Space Required to Protect Against Overheating**

The diagram indicates the safe distance to protect against overheating.



Please note:

- The JVM-407 increases the temperature of the environment as a result of heat emission under load.
  Power consumption is 7.8 W.
- The JVM-407 operates without interruption at an ambient temperature of up to +65 °C.

Consider the heat emission from the device, in particular when installing it in a critical environment:

- in the vicinity of the fuel tank
- in the vicinity of the fuel pipe
- in the vicinity of flammable vehicle components
- in the vicinity of thermally malleable vehicle components

**Installation Location**

The JVM-407 must be installed in the driver's cab.

# 3 Identifying the JVM-407

**Purpose of this Chapter**

This chapter is for supporting you in identifying the following information with regard to JVM-407:

- Hardware revision.
- Electronic data sheet (EDS). Numerous manufacturing-relevant data are stored to EDS.
- OS release of the controller and software components.

**Prerequisites**

To be able to identify technical data about the HMI JVM-407 the following prerequisites must be fulfilled:

- The controller is connected to a PC.
- The programming tool JetSym 4.1.2 or higher is installed on the PC.

**Information for Hotline Requests**

If you have to contact the hotline of Jetter AG in case of a problem, please have the following information on the HMI JVM-407 ready:

- Serial number
- OS release of the HMI
- Hardware revision

**Contents**

# 3.1 Identification by Means of the Nameplate

**Introduction**

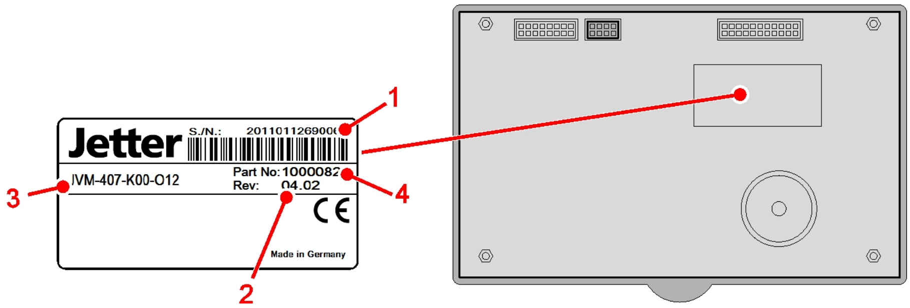
Each HMI JVM-407 can be identified by its nameplate attached to its enclosure. If you have to contact the hotline of Jetter AG in case of a problem, you need to have the hardware revision data and the serial number at hand.

**Contents**

## Nameplate

**Nameplate**

The nameplate of a JVM-407 contains the following information:




| Number | Description |
|---|---|
| **1** | Serial number |
| **2** | Hardware revision |
| **3** | HMI |
| **4** | Part number |

## 3.2 Version Registers

**Introduction**

The operating system of the JVM-407 provides several registers which can be used to read out the version numbers of the OS and its components. You will need this information when contacting the hotline of Jetter AG in case of a problem.

**Contents**

## Software Versions

### Introduction

The HMI JVM-407 features software with unique version numbers which can be read out via special registers.

### Format of Software Version Numbers

The software version number of the HMI JVM-407 is a four-figure value.

| 1 | . | 2 | . | 3 | . | 4 |
|---|---|---|---|---|---|---|

| Entry | 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 value zero.

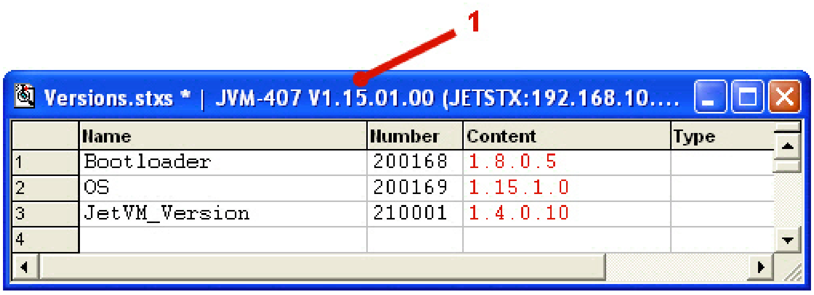
### Overview of Registers

The following registers are used for reading out software versions:

| Register | Description |
|---|---|
| **200168** | Boot loader version |
| **200169** | Operating system version |
| **210001** | Version of the execution unit for the STX application program (JetVM version) |

### Version Numbers in JetSym Setup

The following screenshot shows a JetSym setup window displaying version registers. For displaying the version number in the setup window of JetSym, please select the format "IP address".




| Number | Entry | Function |
|---|---|---|
| 1 | V 1.15.01.00 | OS version of the HMI.<br>JetSym displays this information in the title bar of each setup window. |

# 4 Installing the JVM-407

**Purpose of this Chapter**

This chapter supports the installing of the HMI JVM-407 in the vehicle as regards the following points:

- Wiring layout for the JVM-407
- Installation
- Configuration of the IP interface for the JVM-407

**Contents**

# 4.1 Interfaces

**Connector for the Power Supply and the Digital Inputs/Outputs**

The connector has the following functions:

- Power supply for the JVM-407
- Digital I/Os

**Ethernet Interface**

The function of the RJ45 jack is as follows:

- Ethernet interface to a PC

**Three CAN interfaces**

The function of the CAN interfaces is as follows:

- Interface CAN 0: Configurable as CAN-PRIM interface
- Interfaces CAN 0 through CAN 2: Configurable as CANopen® bus interface

**Connector for Video Camera**

The connector has the following functions:

- Option to connect a video camera, e.g. a rearview camera, with a voltage rating of 12 VDC.

**Contents**

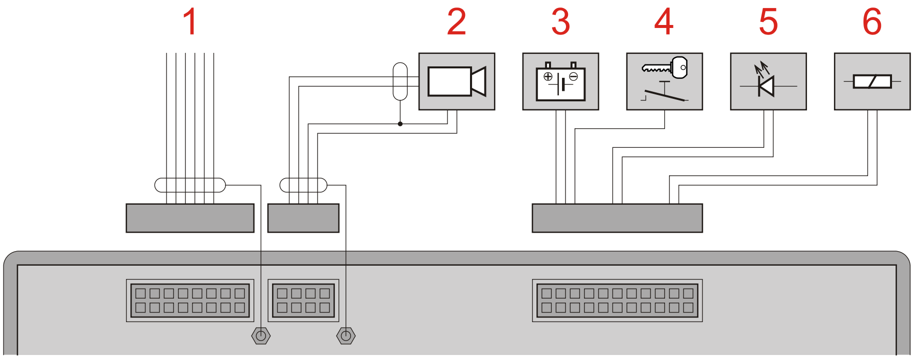
## Example of Wiring Layout

**Introduction**

This chapter uses an example to show how the JVM-407 is connected.

**Example**

The diagram shows an example of a wiring layout.




Explanations are as follows:

| | |
|---|---|
| **1** | CANopen® bus |
| **2** | Video camera |
| **3** | Power supply (battery) |
| **4** | Ignition lock |
| **5** | Input to control the display LEDs |
| **6** | Output, e.g. to control a bypass relay |

# Connecting the Power Supply

**Introduction**

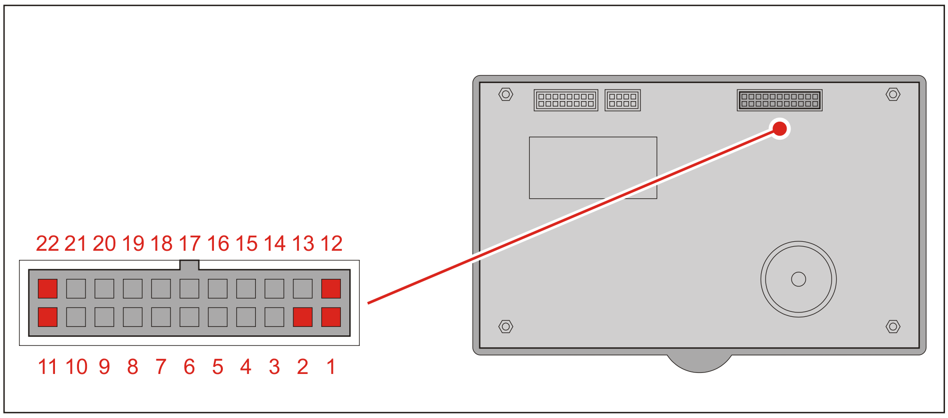
This chapter describes the pin assignment for the connector for the JVM-407 power supply. The connector type is the 22-pin Molex Micro-Fit 3.0 connector (manufacturer's item number 43045-2218).

**Use of the Connector**

This connector is also used for digital inputs and outputs.

**Power Supply**

The diagram shows the pin assignment for the connector for the power supply (cable panel view):




The pin assignment is as follows:

| Pin | Function | Terminal name in vehicle |
|---|---|---|
| 1 | Supply voltage +U BATT (+12 VDC or +24 VDC) | KL 30 |
| 2 | Ignition (+) | KL 15 |
| 11 | GND | KL 31 |
| 12 | Supply voltage U BATT (+12 VDC or +24 VDC) | KL 30 |
| 22 | GND | KL 31 |

**Important Note on Supply Voltage**

In order to halve the current load on pins 1 and 12, as well as on pins 11 and 22, all four pins should be connected to the supply voltage as per the above pin assignment.

**Note on Ignition**

To start the JVM-407, pin 2 (ignition +) must be connected with pin 1 or pin 12. The ignition control signal is issued when the key is in position "Ignition ON". When the key is in position "Ignition OFF", the JVM-407 is able to keep its status as ON.

**Technical Data**

| Parameter | Description |
|---|---|
| Rated voltage | DC 12 V or DC 24 V |
| Permissible voltage range | 9 ... 32 VDC |
| Input current without camera | typ. 650 mA for DC 12 V |
| Input current without camera | typ. 320 mA for DC 24 V |
| Power consumption without camera | 7.8 W |

**Note on Current Consumption**

When the JVM-407 is switched on, the current consumption is temporarily higher. To ensure that the JVM-407 can be activated, the supplied current should be at least 3-times the typical current.

**Mating Parts**

Compatible mating parts for the 22-pin Molex Micro-Fit 3.0 connector are as follows:

| | | |
|---|---|---|
|  | Manufacturer | Molex |
| | Manufacturer's item number - case | 43025-2200 |
| | Manufacturer's item number - crimp contact (jack) | 43030-0007 |
| | Diameter of the cable apt for connecting | 0.2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

| | | |
|---|---|---|
|  | Manufacturer | Würth |
| | Manufacturer's item number - case | 662 022 113 322 |
| | Manufacturer's item number - crimp contact (jack) | 662 001 137 22 |
| | Diameter of the cable apt for connecting | 0.2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

## Connecting Digital Inputs and Outputs

**Introduction**

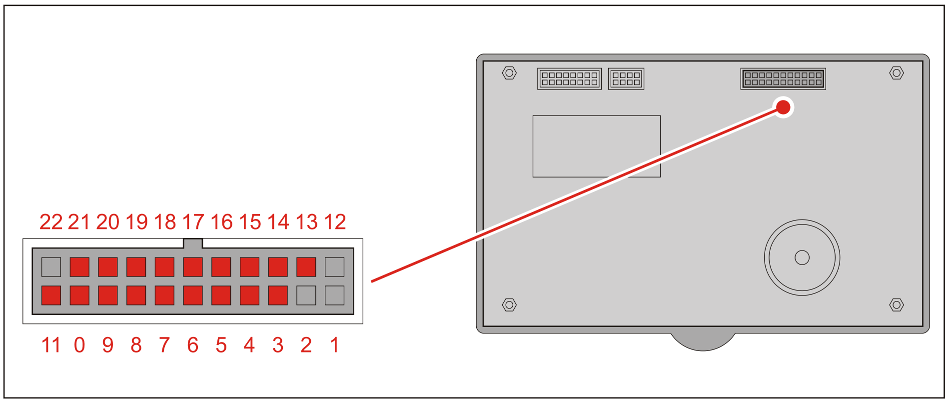
This chapter describes the pin assignment for the connector for the inputs and outputs on the JVM-407. The connector type is the 22-pin Molex Micro-Fit 3.0 connector (manufacturer's item number 43045-2218).

**Use of the Connector**

This connector is also used for the power supply.

**Pin Assignment of Inputs and Outputs**

The diagram shows the pin assignment for the connector for inputs and outputs (cable panel view):




The pin assignment is as follows:

| Pin | Function |
|---|---|
| **3** | Output 1 |
| **4** | Input # 2 |
| **5** | Input # 4 |
| **6** | Input # 6 |
| **7** | Input # 8 |
| **8** | Input # 10 |
| **9** | Input # 12 |
| **10** | Input # 14 |
| **11** | GND |
| **12** | Supply voltage (+12 VDC or +24 VDC) |
| **13** | Output 1 |
| **14** | Input # 1 |
| **15** | Input # 3 |
| **16** | Input # 5 |
| **17** | Input # 7 |
| **18** | Input # 9 |

| Pin | Function |
|---|---|
| **19** | Input # 11 |
| **20** | Input # 13 |
| **21** | Input # 15 |

**Note on Output 1**

Because output 1 can source a current of up to 3 A, output 1 has been assigned to pins 3 and 13. This halves the current load on the individual pins. For this reason, both pins need to be connected.

**Note on LEDs**

Because inputs 1 through 10 are directly connected with LEDs, the vehicle status can be displayed even when the display is disabled. A possible vehicle status can be e.g. full beam, flashing indicators, fault, etc.

**Technical Data of Digital Inputs**

| Parameter | Description |
|---|---|
| Type of inputs | Transistor, npn |
| Rated voltage | DC 9 ... 32 V |
| Threshold level OFF | ~ 8.5 V, 10 mA |
| Threshold level ON | ~ 8.3 V, min. 50 mA |
| Electrical isolation | none |

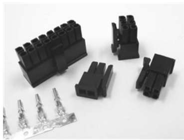
**Technical Data of Digital Outputs**

| Parameter | Description |
|---|---|
| Type of outputs | Transistor, pnp |
| Rated voltage | Supply voltage |
| Signal voltage OFF | < 1.0 V |
| Signal voltage ON | $U_{Supply}$ - 0.025 V |
| Load current | max. 3.0 A |

**Mating Parts**

Compatible mating parts for the 22-pin Molex Micro-Fit 3.0 connector are as follows:

| | | |
|---|---|---|
|  | Manufacturer | Molex |
| | Manufacturer's item number - case | 43025-2200 |
| | Manufacturer's item number - crimp contact (jack) | 43030-0007 |
| | Diameter of the cable apt for connecting | 0.2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

| | | |
|---|---|---|
|  | Manufacturer | Würth |
| | Manufacturer's item number - case | 662 022 113 322 |
| | Manufacturer's item number - crimp contact (jack) | 662 001 137 22 |
| | Diameter of the cable apt for connecting | 0.2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

## HMI Switch Off Delay

**Introduction**

This chapter describes how the switch off delay for the HMI JVM-407 is implemented.

**Objective**

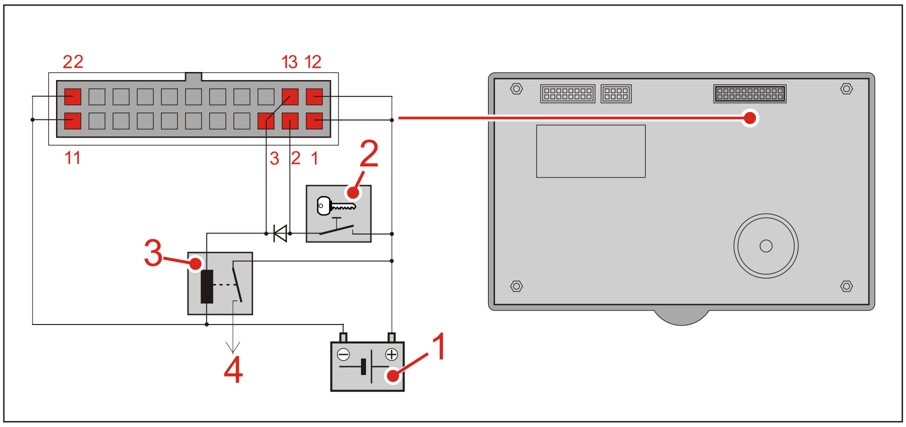
After switching off the ignition, the HMI should still remain switched on for a specific length of time. Only after this time has elapsed, it should switch itself off automatically.

**Duration of Switch Off Delay**

The duration for the switch off delay is defined in the JVM-407 program.

**Wiring**

The diagram shows the wiring for the switch of delay (cable panel view for jacks):




Explanations are as follows:

| | |
|---|---|
| **1** | Battery |
| **2** | Ignition lock |
| **3** | Bypass relay |
| **4** | Vehicle |

**Pin Assignment**

The pin assignment is as follows:

| Pin | Function | Terminal name in vehicle |
|---|---|---|
| 1 | Supply voltage +U BATT (+12 VDC or +24 VDC) | KL 30 |
| 2 | Ignition (+) | KL 15 |
| 3 | Bypass relay | - |
| 11 | GND | KL 31 |
| 12 | Supply voltage +U BATT (+12 VDC or +24 VDC) | KL 30 |
| 22 | GND | KL 31 |

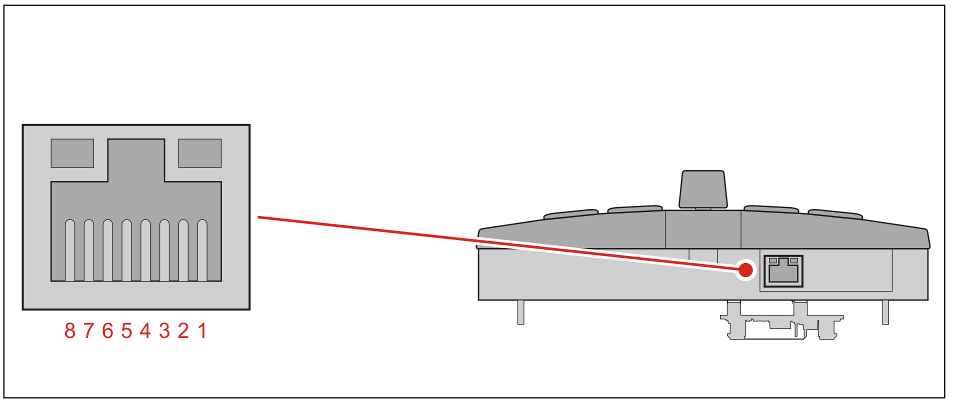
## Ethernet Interface

**Introduction**

This chapter describes the pin assignment for the connector jack for the ethernet cable on the HMI JVM-407.

**Pin Assignment of Ethernet Interface**

The diagram shows the pin assignment for the connector jack for the Ethernet cable:




The pin assignment is as follows:

| Pin | Function |
|---|---|
| 1 | TX+ |
| 2 | TX- |
| 3 | RX+ |
| 6 | RX- |

**Technical Data**

| Parameter | Description |
|---|---|
| Type of terminal | RJ45 Ethernet jack |
| Number of ports | 1 |
| Baud rate | 10 Mbit/s, 100 Mbit/s |
| Auto cross-over | Yes |

**Cable for Ethernet Interface**

For connecting devices to the ethernet interface, you can order the following cables separately from Jetter AG :

| Item # | Item |
|---|---|
| 60537500 | Patch cable 1:1, 1 m gray Hirose, Cat 5e, shielded |
| 60854512 | Patch cable 1:1, 2 m grey Hirose, Cat 5e, shielded |
| 60854514 | Patch cable 1:1, 5 m grey Hirose, Cat 5e, shielded |
| 60854515 | Patch cable 1:1, 10 m grey Hirose, Cat 5e, shielded |
| 60854078 | Patch cable cross-over, 1 m gray Hirose, Cat 5e, shielded |

| Item # | Item |
|---|---|
| 60851216 | Patch cable cross-over, 3 m blue Hirose, Cat 5e, shielded |
| 60854079 | Patch cable cross-over, 5 m gray Hirose, Cat 5e, shielded |

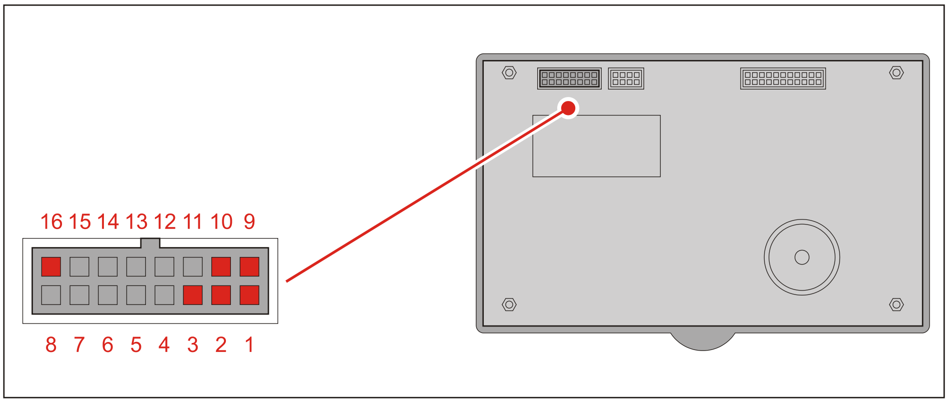
## CAN Interface

**Introduction**

This chapter describes the pin assignment for the connector for the CANopen® bus on the JVM-407. The connector type is the 16-pin Molex Micro-Fit 3.0 connector (manufacturer's item number 43045-1618).

**Pin Assignment CANopen® 0**

The diagram shows the pin assignment for the connector for the CANopen® bus 0 (cable panel view):

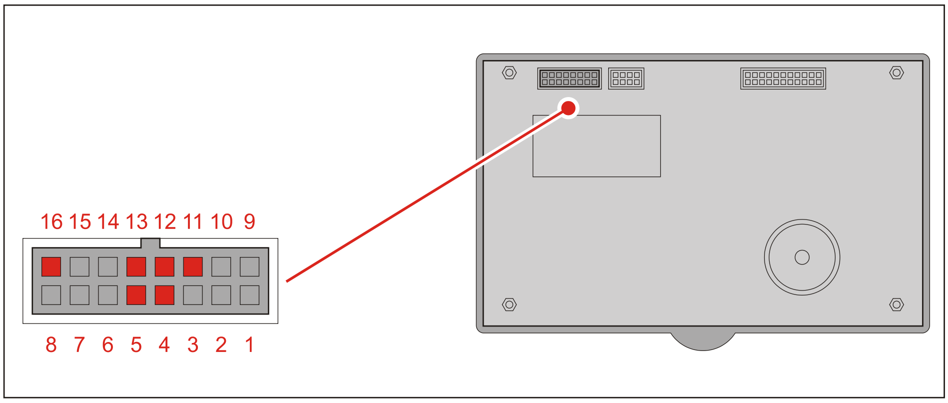



The pin assignment is as follows:

| Pin | Function |
|---|---|
| 1 | IN_CAN_0_H |
| 2 | TERM_CAN_0 |
| 3 | OUT_CAN_0_L |
| 9 | IN_CAN_0_L |
| 10 | OUT_CAN_0_H |
| 16 | Shield |

**Pin Assignment CANopen® 1**

The diagram shows the pin assignment for the connector for the CANopen® bus 1 (cable panel view):

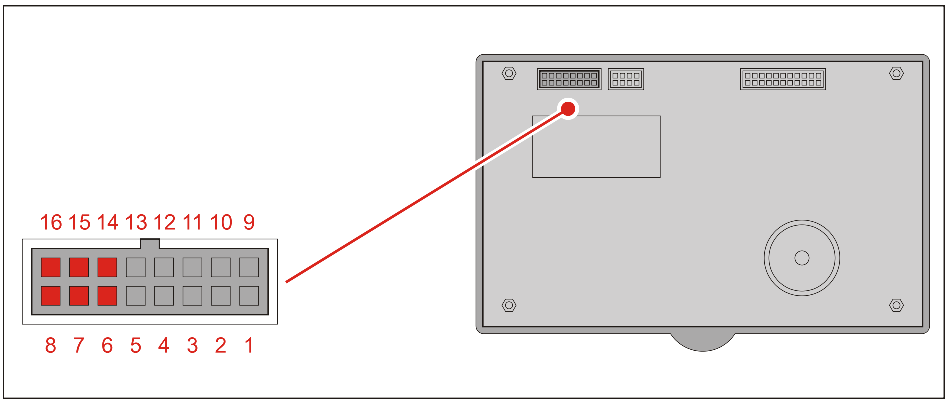



The pin assignment is as follows:

| Pin | Function |
|---|---|
| 11 | IN_CAN_1_H |
| 4 | IN_CAN_1_L |
| 12 | TERM_CAN_1 |
| 5 | OUT_CAN_1_H |
| 13 | OUT_CAN_1_L |
| 16 | Shield |

**Pin Assignment CANopen® 2**

The diagram shows the pin assignment for the connector for the CANopen® bus 2 (cable panel view):



The pin assignment is as follows:

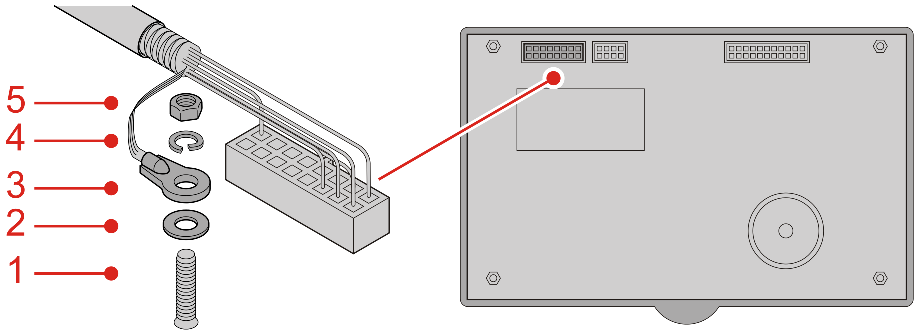
| Pin | Function |
|---|---|
| 6 | IN_CAN_2_H |
| 7 | TERM_CAN_2 |
| 8 | OUT_CAN_2_L |
| 14 | IN_CAN_2_L |
| 15 | OUT_CAN_2_H |
| 16 | Shield |

**Activating the Bus Termination Resistor**

To enable the resistor in the JVM-407 as the bus termination resistor, the TERM_CAN_x pin must be connected to the Pin OUT_CAN_x_H.

**Shield**

To satisfy EMC requirements, the CAN cable shield must be connected to the module housing. Connection of pin 16 (shield) alone is insufficient for effective shielding.
Connect the video cable shield to the threaded pins of the module housing:




Explanations are as follows:

| Number | Part |
|---|---|
| **1** | Threaded pins of the module housing |
| **2** | Washer |
| **3** | Cable lug |
| **4** | Lock washer |
| **5** | Screw nut |

**Mating Parts**

Compatible mating parts for the 16-pin Molex Micro-Fit 3.0 connector are as follows:

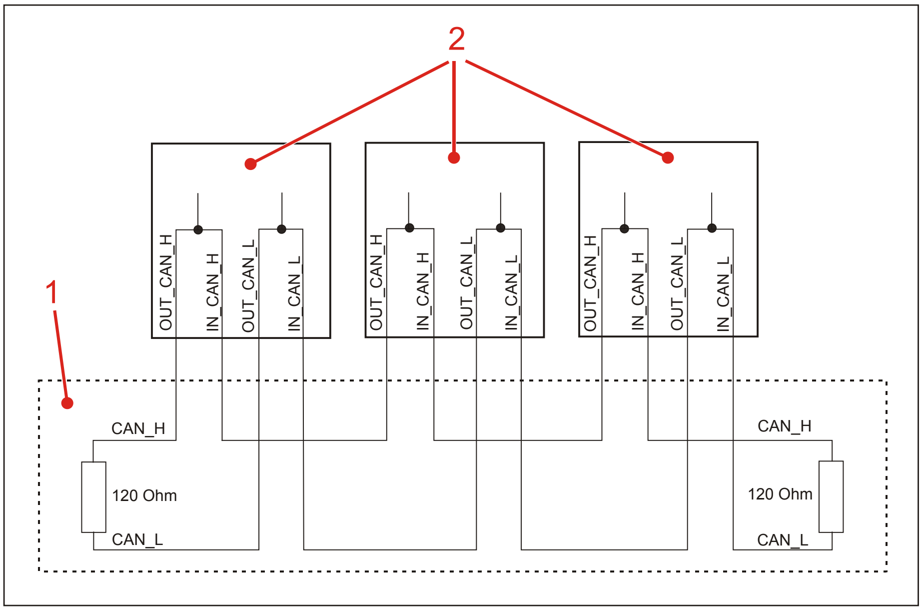
| | | |
|---|---|---|
|  | Manufacturer | Molex |
| | Manufacturer's item number - case | 43025-1600 |
| | Manufacturer's item number - crimp contact (jack) | 43030-0007 |
| | Diameter of the cable apt for connecting | 0,2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

| | | |
|---|---|---|
|  | Manufacturer | Würth |
| | Manufacturer's item number - case | 662 016 113 322 |
| | Manufacturer's item number - crimp contact (jack) | 662 001 137 22 |
| | Diameter of the cable apt for connecting | 0,2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

## Specification - CANopen® Bus Cable

**Layout of CAN Bus Wiring**

Jetter AG CANopen® devices are wired in accordance with the following diagram.




| Number | Description |
|---|---|
| **1** | CAN bus |
| **2** | Jetter AG CANopen® devices |

There is an option to enable a resistor in the device as a bus termination resistor of 120 Ohm.

The stub length with this type of wiring is practically zero.

The CAN_L and CAN_H cables must be twisted together.

**CAN Bus Cable Specification**

| Parameter | Description |
|---|---|
| Core cross-sectional area | 1000 kBaud: 0.25 ... 0.34 $mm^2$<br>500 kBaud: 0.34 ... 0.50 $mm^2$<br>250 kBaud: 0.34 ... 0.60 $mm^2$<br>125 kBaud: 0.50 ... 0.60 $mm^2$ |
| Cable capacitance | 60 pF/m max. |
| Resistivity | 1000 kBaud: max. 70 Ω/km<br>500 kBaud: max. 60 Ω/km<br>250 kBaud: max. 60 Ω/km<br>125 kBaud: max. 60 Ω/km |
| Number of cores | 2 |
| Shield | Complete shielding, no paired shielding |
| Twisting | Core pairs CAN_L and CAN_H are twisted |

**Cable Lengths**

The maximum permitted cable length depends on the baud rate used and the number of CANopen® devices connected.

| Baud Rate | Cable length | Stub length | Overall stub length |
|---|---|---|---|
| 1000 kBaud | max. 25 m | max. 0.3 m | 3 m |
| 500 kBaud | max. 100 m | max. 1.0 m | 39 m |
| 250 kBaud | max. 200 m | max. 3.0 m | 78 m |
| 125 kBaud | max. 200 m | - | - |

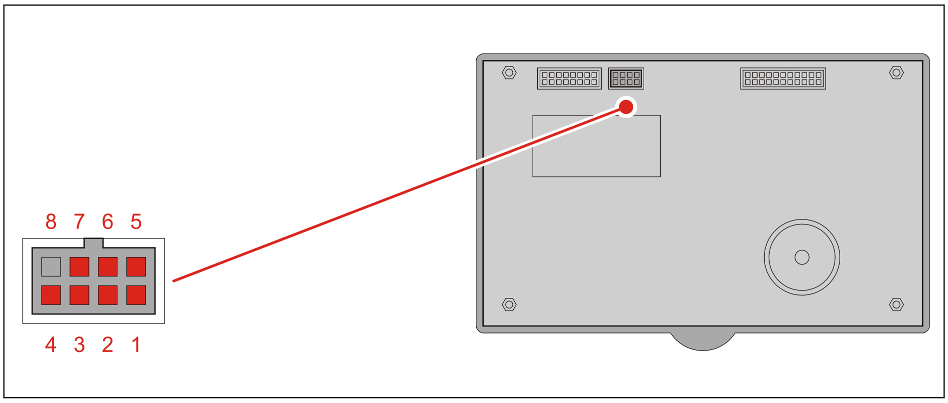
## Connecting a Video Camera

**Introduction**

This chapter describes the pin assignment for the connector for the video camera on the JVM-407. The connector type is the 8-pin Molex Micro-Fit 3.0 connector (manufacturer's item number 43045-0818).

**Pin Assignment - Video Input**

The diagram shows the pin assignment for the connector for the video camera (cable panel view):




The pin assignment is as follows:

| Pin | Function |
|---|---|
| 1 | Supply voltage (+12 VDC) e.g. for a camera |
| 2 | Video signal (+) |
| 3 | Shield |
| 4 | Ground (GND) |
| 5 | Video signal (-) |
| 6 | Ground (GND) |
| 7 | Video signal (-) |
| 8 | Reserved (do not connect!) |

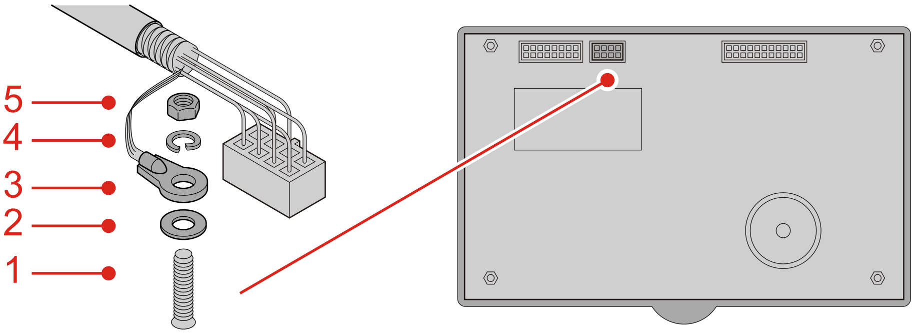
**Note on Video Signal**

If no differential video signal is used, video signal (-) (pin 7) and GND (pin 6) should be connected.

**Shield**

To satisfy EMC requirements, the video cable shield must be connected to the module housing. The ground connections (pin 4 and pin 7) are insufficient for effective shielding.

Connect the video cable shield to the threaded pins of the module housing:




Explanations are as follows:

| Number | Part |
|---|---|
| **1** | Threaded pins of the module housing |
| **2** | Washer |
| **3** | Cable lug |
| **4** | Lock washer |
| **5** | Screw nut |

**Technical Data**

| Parameter | Description |
|---|---|
| Power supply for a camera | DC 12 V, max. 1 A |
| Type of video input | analog, differential composite color signal (FBAS) video input with PAL signal or NTSC signal. |

**Mating Parts**

Compatible mating parts for the 8-pin Molex Micro-Fit 3.0 connector are as follows:

| | | |
|---|---|---|
|  | Manufacturer | Molex |
| | Manufacturer's item number - case | 43025-0800 |
| | Manufacturer's item number - crimp contact (jack) | 43030-0007 |
| | Diameter of the cable apt for connecting | 0.2 ... 0.5 mm$^2$ (AWG 24 ... 20) |

| | | |
|---|---|---|
|  | Manufacturer | Würth |
| | Manufacturer's item number - case | 662 008 113 322 |
| | Manufacturer's item number - crimp contact (jack) | 662 001 137 22 |
| | Diameter of the cable apt for connecting | 0.2 ... 0.5 $mm^2$ (AWG 24 ... 20) |

## 4.2 Interfaces on the Center Console with Mounted Support Arm

**Interconnecting Cable to the Center Console**

This chapter covers the layout of the connection cables already installed in the support arm, if the HMI JVM-407 is mounted on the support arm.

It also covers the connector types required to connect the JVM-407 to the center console.

In the support arm, connection cables have been installed for the following purposes:

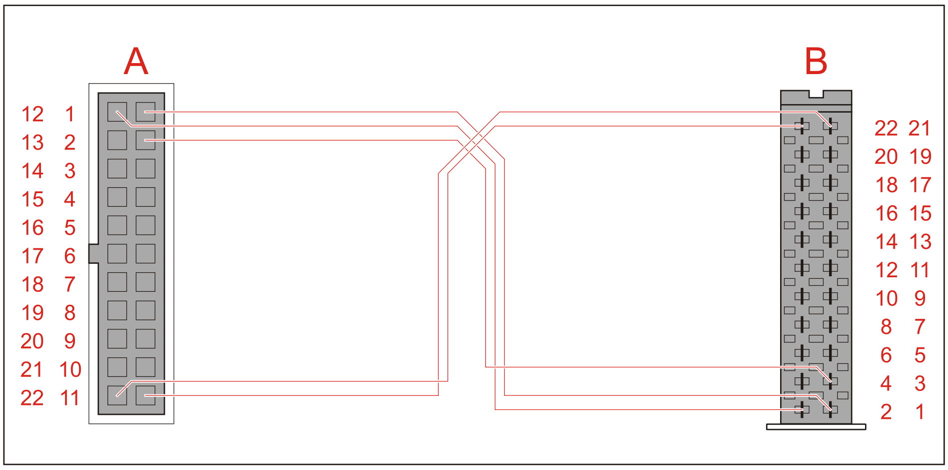
- Power supply
- Digital inputs/outputs
- CANopen® interfaces
- Video

**Contents**

## Connection Cable - Power Supply

**Wiring**

The diagram shows the wiring for the power supply in the support arm (cable panel view in each case):




Explanations are as follows:

| | |
|---|---|
| **A** | Connector for the HMI JVM-407 |
| **B** | Connector for the center console |

**Pin Assignment**

The pin assignment is as follows:

| Pin (A) | Function | Terminal name in vehicle | Pin (B) |
|---|---|---|---|
| 1 | +U BATT | KL 30 | 1 |
| 2 | Ignition (+) | KL 15 | 3 |
| 11 | GND | KL 31 | 21 |
| 12 | +U BATT | KL 30 | 2 |
| 22 | GND | KL 31 | 22 |

**Use of Connector B**

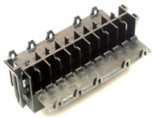
This connector is also used for digital inputs and outputs.

**Specification of Connector B**

| | |
|---|---|
| Type | AMP Junior Power Timer (male) |
| Number of pins | 22 |

**Mating part**

The following is a compatible mating part for the 22-pin connector AMP Junior Power Timer:

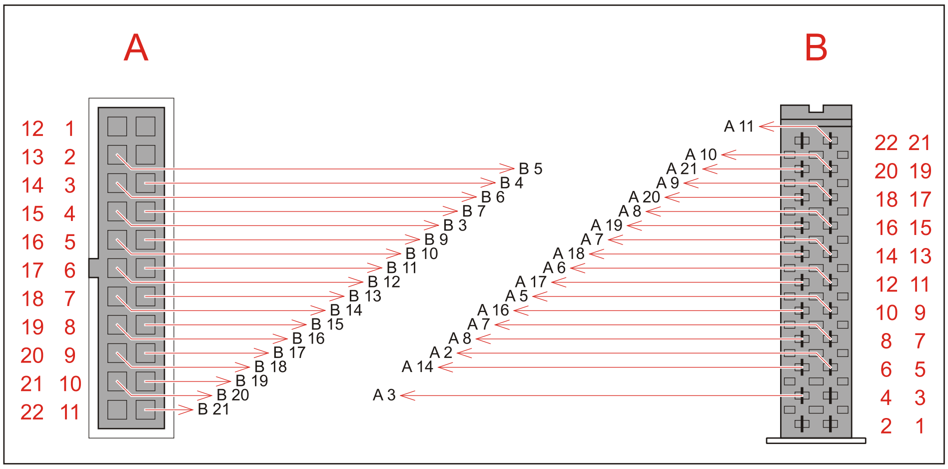


| | | |
|---|---|---|
| | Manufacturer | AMP |
| | Manufacturer's item number - Socket housing | 929504-7 |
| | Manufacturer's item number - Crimp contact (jack) | 927771 (reel)<br>927779 (single) |
| | Diameter of the cable apt for connecting | 0.5 ... 1.0 $mm^2$<br>(AWG 20 ... 16) |

## Connection Cable - Inputs and Outputs

**Wiring**

The diagram shows the wiring for the digital inputs and digital outputs in the support arm (cable panel view in each case):




Explanations are as follows:

| | |
|---|---|
| **A** | Connector for the HMI JVM-407 |
| **B** | Connector for the center console |

**Pin Assignment**

The pin assignment is as follows:

| Pin (A) | Function | | Pin (B) |
|---|---|---|---|
| 3 | Output 1 | | 4 |
| 4 | Input # 2 | To control the LEDs for pictogram illumination on the display area | 7 |
| 5 | Input # 4 | | 9 |
| 6 | Input # 6 | | 11 |
| 7 | Input # 8 | | 13 |
| 8 | Input # 10 | | 15 |
| 9 | Input # 12 | For free use | 17 |
| 10 | Input # 14 | | 19 |
| 11 | GND | | 21 |
| 13 | Output 1 | | 5 |
| 14 | Input # 1 | To control the LEDs for pictogram illumination on the display area | 6 |
| 15 | Input # 3 | | 8 |
| 16 | Input # 5 | | 10 |
| 17 | Input # 7 | | 12 |
| 18 | Input # 9 | | 14 |

| Pin (A) | Function | | Pin (B) |
|---|---|---|---|
| 19 | Input # 11 | For free use | 16 |
| 20 | Input # 13 | | 18 |
| 21 | Input # 15 | | 20 |

**Note on Output 1**

Because the ignition coil is controlled via output 1 and consumes a high volume of power, output 1 has been assigned to pins 3 and 13 (A) or 4 and 5 (B). This halves the current load on the individual pins. For this reason, both pins need to be connected.

**Use of Connector B**

This connector is also used for the power supply.

**Specification of Connector B**

| | |
|---|---|
| Type | AMP Junior Power Timer (male) |
| Number of pins | 22 |

**Mating part**

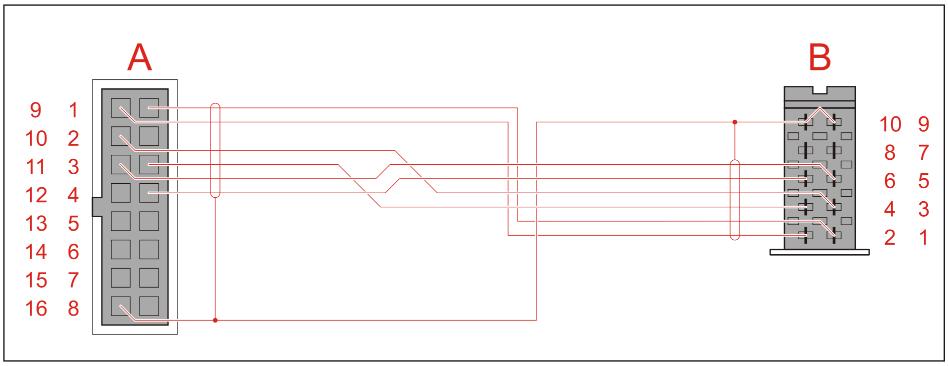
The following is a compatible mating part for the 22-pin connector AMP Junior Power Timer:

| | | |
|---|---|---|
|  | Manufacturer | AMP |
| | Manufacturer's item number - Socket housing | 929504-7 |
| | Manufacturer's item number - Crimp contact (jack) | 927771 (reel)<br>927779 (single) |
| | Diameter of the cable apt for connecting | 0.5 ... 1.0 mm$^2$<br>(AWG 20 ... 16) |

## Connection Cable - CANopen®

**Wiring**

The diagram shows the wiring for the CAN cable in the support arm (cable panel view in each case):




CAN_0_H and CAN_0_L must be twisted together.
CAN_1_H and CAN_1_L must be twisted together.

Explanations are as follows:

| | |
|---|---|
| **A** | Connector for the HMI JVM-407 |
| **B** | Connector for the center console |

**Pin Assignment**

The pin assignment is as follows:

| Pin (A) | Function | Pin (B) |
|---|---|---|
| 1 | IN_CAN_0_H | 1 |
| 2 | TERM_CAN_0 | |
| 3 | OUT_CAN_0_L | 4 |
| 4 | IN_CAN_1_L | 6 |
| 5 | OUT_CAN_1_H | |
| 6 | IN_CAN_2_H | |
| 7 | TERM_CAN_2 | |
| 8 | OUT_CAN_2_L | |
| 9 | IN_CAN_0_L | 2 |
| 10 | OUT_CAN_0_H | 3 |
| 11 | IN_CAN_1_H | 5 |
| 12 | TERM_CAN_1 | |
| 13 | OUT_CAN_1_L | |
| 14 | IN_CAN_2_L | |

| Pin (A) | Function | Pin (B) |
|---|---|---|
| 15 | OUT_CAN_2_H | |
| 16 | Shield | 9 |
| | | 10 |

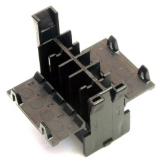
CAN 1 is terminated on the connector for JVM-407 (jumper between pin 6-8).

**Specification of Connector B**

| | |
|---|---|
| Type | AMP Junior Power Timer (male) |
| Number of pins | 10 |

**Mating part**

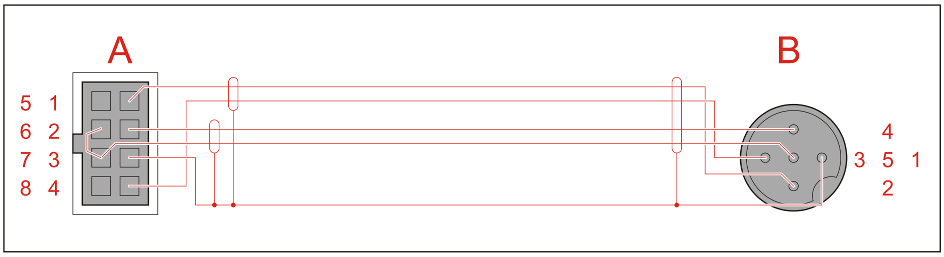
The following is a compatible mating part for the 10-pin connector AMP Junior Power Timer:

| | | |
|---|---|---|
|  | Manufacturer | AMP |
| | Manufacturer's item number - Socket housing | 929504-4 |
| | Manufacturer's item number - Crimp contact (jack) | 927771 (reel)<br>927779 (single) |
| | Diameter of the cable apt for connecting | 0.5 ... 1.0 $mm^2$<br>(AWG 20 ... 16) |

## Connection Cable - Video

**Wiring**

The diagram shows the wiring for the video cable in the support arm (cable panel view in each case):




Explanations are as follows:

| | |
|---|---|
| **A** | Connector for the HMI JVM-407 |
| **B** | Connector for the center console |

**Pin Assignment**

The pin assignment is as follows:

| Pin (A) | Function | Pin (B) |
|---|---|---|
| 1 | Supply voltage +12 V | 2 |
| 2 | Video signal (+) | 4 |
| 3 | Shield | 1 |
| 4 | Ground | 3 |
| 5 | Video signal (-) | 5 |
| 6 | Ground | 3 |
| 7 | Video signal (-) | 5 |
| 8 | Reserved (do not connect!) | |

**Note on Video Signal**

By default, cables in the support arm are connected to A pin 6 and 7, i.e. no differential video signal is used.

**Specification of Connector B**

| | |
|---|---|
| Type | Jack M12 |
| Number of pins | 5 |

**Mating Part**

The 5-pin M12 connector is a compatible mating part.

## 4.3 Installing the JVM-407

**Introduction**

This chapter describes how to install the JVM-407.

**Contents**

## Installing the HMI

**Introduction**

This chapter describes how to install the HMI JVM-407.

**Selecting a Place for Installation**

Select a suitable place for the device to be mounted.
A place is suitable if it fulfils the following requirements:

- The installation surface must be made from one of the following materials:
  - aluminum plate
  - galvanized steel plate
  - lacquered steel plate
  - plastic
- The installation surface must be level.
- The installation surface should be no more than 5 mm thick.
- The installation location must allow air to circulate.
- The installation location must be accessible for servicing.
- The installation location must be of sufficient size.

**Avoiding Unsuitable Installation Locations**

Do not install the device in inappropriate locations.
The following installation locations are unsuitable for mounting the HMI:

| Unsuitable installation location | Reason |
|---|---|
| Outdoor installation | The HMI must not be exposed to rain or a jet of water. Therfore, do not use a steam jet or other such devices to clean the HMI. |
| Unventilated installation location | The HMI could overheat as heat builds up. |
| Installation location close to heat-sensitive materials | The materials could become warped or misshapen as a result of heat produced by the HMI. |
| Installation surfaces are uneven | The installation surface could become misshapen when fitting the HMI.<br>Installation is unstable and precarious. |

**Consider Ergonomic Principles**

Consider ergonomic principles.
Select a user-friendly place for installation:

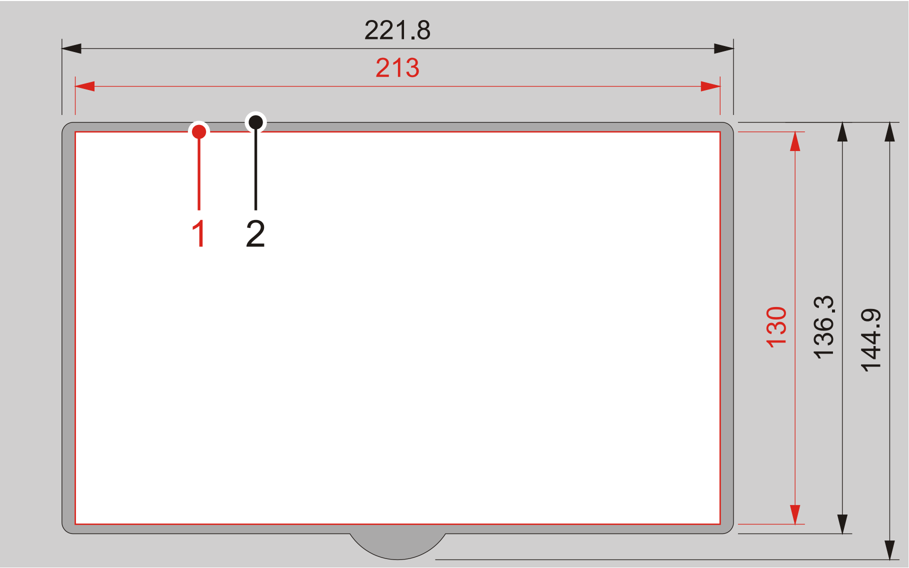
- The controls must be easy to reach.
- The HMI screen must be easy to read.

Avoid installation locations that are ergonomically unsuitable:

- Extreme angles, which could make it difficult to see the HMI
- Unsuitable lighting conditions with reflection and glare
- Concealed installation locations that are difficult for the user to access

**Preparing for Installation**

Make a square opening.
The diagram shows the dimensions:

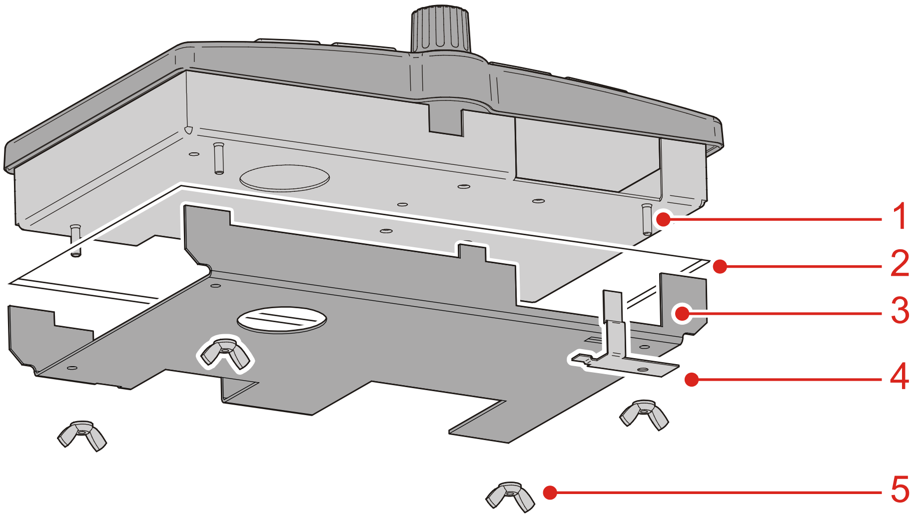



Explanations are as follows:

| | |
|---|---|
| **1** | Opening |
| **2** | Outline of the front panel |

**Installing the HMI**

The diagram shows how to install:




Explanations are as follows:

| | |
|---|---|
| **1** | Threaded pins on the JVM-407 housing |

| | |
|---|---|
| **2** | Opening |
| **3** | Fitting panel |
| **5** | SD card holder |
| **6** | 4 x wing nut |

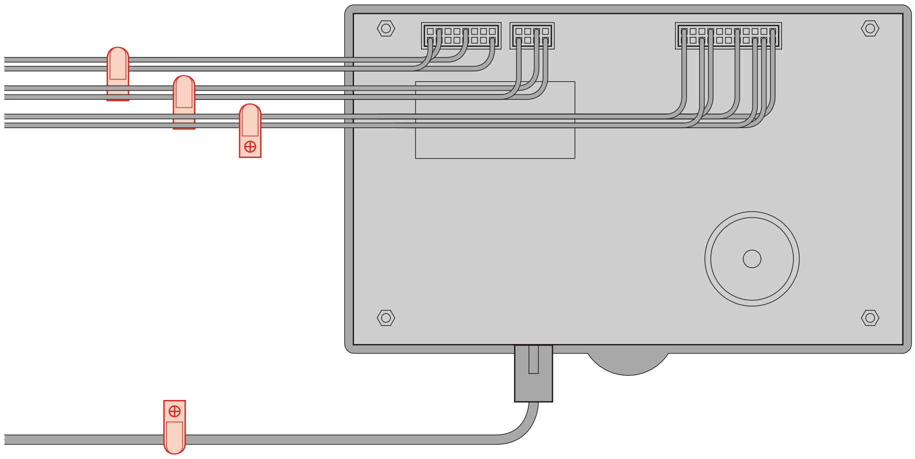
| Step | Action |
|---|---|
| **1** | Insert the HMI into the front of the opening. |
| **2** | Attach the fitting panel at the back.<br>Ensure it is correctly positioned: Hole over the buzzer. |
| **3** | Push the SD card holder onto the threaded pins for the SD card slot. |
| **4** | Screw the holder firmly into place with a wing nut. |
| **5** | Screw the fitting panel firmly into place with the remaining three wing nuts. |

**Installing the Strain Relief**

Install the strain relievers for the connection cable.

Take care to leave enough space for the connectors.

Connectors should not be obstructed, so that they can be removed in the event of a service requirement.



# Mounting the Support Arm

**Introduction**

This chapter describes how to mount the support arm for the HMI JVM-407.

**Selecting a Place for Installation**

Select a suitable place for the device to be mounted.
A place is suitable if it fulfils the following requirements:

- The installation surface must be level. The surface should not be uneven.
- The mounting surface must be rigid to be able to withstand the leverage force of the support arm.
- Underneath the mounting area, there must be enough space for cable guides.
- The installation location must be accessible for tightening and loosening screws.

**Consider Ergonomic Principles**

Consider ergonomic principles.
Select a user-friendly place for installation.

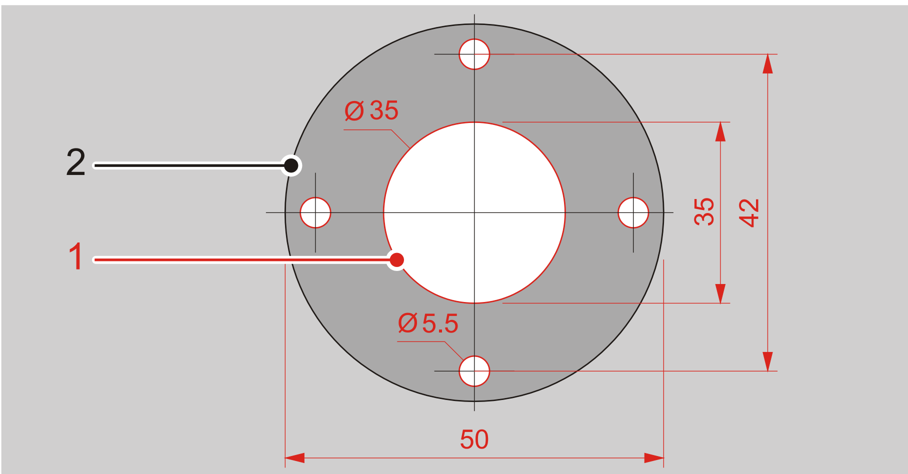
- The controls must be easy to reach.
- The HMI screen must be easy to read.

Avoid installation locations that are ergonomically unsuitable:

- Extreme angles, which could make it difficult to see the HMI.
- Unsuitable lighting conditions with reflection and glare
- Concealed installation locations that are difficult for the user to access

**Preparing for Installation**

Drill the following holes. The diagram shows the dimensions:



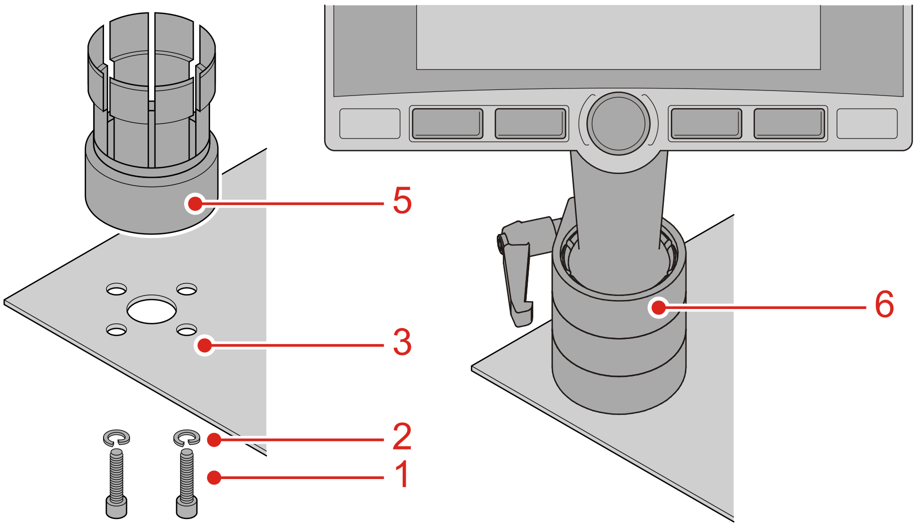

Deburr the holes.

Explanations are as follows:

| | |
|---|---|
| **1** | Holes for screws and cable feed-through |
| **2** | Outline of the support arm base |

**Mounting the Support Arm Base**

The diagram shows how to install:




Explanations are as follows:

| | |
|---|---|
| **1** | 4 x screws M 5 x 14 mm + thickness of mounting surface<br>(It is permitted to use e.g. hexagon socket head cap screws DIN 912) |
| **2** | 4 x lock washers |
| **3** | 4 x screw holes |
| **5** | Support arm base |
| **6** | Support arm |

| Step | Action |
|---|---|
| 1 | Place the support arm on the installation surface. |
| 2 | Screw the support arm firmly into place from underneath using four screws and four lock washers. |
| 3 | Guide the cable through the support arm base. |
| 4 | Push the support arm onto the support arm base. It must click into place. |
| 5 | Loosen the locking lever. |
| 6 | Adjust the angle of the HMI so that it is comfortable for the user. |
| 7 | Lock the HMI into place by tightening the locking lever. |

## 4.4 IP Configuration

**Introduction**

This chapter describes how the IP configuration for the HMI JVM-407 is implemented. To this end, the following parameters can be set:

- IP address of the HMI
- Subnet mask
- IP address of default gateway
- IP address of DNS server
- HMI name
- IP port number for the JetSym debugger
- Base port number for communication via JetIP

**Engineer's Skills Required**

To carry out IP configuration of the HMI JVM-407, knowledge of IP networks is required, for example:

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

**Contents**

## Factory Settings

**Introduction**

Before the HMI JVM-407 is shipped, various parameters are set to a certain value.

These parameters can be modified by the user.

**Factory Settings**

| Parameter | Value |
|---|---|
| IP address of the controller | 192.168.10.15 |
| Subnet mask | 255.255.255.0 |
| IP address of default gateway | 0.0.0.0 |
| IP address of DNS server | 0.0.0.0 |
| Controller name | JVM-4xx |
| IP port number for debugger | 52000 |
| IP port number for JetIP | 50000 |
| Administrator password | admin |
| System password | system |

## Configuration Memory

**Introduction**

The parameters for initializing the IP interface are uploaded from the configuration memory by the HMI during the boot process. Data stored to the configuration memory can be accessed in the following ways:

- Configuration data can be read out of a file located in the system directory of the file system. They can also be modified in this file.
- Configuration data can be read out via registers.

**Enabling Conditions**

The HMI reads data located in the configuration memory only during the boot process. If changes are made to the configuration memory, the HMI must be rebooted for these changes to become effective.

**Default Values**

Before data from the configuration memory are used, the HMI checks them for plausibility. If entries are invalid or absent, the following default values are used:

| Parameter | default |
|---|---|
| IP address of the controller | 192.168.10.15 |
| Subnet mask | 255.255.255.0 |
| IP address of default gateway | 0.0.0.0 |
| IP address of DNS server | 0.0.0.0 |
| Controller name | JVM-4xx |
| Suffix type of the name | 0 |
| IP port number for debugger | 52000 |
| IP port number for JetIP | 50000 |

**Related Topics**

- **Configuration File cfgvar.ini** on page 71
- **Configuration Registers** on page 75

## Configuration File cfgvar.ini

**Introduction**

The configuration file *cfgvar.ini* can be used to access the configuration memory of the JVM-407.

**Properties**

- The file can be accessed via the file system of the HMI.
- For an FTP connection, the user must have administrator or system rights.
- This file is located in the subdirectory "/System".
- This file cannot be deleted; it can only be overwritten.
- Formatting the flash disk has no impact on this file.

**File Structure**

This configuration file is a text file, the entries of which are grouped into several sections. For missing IP configuration parameters default values are used.

**Example for cfgvar.ini**

This is an example for a configuration file cfgvar.ini:

```
;JVM-407 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       = JVM-4xx

[PORTS]
JetIPBase = 50000
JVMDebug  = 52000
```

**Section [IP]**

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

| **Address** | |
|---|---|
| In the given example | 192.168.50.1 |
| Function | IP address of the HMI |
| Allowed values | ■ > 1.0.0.0<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address |
| in the event of an illegal value | the HMI will set all 4 values to their default values. |
| **SubnetMask** | |
| In the given example | 255.255.255.0 |
| Function | Subnet mask |
| Allowed values | ■ >= 128.0.0.0 |
| Illegal values | ■ 1 and 0 mixed |
| in the event of an illegal value | the HMI will set all 4 values to their default values. |
| **DefGateWay** | |
| In the given example | 192.168.50.11 |
| Function | IP address of the gateway to other subnets;<br>The HMI must be able to reach the subnet (Address/SubnetMask), otherwise it will set this parameter to 0.0.0.0. |
| Allowed values | ■ > 1.0.0.0 and<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address<br>■ A value (Address/SubnetMask) which cannot be reached by the HMI.<br>■ The address value |
| in the event of an illegal value | will be set to 0.0.0.0 by the controller. |
| **DNSServer** | |
| In the given example | 192.168.1.44 |
| Function | IP address of the server for the Domain Name System. |
| Allowed values | ■ 1.0.0.0<br>■ 223.255.255.255 |
| in the event of an illegal value | the HMI will set the value to 0.0.0.0 |

**Section [HOSTNAME]**

In the section [HOSTNAME] the name of the HMI is specified. The HMI is able to generate an individual name automatically. This host name is not currently used.

| **SuffixType** | |
|---|---|
| In the given example | 0 |
| Function | The type of the automatically generated suffix is attached to the HMI name. |
| Allowed values | ▪ 0: No attachment<br>▪ 1: Low-order byte of the IP address in decimal notation.<br>▪ 2: Low-order byte of the IP address in hexadecimal notation. |
| in the event of an illegal value | 0 |
| **Name** | |
| In the given example | JVM-4xx |
| Function | Specifies the HMI name. |
| Allowed values | ▪ First character: 'A' ... 'Z', 'a' ... 'z'<br>▪ Next character: 'A' ... 'Z', 'a' ... 'z', '0' ... '9', '-' |
| in the event of an illegal value | JVM-4xx |

**Section [PORTS]**

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

| **JetIPBase** | |
|---|---|
| In the given example | 50000 |
| Function | IP port for OS update and communication with the PC |
| Allowed values | ▪ 1024 ... 65535 |
| In the event of an illegal value | 50000 |
| **JVMDebug** | |
| In the given example | 52000 |
| Function | IP port for debugger/setup in JetSym |
| Allowed values | ▪ 1024 ... 65535 |
| In the event of an illegal value | 52000 |

**Changing IP Configuration**

| Step | Action |
|---|---|
| **1** | Use a text editor to create a configuration file on your PC named *cfgvar.ini* and make the corresponding entries. |
| **2** | Open an FTP connection between the PC and JVM-407. |
| **3** | Log in as user with administrator or system rights.<br>Standard:<br>User: admin; Password: admin |
| **4** | Browse to subdirectory "/System" of the JVM-407. |
| **5** | Copy the configuration file *cfgvar.ini* you created on the JVM-407. |
| **6** | Close the FTP connection. |
| **7** | Reboot the device.<br>**Result**: The new configuration is active. |

**Related Topics**


- **Configuration Memory** on page 70
- **Configuration Registers** on page 75

## Configuration Registers

**Introduction**

The IP configuration parameters can be read via configuration registers. A range of registers holds the data contained in the configuration memory. Another range contains the parameters actually used for initializing the IP interface.

**Register Numbers**

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

| HMI | Data Range | Basic Register Number | Register Numbers |
|---|---|---|---|
| JVM-407 | Configuration Memory | 101100 | 101100 ... 101165 |
| | Used parameter | 101200 | 101200 ... 101265 |

**Configuration Registers**

The following table provides an overview of the registers of both ranges, as well as their connection to the entries in the configuration file "/System/cfgvar.ini".

| Registers | Section in config.ini | Name in config.ini | Function |
|---|---|---|---|
| **MR 0** | IP | Address | IP address of the controller |
| **MR 1** | | SubnetMask | 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 to MR 51** | | Name | Specifies the controller name |
| **MR 64** | PORTS | JetIPBase | IP port for OS update and communication between controllers |
| **MR 65** | | JVMDebug | IP port for debugger/setup in JetSym |

**Related Topics**


- **Configuration Memory** on page 70
- **Configuration file cfgvar.ini** on page 71

## Modifying the IP Address of the Controller

**Introduction**

To be able to communicate with the JVM-407 via Ethernet, a unique IP address has to be set on the HMI.

**Configuration Options**

The IP address can be configured in the following ways:

- Default IP address
- Configuration via the file cfgvar.ini
- Configuration during runtime via special registers

**Related Topics**


- **Setting the IP Address via the File cfgvar.ini** on page 77
- **Setting the IP Address During Runtime** on page 78

## Setting the IP Address via the File cfgvar.ini

**The File cfgvar.ini**

The IP address of the HMI JVM-407 can be set using the "cfgvar.ini" file.

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

| Element | Function |
|---|---|
| Address | Line for entering the IP-address |
| aaa | 1st byte of IP address |
| bbb | 2nd byte of IP address |
| ccc | 3rd byte of IP address |
| ddd | 4th byte of IP address |

**Note**

The IP address setting in the file cfgvar.ini is only copied if the data in the configuration memory are not OK.

**Transmitting the File cfgvar.ini**

| Step | Action |
|---|---|
| 1 | Establish an FTP connection to the JVM-407. |
| 2 | Log in as user with administrator or system rights.<br>Standard:<br>User: admin; Password: admin (default) |
| 3 | Open the directory /System. |
| 4 | Copy the cfgvar.ini file into the directory /System. |
| 5 | Clear the FTP connection. |
| 6 | Restart the JVM-407. |

## Setting the IP Address During Runtime

**Introduction**

The parameters for initializing the IP interface are read out of the configuration memory during the boot process. The following settings can also be changed during the runtime of the JVM-407 via registers:

- IP Address
- Subnet mask
- IP address of default gateway

Settings made during runtime do not affect the parameters stored in the configuration memory, but will be lost when the JVM-407 is switched off.

**Prerequisites**

- These settings must only be made when there is no active communication via IP interface, otherwise data may be lost.
- It must be ensured that the values entered are valid (e.g. through proper programming within the application program), as the JVM-407 will not validate the values which are set during runtime.

**Overview of Registers**

| Registers | Description |
|---|---|
| **104531** | IP address of JVM-407 |
| **104532** | Subnet mask |
| **104533** | IP address of default gateway |

**Setting IP Addresses and Subnet Mask**

To set the IP address and the subnet mask 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 the JVM-407 into 104531. |
| **4** | Enter the desired subnet mask into 104532. |
| **5** | Enter the desired IP address of the default gateway into 104533. |

**Result:** The settings are made and communication is enabled.

**Related Topics**

- **Configuration Memory** on page 70

## Using Names for IP Address

**Introduction**

When specifying IP addresses of target systems (e.g. when configuring the e-mail client), names can be used as IP addresses. Then the JVM-407 translates these 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>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>During the boot process the JVM-407 reads the IP address of the DNS server from the configuration memory.</td></tr>
<tr><td>2</td><td>During the boot process the JVM-407 reads the file "/etc/hosts", creates a translation table with the names and IP addresses found in this file.</td></tr>
<tr><td>3</td><td>After the boot process the JVM-407 detects a name instead of an IP address.</td></tr>
<tr><td>4</td><td>Based on this translation table, the JVM-407 tries to resolve the name into a related IP address.
<table>
<tr><th>If ...</th><th>... Then ...</th></tr>
<tr><td>the name was resolved</td><td>the JVM-407 continues with step 6</td></tr>
<tr><td>the name could not be resolved</td><td>the JVM-407 continues with step 5</td></tr>
</table></td></tr>
<tr><td>5</td><td>The JVM-407 tries to resolve the name into a related IP address by sending a request to the DNS server.
<table>
<tr><th>If ...</th><th>... Then ...</th></tr>
<tr><td>the name was resolved</td><td>it enters the name and IP address into the translation table and proceeds with step 6</td></tr>
<tr><td>the name could not be resolved</td><td>the controller aborts the function (e.g. system function for sending an e-mail) with an error message</td></tr>
</table></td></tr>
<tr><td>6</td><td>The IP address found is used for further communication.</td></tr>
</table>

**Configuration File "hosts"**

A static assignment between name and IP address is specified in this file. This file is read once when the JVM-407 is booting.

| | |
|---|---|
| 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 resolve the IP address by obtaining the corresponding IP address from a DNS server. During the booting process of the JVM-407, the IP address of the DNS server is read from the configuration memory.

**Related Topics**


- **Configuration Memory** on page 70

# 5 Initial Commissioning

**Purpose of this Chapter**

This chapter covers the initial commissioning of the JVM-407 with the aid of the following steps:

- Creating IOP files in JetViewSoft for the JVM-407 device.
- Transferring the IOP files to the JVM-407 device.
- Creating an STX project in JetSym and configuring the hardware.
- Including the .iop.h file in the STX project.
- Including the ISO library in the STX project.
- Creating a compilable program.

Everything is then prepared as far as possible for creating a program.

**Minimum Requirements**

The instructions for initial commissioning apply to JetSym from version 4.3 and JetViewSoft from version 3.2.

**ISO Functions**

The ISO functions are defined in the ISO 11783-6 standard.

**Contents**

## Preparatory Work for Initial Commissioning

**Ethernet Connection to the Controller**

The HMI JVM-407 default IP address is 192.168.10.15. Configure the Ethernet interface of your PC so that it is able to communicate with the JVM-407 via this IP address.

**Requirement for Power-up**

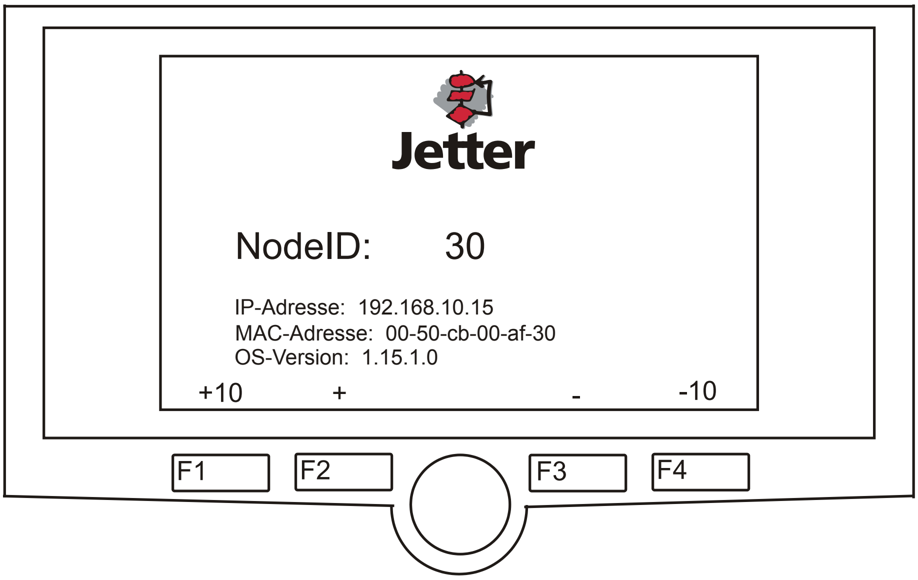
The JVM-407 only powers up if the supply voltage +U BATT is applied to the ignition (+).

**Behavior after Power-up**

If you press function keys F1 and F3 at the same time when powering up, the application program is not launched.

**Default Display**

The default application program launched on the JVM-407 after powering up displays the following input mask on the JVM-407 display.




The node ID displayed is the address of the CANopen® 0 bus set in the JVM-407. This address can be set by using the function keys F1 to F4.

The function key F1 increases the address in steps of 10. The function key F4 decreases the address in steps of 10.

The function key F2 increases the address in steps of 1. The function key F3 decreases the address in steps of 1.

The IP address, MAC address and OS version are also displayed.

# Initial Commissioning in JetViewSoft

**Introduction**

With JetViewSoft, the **IOP files** are created for the HMI JVM-407 and transferred to the device. The following is detailed here:

- Creating a project in JetViewSoft
- Executing Project Settings
- Creating an IOP file and transferring it to the HMI

The visualization created is programmed with JetSym STX.

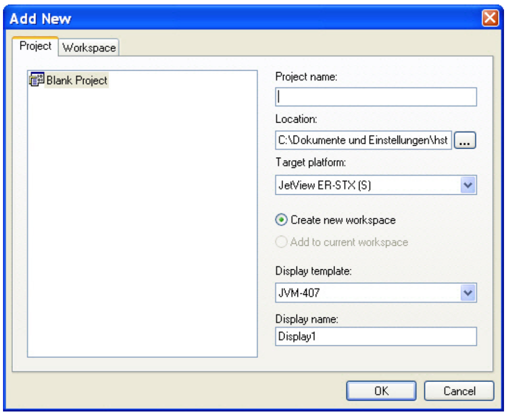
**Prerequisites**

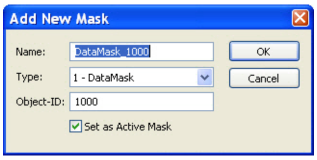
The following requirements must be satisfied:

- JetViewSoft is installed on the PC used
- JetViewSoft has been licensed (see online help in JetView Soft)
- An active Ethernet connection between the PC and the HMI is set up

**Creating a Project**

A new project for the HMI is created in JetViewSoft as follows:

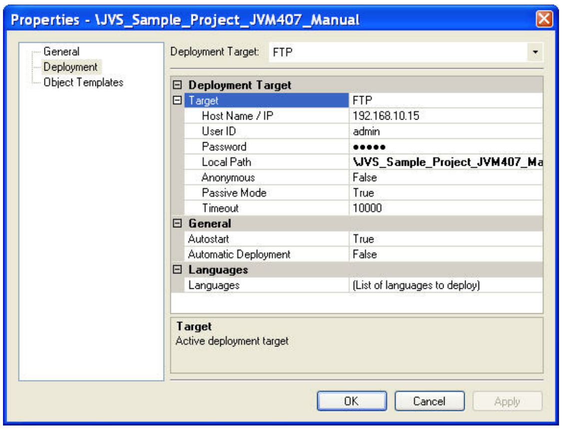
| Step | Action |
|---|---|
| 1 | Start JetViewSoft |
| 2 | Open the menu item **File** and select the entry **New**.<br>**Result:**<br>The following dialog box opens:<br> |
| 3 | In **Project name**, enter the name of the project. |
| 4 | If necessary, change the project menu path under **Location**. |
| 5 | Under **Target platform**, select JetView ER-STX(S). |
| 6 | Under **Display template**, select the appropriate one for the HMI. |
| 7 | Under **Display name**, select a program-internal name for the HMI.<br>Several displays can be created in one project. |

| Step | Action |
|---|---|
| **8** | Confirm your settings by clicking **OK**.<br>**Result:**<br>The dialog box closes and the **Add New Mask** dialog box opens.<br> |
| **9** | Under **Name**, enter the name of the first DataMask. You can retain the other settings. This mask is automatically the active mask when launching the HMI. |
| **10** | Confirm by clicking **OK**. |

**Result:** A project has now been created.

**Configuring Deployment**

In order to be able to transfer the IOP files created with JetViewSoft to the HMI, the required deployment settings need to be made:

| Step | Action |
|---|---|
| **1** | Open the menu item **Project** and select the entry **Properties**.<br>**Result**:<br>A dialog box with the same name opens. |
| **2** | Open the **Deployment** pane from the navigation panel on the left-hand side of the dialog box.<br> |
| **3** | Under **Deployment Target** (right at the top of the dialog box), select **FTP**. |
| **4** | Click on the + sign next to **Target** to expand the settings. |
| **5** | Under **Host Name/IP, enter the IP address for the HMI. The default IP address for a JVM-407 is 192.168.10.15.** |
| **6** | Confirm your settings by clicking **OK**. |

**Result:** The Deployment settings have now been made and the IOP files can now be transferred to the HMI.

**IOP Files**

The IOP files are created and transferred from a JetViewSoft project as follows:

| Step | Action |
|---|---|
| **1** | Open the menu item **File** and select the entry **Save all**. |
| **2** | Press the **F7** key for a project build.<br>**Result:**<br>The IOP files are created as long as no errors have occurred. |
| **3** | Open the menu item **Build** and select the entry **Deploy**.<br>**Result:**<br>The IOP files are transferred to the HMI as long as no errors have occurred. |
| **4** | Restart the HMI so that the IOP files can be imported |

**Result:** The IOP files are now displayed on the device.

## Initial Commissioning in JetSym

**Introduction**

The STX program for the visualization of the HMI JVM-407 is created with JetSym. The following is detailed here:

- Creating a project in JetSym
- Configuring the Hardware
- Including the JetViewSoft .iop.h file
- Including the ISO Library
- Creating a program that can be compiled and transferred to the HMI

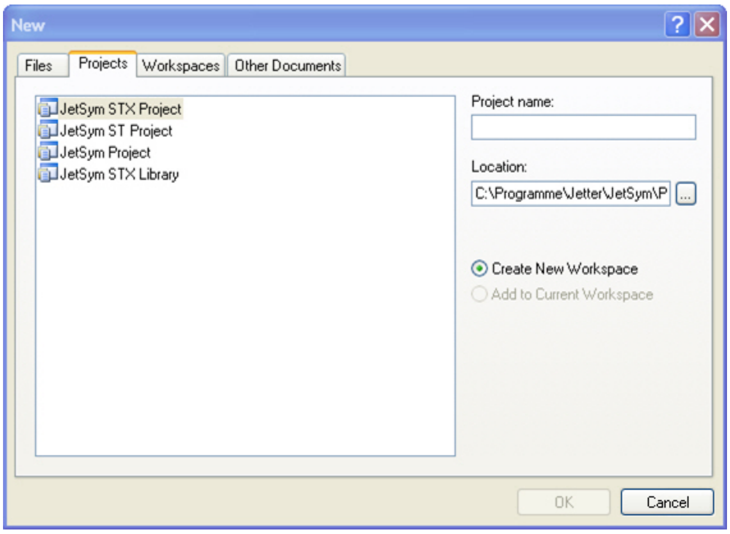
**Prerequisites**

The following requirements must be satisfied:

- JetSym is installed on the PC used.
- JetSym has been licensed (see online help in JetSym).
- An active Ethernet connection between the PC and the HMI is set up.
- Initial commissioning in JetViewSoft has been completed.

**Creating a Project**

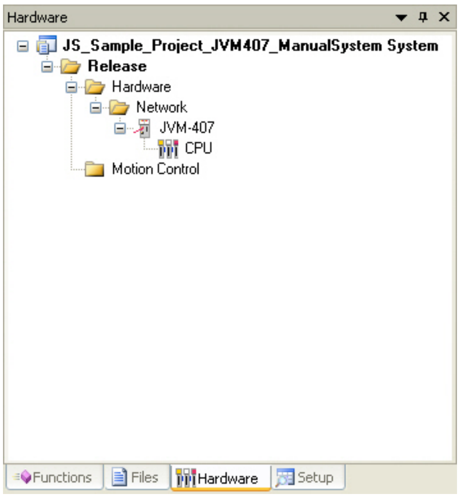
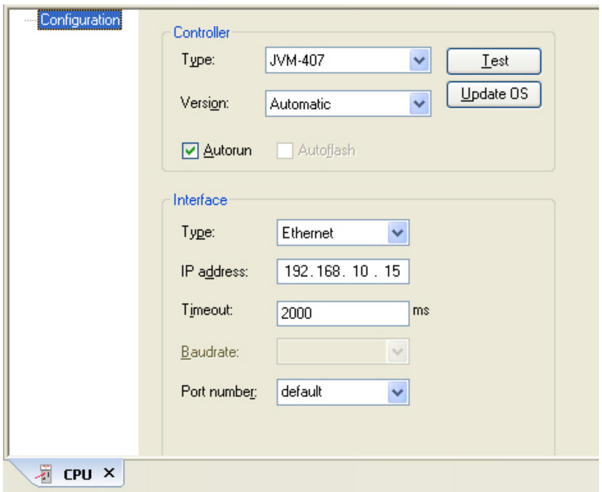
A new project for the programming is created in JetSym as follows:

| Step | Action |
|---|---|
| 1 | Start JetSym. |
| 2 | Open the menu item**File** and select the entry **New**.<br>**Result:**<br>The dialog box **New** opens<br><br> |
| 3 | Select **JetSym STX project** as the project type. |
| 4 | Enter the project name. |
| 5 | Confirm your settings by clicking **OK**. |

**Result:** A project has now been created.

**Configuring the Hardware**

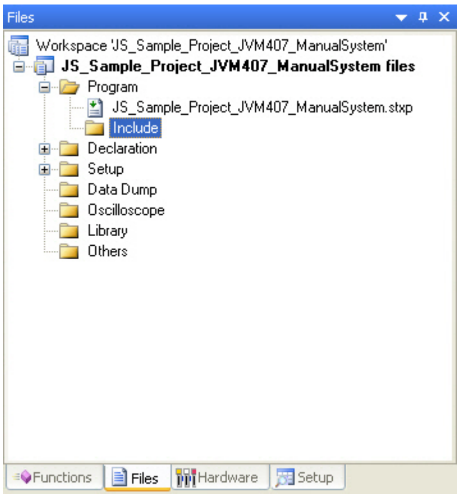
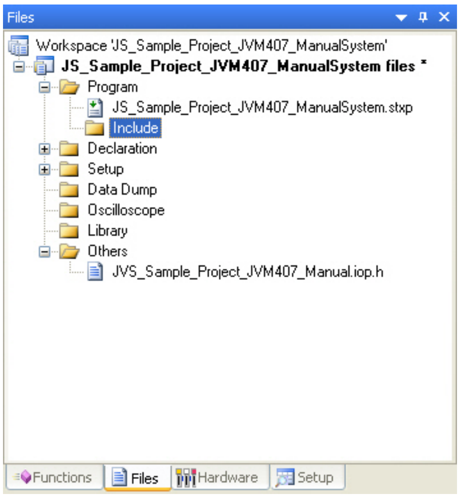
To establish a connection between JetSym and the HMI, you need to configure the hardware as follows:

| Step | Action |
|---|---|
| 1 | Switch to the **Hardware** view by clicking on the tab with the same name.<br><br> |
| 2 | Fully expand the **Hardware tree**. |
| 3 | Double-click on **CPU**, if the HMI JVM-407 is not set as the hardware.<br>**Result:**<br>The **Configuration** pane opens.<br><br> |
| 4 | Under **Controller/Type**, select JVM-407. |
| 5 | Under **Interface/IP address, enter the IP address for the HMI. The default IP address for a JVM-407 is 192.168.10.15.** |
| 6 | Test the connection by clicking on the **Test** button. If this is unsuccessful, check the IP address and the Ethernet connection for the JVM-407. |
| 7 | Save your settings using the shortcut **Ctrl + S**. |

**Result:** The hardware settings are now configured in JetSym.

**Header File .iop.h**

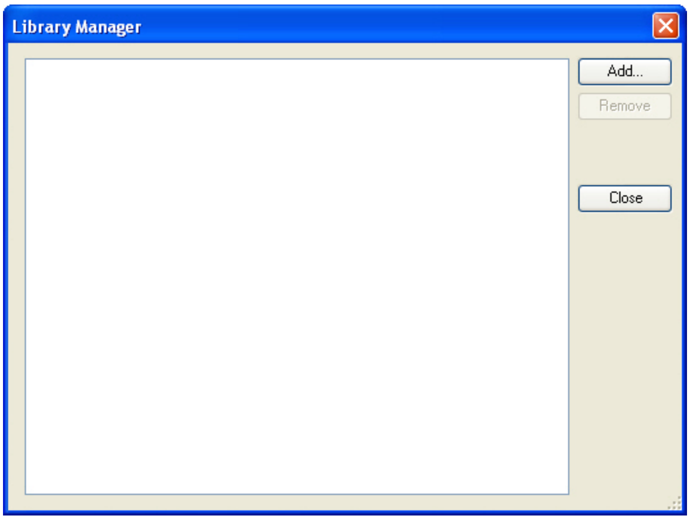
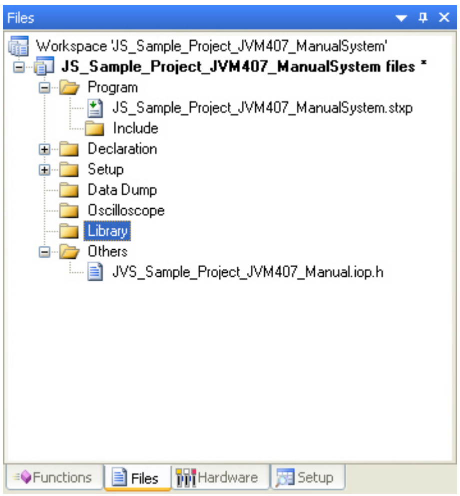
In order for the description of the ISO objects and masks for visualization to be available for programming, the .iop.h file must be included as follows:

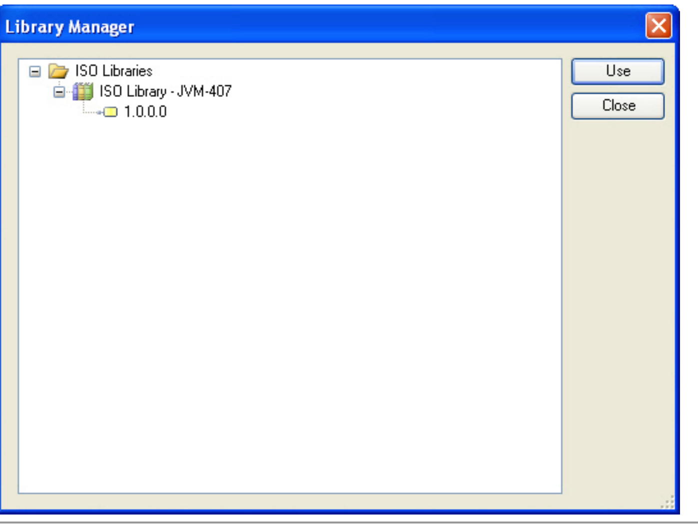
| Step | Action |
|---|---|
| **1** | Switch to **Files** view.<br><br> |
| **2** | Expand the **Program** folder. |
| **3** | Click on the **Include** folder and open the context menu (right-click with the mouse). |
| **4** | Select the context menu entry **Add Files to Directory**.<br>**Result:**<br>An Explorer window opens, which can be used to select a file. |
| **5** | Navigate to the **Output** folder for the JetViewSoft project. The default location for this is under **Own Files/JetViewSoft Projects/Name of JVS project/Output**. |
| **6** | Select file type **All Files (*.*)**. |
| **7** | Select the **.iop.h file**. |
| **8** | Click the **Open** button.<br><br> |

**Result:** The .iop.h file is now included in the JetSym project.

**Including the ISO library**

In order for the ISO library with the ISO functions to be available in JetSym, it must be included as follows:

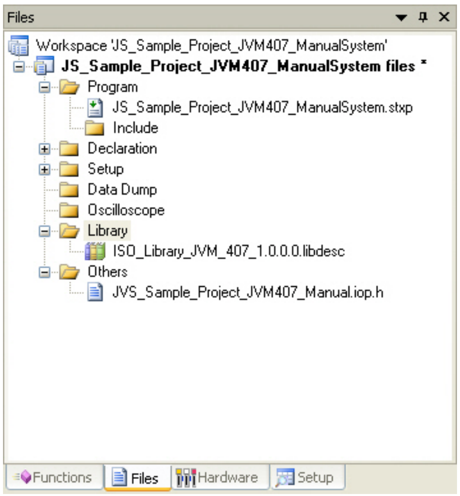
| Step | Action |
|---|---|
| **1** | Open the menu item **Tools** and select the entry **Library Manager**.<br>**Result:**<br>A dialog box with the same name opens.<br> |
| **2** | Click the **Add** button.<br>**Result:**<br>An Explorer window opens in the **Lib** folder of the JetSym installation. |
| **3** | Select **ISO_Library_1.0.0.0.libpackage**. |
| **4** | Click the **Open** button.<br>**Result:**<br>The libpackage file is included in Library Manager and can now be included in the JetSym project. |
| **5** | Switch to Files view.<br> |
| **6** | Select the **Library** folder and open the context menu by right-clicking with the mouse. |

| Step | Action |
|---|---|
| **7** | Select the option **Add Libraries**.<br>**Result:**<br>The **Library Manager** opens. |
| **8** | Select the libpackage file and click on the **Use** button.<br><br> |

**Result:** The file is now included in the project.

**Creating a Compilable Program.**

A compilable program is created and compiled as follows:

| Step | Action |
|---|---|
| **1** | Switch to Files view.<br> |
| **2** | Double-click on the program file (in this example **JS_Sample_Project_JVM407_Manual.stxp)**. The program file has the same name as the project, plus the extension **stxp**.<br>**Result:**<br>The program file opens in JetSym-Editor. |
| **3** | Enter the following program code. The .iop.h file has the same name as the project, plus the extension **iop.h**.<br>Please note this for the Include instruction.<br>`#Include "JVS_Sample_Project_JVM407_Manual.iop.h";`<br>`Task Main Autorun`<br>`End_Task;` |
| **4** | Press the **F7** key to trigger a project build.<br>**Result:**<br>The ISO functions and the IOP header file are now available for programming. |

**Result:**

The program can now be enhanced. In **IntelliSense** (**Ctrl + Space Bar**), the ISO functions and the information from the IOP header file are now available. You can use the shortcut **Ctrl+F5** to transfer the program to the HMI **.**
**However, it has no function as yet.**

**Related Topics:**

- **Initial Commissioning in JetViewSoft** on page 83

# 6 CANopen® STX API

**Introduction**

This chapter describes the STX functions of the CANopen® STX API.

**The CANopen® Standard**

CANopen® is an open standard for networking and communication in the automobile sector, for example.

The CANopen® protocol has been further developed by the CiA e.V. (CAN in Automation) and works on the physical layer with CAN Highspeed in accordance with ISO 11898.

**Application**

These STX functions are used in communication between the controller JVM-407 and e.g. the peripheral modules JXM-IO-E02, JXM-IO-E09, JXM-IO-E10, JXM-IO-E11 and JXM-MUX.

**Documentation**

The CANopen® specifications can be obtained from the **CiA e.V. http://www.can-cia.org** homepage. The key specification documents are:

- CiA DS 301 - This document is also known as the communication profile and describes the fundamental services and protocols used under CANopen®.
- CiA DS 302 - Framework for programmable devices (CANopen® Manager, SDO Manager)
- CiA DR 303 - Information on cables and connectors
- CiA DS 4xx - These documents describe the behavior of a number of device classes in, what are known as, device profiles.

**Contents**

## STX Function CanOpenInit

**Introduction**

Calling up the CanOpenInit () function initializes one of the CAN busses. The JVM-407 then automatically sends the heartbeat message every second with the following communication object identifier (COB-ID): Node ID + 0x700

**Function Declaration**

```
Function CanOpenInit (
    CANNo:Int,
    NodeID:Int,
    const ref SWVersion:String,
) :Int;
```

**Function Parameters**

The CanOpenInit () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 0 ... CANMAX |
| NodeID | Own Node ID | 1 ... 127 |
| SWVersion | Reference to own software version<br>This software version is entered into the index 0x100A in the object directory. | String up to 255 characters |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |
| -3 | Initialization has not worked |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | 0 |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Using this Function**

Initializing the CAN bus 0. The JVM-407 has node ID 20 (0x14).

```
Result := CanOpenInit(0, 20, 'Version: 01.00.0.00');
```

**How it Works**

During initialization, the JVM-407 processes the following process steps:

| Stage | Description |
|---|---|
| 1 | First, the bootup message is sent as a heartbeat message. |
| 2 | As soon as the JVM-407 goes into Pre-operational status, it sends the Pre-operational heartbeat message. |

**Access to the Object Directory**

The Object Directory can only be accessed via SDO, if the JVM-407 is in "Pre-operational" status.

**NMT Messages**

After initialization, NMT messages can be sent and received. The own heartbeat status can be changed with the "CanOpenSetCommand" function.

**Related Topics:**

- **STX Function CanOpenSetCommand** on page 96

## STX Function CanOpenSetCommand

**Introduction**

By calling up the CanOpenSetCommand () function, the own heartbeat status and the heartbeat status for all other devices (NMT slaves) can be changed on the CAN bus.

**Function Declaration**

```
Function CanOpenSetCommand (
    CANNo:Int,
    iType:Int,
    Value:Int,
) :Int;
```

**Function Parameters**

The CanOpenSetCommand () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 0 ... CANMAX |
| iType | Command selection | CAN_CMD_HEARTBEAT: Only the own heartbeat status is changed.<br>CAN_CMD_NMT: The heartbeat status is changed for all other devices or for a specific device on the CAN bus. |
| Value | Selection of the heartbeat status for command CAN_CMD_HEARTBEAT:<br>CAN_HEARTBEAT_STOPPED (0x04)<br>CAN_HEARTBEAT_OPERATIONAL (0x05)<br>CAN_HEARTBEAT_PREOPERATIONAL (0x7F)<br><br>Selection of the heartbeat status for command CAN_CMD_NMT (NMT master):<br>CAN_NMT_OPERATIONAL (0x01) or CAN_NMT_START (0x01)<br>CAN_NMT_STOP (0x02)<br>CAN_NMT_PREOPERATIONAL (0x80)<br>CAN_NMT_RESET (0x81)<br>CAN_NMT_RESETCOMMUNICATION (0x82) | |

**Note**

The command CAN_CMD_NMT is selected via the macro function CAN_CMD_NMT_Value (NodeID, CAN_CMD_NMT).

Values from 0 to 127 are permitted for the node ID parameter. 1 to 127 is the node ID for a specific device. If the command should be sent to all devices on the CAN bus, the parameter CAN_CMD_NMT_ALLNODES (0) is used.

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | 0 |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters<br>Command not known |

**Using the Function (Example 1)**

The own heartbeat status should be set to Operational.

```
Result := CanOpenSetCommand(0, CAN_CMD_HEARTBEAT,
CAN_HEARTBEAT_OPERATIONAL);
```

**Using the Function (Example 2)**

The own heartbeat status and the status of all other devices on the CAN bus should be set to Operational.

```
Result := CanOpenSetCommand(0,
CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES, CAN_CMD_NMT),
CAN_NMT_OPERATIONAL);
```

**Using the Function (Example 3)**

The heartbeat status of the device with the node ID 60 (0x3C) should be set to Operational.

```
Result := CanOpenSetCommand(0, CAN_CMD_NMT_Value(60, CAN_CMD_NMT),
CAN_NMT_OPERATIONAL);
```

## STX Function CanOpenUploadSDO

**Introduction**

Calling up the CanOpenUploadSDO () function is aimed at accessing a particular object in the Object Directory of the message recipient and the value of the object is read. Data is exchanged in accordance with the SDO upload protocol. Supported transfer types are "segmented" (more than 4 data bytes) and "expedited" (up to 4 data bytes).

**Function Declaration**

```
Function CanOpenUploadSDO (
    CANNo:Int,
    NodeID:Int,
    wIndex:Word,
    SubIndex:Byte,
    DataType:Int,
    DataLength:Int,
    const ref DataAddr,
    ref Busy: Int,
) :Int;
```

**Function Parameters**

The CanOpenUploadSDO () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 0 ... CANMAX |
| NodeID | Node ID of the message recipient | 1 ... 127 |
| wIndex | Index number of the object | 0 ... 0xFFFF |
| SubIndex | Sub-index number of the object | 0 ... 255 |
| DataType | Type of object to be received | 2 ... 27 |
| DataLength | Volume of data for the global variable DataAddr | |
| DataAddr | Global variable into which the received value is to be entered | |
| Busy | Status of the SDO transmission | |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |
| -2 | Controller in Stop status |
| -3 | DataType is greater than DataLength |
| -4 | insufficient memory |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | 0 |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter DataType**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Busy**

After calling up the function, the Busy parameter is set to SDOACCESS_INUSE. With an error in transmission, Busy is set to SDOACCESS_ERROR. With a successful transmission, the number of bytes transmitted is returned.

**"Busy" Error Codes**

With an error in transmission, Busy returns an error code. The following error codes are available:

**SDOACCESS_STILLUSED**

Another task is communicating with the same node ID.

**SDOACCESS_TIMEOUT**

The task has been timed out because the device with the given node ID is not responding.

If the specified device does not respond within 1 second, the timeout code is set

**SDOACCESS_ILLCMD**

The response to the request is invalid.

**SDOACCESS_ABORT**

The device with the node ID was aborted.

**SDOACCESS_SYSERROR**

General internal error

**Macro Definitions**

The following macros have been defined in connection with this function:

**SDOACCESS_FINISHED (busy)**

This macro checks whether communication has finished.

**SDOACCESS_ERROR (busy)**

This macro checks whether an error has occurred.

**Using this Function**

```
Result := CanOpenUploadSDO (
    0,
    66,
    0x100A,
    0,
    CANOPEN_STRING,
    sizeof(var_Versionstring),
    var_Versionstring,
    busy);
```

**JetSym STX Program**

In the following example, the manufacturer's software version is read from the CANopen® Object Directory of the device with the addressed node ID.

```
#Include "CanOpen.stxp"

Const
    // CAN no.
    CAN_CONTROLLER_0 = 0;
    // Node ID Node_1
    NodeID_Node_0 = 10;
    // Node ID node 2
    NodeID_Node_1 = 66;
End_Const;

Var
    busy: Int;
    Versionstring: String;
    Objectindex: Word;
    Subindex: Byte;
End_Var;


Task main autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

// Initialization CAN 0
CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// All devices on the CAN bus have the status of PREOPERATIONAL

// Request manufacturer's software version per SDO
Objectindex := 0x100A;
Subindex := 0;
CanOpenUploadSDO(CAN_CONTROLLER_0, NodeID_Node_1, Objectindex,
Subindex, CANOPEN_STRING, sizeof(Versionstring), Versionstring,
busy);

When SDOACCESS_FINISHED(busy) Continue;

If (SDOACCESS_ERROR(busy)) Then
// Troubleshooting

End_If;

//      ...
//      ...
```

```
//     ...

End_Task;
```

## STX Function CanOpenDownloadSDO

**Introduction**

Calling up the CanOpenDownloadSDO () function is aimed at accessing a particular object in the Object Directory of the message recipient and the value of the object is specified. Data is exchanged in accordance with the SDO download protocol. Supported transfer types are "segmented" or "block" (more than 4 data bytes) and "expedited" (up to 4 data bytes).

**Function Declaration**

```
Function CanOpenDownloadSDO (
    CANNo:Int,
    NodeID:Int,
    wIndex:Word,
    SubIndex:Byte,
    DataType:Int,
    DataLength:Int,
    const ref DataAddr,
    ref Busy: Int,
) :Int;
```

**Function Parameters**

The CanOpenDownloadSDO () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 0 ... CANMAX |
| NodeID | Node ID of the message recipient | 1 ... 127 |
| wIndex | Index number of the object | 0 ... 0xFFFF |
| SubIndex | Sub-index number of the object | 0 ... 255 |
| DataType | Type of object to be sent | 2 ... 27 |
| DataLength | Volume of data for the global variable DataAddr | |
| DataAddr | Global variable into which the sent value is to be entered | |
| Busy | Status of the SDO transmission | |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |
| -2 | HMI in Stop status (own heartbeat status) |
| -3 | DataType is greater than DataLength |
| -4 | insufficient memory |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | 0 |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter DataType**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Busy**

After calling up the function, the Busy parameter is set to SDOACCESS_INUSE. With an error in transmission, Busy is set to SDOACCESS_ERROR. With a successful transmission, the number of bytes transmitted is returned.

**"Busy" Error Codes**

With an error in transmission, Busy returns an error code. The following error codes are available:

**SDOACCESS_STILLUSED**
Another task is communicating with the same node ID.

**SDOACCESS_TIMEOUT**
The task has been timed out because the device with the node ID is not responding.
If the specified node ID does not respond within 1 second, the timeout code is set

**SDOACCESS_ILLCMD**
The response to the request is invalid.

**SDOACCESS_ABORT**
The device with the node ID was aborted.

**SDOACCESS_BLKSIZEINV**
Communication error with Block Download

**SDOACCESS_SYSERROR**
General internal error

**Macro Definitions**

The following macros have been defined in connection with this function:

**SDOACCESS_FINISHED (busy)**
This macro checks whether communication has finished.

**SDOACCESS_ERROR (busy)**
This macro checks whether an error has occurred.

**Using this Function**

```
Result := CanOpenDownloadSDO (
    0,
    68,
    0x1017,
    0,
    CANOPEN_WORD,
    sizeof(var_Heartbeat_time),
    var_Heartbeat_time,
    busy);
```

**JetSym STX Program**

In the following example, the heartbeat time is entered in the CANopen® Object Directory of the device with the addressed node ID.

```
#Include "CanOpen.stxp"

Const
    // CAN no.
    CAN_CONTROLLER_0 = 0;
    // Node ID Node_1
    NodeID_Node_0 = 10;
    // Node ID Node 2
    NodeID_Node_1 = 68;
End_Const;

Var
    busy: Int;
    Heartbeat_time: Int;
    Objectindex: Word;
    Subindex: Byte;
End_Var;


Task main autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

// Initialization CAN 0
CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// Set device with the node ID NodeID_Node_1 on the CAN bus to
PREOPERATIONAL status
CanOpenSetCommand(0, CAN_CMD_NMT_Value(NodeID_Node_1,
CAN_CMD_NMT), CAN_NMT_PREOPERATIONAL);

// Change heartbeat time of the addressed device per SDO
Objectindex := 0x1017;
Subindex := 0;
CanOpenDownloadSDO(CAN_CONTROLLER_0, NodeID_Node_1, Objectindex,
Subindex, CANOPEN_WORD, sizeof(Heartbeat_time), Heartbeat_time,
busy);

When SDOACCESS_FINISHED(busy) Continue;

If (SDOACCESS_ERROR(busy)) Then
// Troubleshooting

End_If;
```

```
// Reset all devices on the CAN bus to OPERATIONAL status
CanOpenSetCommand(CAN_CONTROLLER_0,
CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES, CAN_CMD_NMT),
CAN_NMT_OPERATIONAL);

//      ...
//      ...
//      ...

End_Task;
```

## STX Function CanOpenAddPDORx

**Introduction**

By calling up the CanOpenAddPDORx () function, process data, sent by other CANopen® devices, can be entered on receipt.

Process data are only received if sent by a CANopen® device.

**Notes**

- The PDO telegram is, however, only then transmitted if the CANopen® devices on the bus have a status of "Operational".
- The smallest time unit for the Event Time is 1 ms.
- The smallest time unit for the Inhibit Time is 1 ms.

**Function Declaration**

```
Function CanOpenAddPDORx (
    CANNo:Int,
    CANID:Int,
    BytePos:Int,
    DataType:Int,
    DataLength:Int,
    const ref VarAddr,
    EventTime: Int,
    InhibitTime: Int,
    Paramset: Int,
) :Int;
```

**Function Parameters**

The CanOpenAddPDORx () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 0 ... CANMAX |
| CANID | CAN identifier 11-bit<br>CAN identifier 29-bit | 0 ... 0x7FF<br>0 ... 0x1FFFFFFF |
| BytePos | Starting position of data to be received | 0 ... 7 |
| DataType | Data type of data to be received | 2 ... 13, 15 ... 27 |
| DataLength | Volume of data for the global variable VarAddr | |
| VarAddr | Global variable into which the received value is entered | |
| EventTime | Time lag between two telegrams (> Inhibit Time) | |
| InhibitTime | Minimum time lag between two telegrams received (< EventTime) | |
| Paramset | Parameter bit-coded | |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |
| -3 | DataType is greater than DataLength |
| -4 | insufficient memory |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | 0 |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter CANID**

The CANID parameter is used to transfer the CAN identifier. The CAN identifier is generated with a macro. The CAN identifier depends on the node ID of the other communicating user and on whether it is a PDO1, PDO2, PDO3 or PDO4 message.

**Macro definitions:**

```
#Define CANOPEN_PDO1_RX (NodeID)    ((NodeID) + 0x180)
#Define CANOPEN_PDO2_RX (NodeID)    ((NodeID) + 0x280)
#Define CANOPEN_PDO3_RX (NodeID)    ((NodeID) + 0x380)
#Define CANOPEN_PDO4_RX (NodeID)    ((NodeID) + 0x480)

#Define CANOPEN_PDO1_TX (NodeID)    ((NodeID) + 0x200)
#Define CANOPEN_PDO2_TX (NodeID)    ((NodeID) + 0x300)
#Define CANOPEN_PDO3_TX (NodeID)    ((NodeID) + 0x400)
#Define CANOPEN_PDO4_TX (NodeID)    ((NodeID) + 0x500)
```

**Example for calling up the macro:**

CANOPEN_PDO2_RX (64)

⇨ The resulting CAN identifier is: 2C0h = 40h + 280h

**Default CAN Identifier Distribution**

For CANopen® the following CAN identifier distribution is predefined. In this case, the node number is embedded in the identifier.

| 11-bit identifier (binary) | Identifier (decimal) | Identifier (hexadecimal | Function |
|---|---|---|---|
| 000000000000 | 0 | 0 | Network Management |
| 000100000000 | 128 | 80h | Synchronization |
| 0001xxxxxxxx | 129 - 255 | 81h - FFh | Emergency |
| 0011xxxxxxxx | 385 - 511 | 181h - 1FFh | PDO1 (tx) |
| 0100xxxxxxxx | 513 - 639 | 201h - 27Fh | PDO1 (rx) |
| 0101xxxxxxxx | 641 - 767 | 281h - 2FFh | PDO2 (tx) |
| 0110xxxxxxxx | 769 - 895 | 301h - 37Fh | PDO2 (rx) |
| 0111xxxxxxxx | 897 - 1023 | 381h - 3FFh | PDO3 (tx) |
| 1000xxxxxxxx | 1025 - 1151 | 401h -47Fh | PDO3 (rx) |
| 1001xxxxxxxx | 1153 - 1279 | 481h - 4FFh | PDO4 (tx) |
| 1010xxxxxxxx | 1281 - 1407 | 501h - 57Fh | PDO4 (rx) |
| 1011xxxxxxxx | 1409 - 1535 | 581h - 5FFh | Send SDO |
| 1100xxxxxxxx | 1537 - 1663 | 601h - 67Fh | Receive SDO |
| 1110xxxxxxxx | 1793 - 1919 | 701h - 77Fh | NMT Error Control |
| xxxxxxxx = Node number 1 - 127 | | | |

**Parameter DataType**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Parameter Paramset**

The following parameters can be transferred to the function. Several parameters can be linked together using the Or function.

**CANOPEN_ASYNCPDORTRONLY**

Receive asynchronous PDOs by sending an RTR frame (after expired EventTime) to the sender.

**CANOPEN_ASYNCPDO**

Receive asynchronous PDOs.

**CANOPEN_PDOINVALID**

PDO not received. Disk space is reserved.

**CANOPEN_NORTR**

PDO cannot be requested by RTR (Remote Request).

**CANOPEN_29BIT**

Use 29-bit identifier

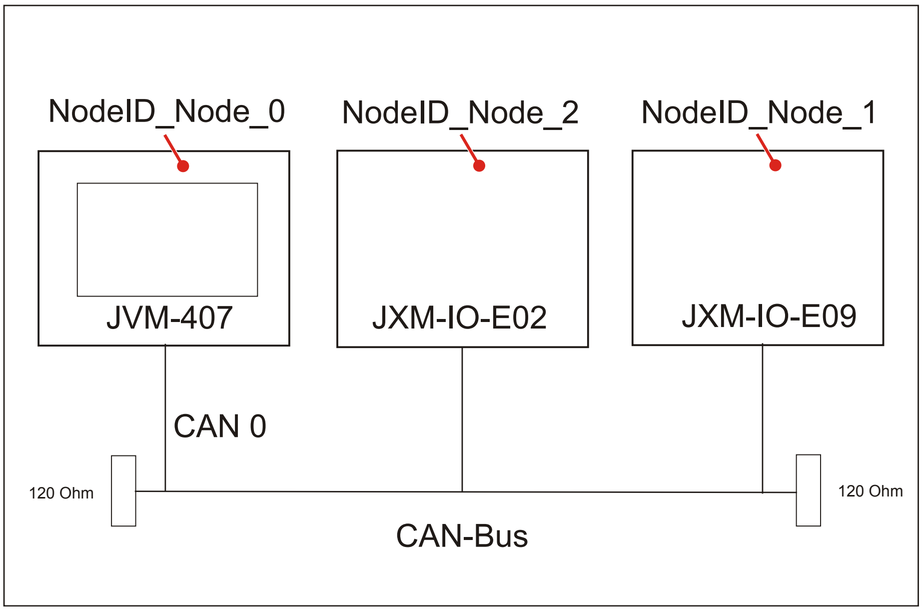
Default: 11-bit identifier

**Using this Function**

```
Result := CanOpenAddPDORx (
    0,
    662,
    0,
    CANOPEN_DWORD,
    sizeof(var_Data_1_of_Node_1),
    var_Data_1_of_Node_1,
    1000,
    10,
    CANOPEN_ASYNCPDO | CANOPEN_NORTR);
```

**JetSym STX Program**

JVM-407 with node ID 10 wants to receive a PDO from two CANopen® devices with node ID 64 and 102. The function CanOpenAddPDORx () is called up for this purpose. After running the program, the JVM-407 receives the cyclic PDO telegrams.




```
#Include "CanOpen.stxp"

Const
    // CAN no.
    CAN_CONTROLLER_0 = 0;
    // Node ID Node_1
    NodeID_Node_0 = 10;
    // Node ID Node 2
    NodeID_Node_1 = 64;
    // Node ID Node 3
    NodeID_Node_2 = 102;
    // Event_Time in ms
    Event_Time = 1000;
    // Inhibit time in ms
    Inhibit_Time = 10;
End_Const;

Var
    Data_1_of_Node_1: Int;
    Data_2_of_Node_1: Int;
    Data_1_of_Node_2: Int;
End_Var;


Task main autorun

Var
```

```
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

// Initialization CAN 0
CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// Enter process data on receipt
CanOpenAddPDORx(CAN_CONTROLLER_0,
CANOPEN_PDO2_RX(NodeID_Node_1), 0, CANOPEN_DWORD,
sizeof(Data_1_of_Node_1), Data_1_of_Node_1, Event_Time,
Inhibit_Time, CANOPEN_ASYNCPDORTRONLY | CANOPEN_NORTR);
CanOpenAddPDORx(CAN_CONTROLLER_0,
CANOPEN_PDO2_RX(NodeID_Node_1), 4, CANOPEN_DWORD,
sizeof(Data_2_of_Node_1), Data_2_of_Node_1, Event_Time,
Inhibit_Time, CANOPEN_ASYNCPDORTRONLY | CANOPEN_NORTR);
CanOpenAddPDORx(CAN_CONTROLLER_0,
CANOPEN_PDO3_RX(NodeID_Node_2), 0, CANOPEN_BYTE,
sizeof(Data_1_of_Node_2), Data_1_of_Node_2, Event_Time,
Inhibit_Time, CANOPEN_ASYNCPDO | CANOPEN_NORTR);

// All devices on the CAN bus have the status of PREOPERATIONAL
// Setting all devices on the CAN bus to OPERATIONAL status
CanOpenSetCommand(CAN_CONTROLLER_0,
CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES, CAN_CMD_NMT),
CAN_NMT_START);

//As from now, PDO telegrams will be transmitted.
//      ...
//      ...
//      ...

End_Task;
```

## STX Function CanOpenAddPDOTx

**Introduction**

By calling up the CanOpenAddPDOTx () function, process data can be deposited on the bus.

However, that should not mean that other CANopen® devices on the bus can also read this process data.

**Notes**

- The PDO telegram is, however, only then transmitted if the CANopen® devices on the bus have a status of "Operational".
- As soon as there are any changes to the process data, another PDO telegram is transmitted immediately.
- The smallest time unit for the Event Time is 1 ms.
- The smallest time unit for the Inhibit Time is 1 ms.
- Any unused bytes of a telegram are sent as null.

**Function Declaration**

```
Function CanOpenAddPDOTx (
    CANNo:Int,
    CANID:Int,
    BytePos:Int,
    DataType:Int,
    DataLength:Int,
    const ref VarAddr,
    EventTime: Int,
    InhibitTime: Int,
    Paramset: Int,
) :Int;
```

**Function Parameters**

The CanOpenAddPDOTx () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 0 ... CANMAX |
| CANID | CAN identifier 11-bit<br>CAN identifier 29-bit | 0 ... 0x7FF<br>0 ... 0x1FFFFFFF |
| BytePos | Starting position of data to be sent | 0 ... 7 |
| DataType | Data type of data to be sent | 2 ... 13, 15 ... 27 |
| DataLength | Volume of data for the global variable VarAddr | |
| VarAddr | Global variable into which the value to be sent is entered | |
| EventTime | Time lag between two telegrams (> Inhibit Time) | |
| InhibitTime | Minimum time lag between two telegrams sent (< EventTime) | |
| Paramset | Parameter bit-coded | |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |
| -3 | DataType is greater than DataLength |
| -4 | insufficient memory |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | 0 |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter CANID**

The CANID parameter is used to transfer the CAN identifier. The CAN identifier is generated with a macro. The CAN identifier depends on the node ID of the other communicating user and on whether it is a PDO1, PDO2, PDO3 or PDO4 message.

**Macro definitions:**

```
#Define CANOPEN_PDO1_RX (NodeID)    ((NodeID) + 0x180)
#Define CANOPEN_PDO2_RX (NodeID)    ((NodeID) + 0x280)
#Define CANOPEN_PDO3_RX (NodeID)    ((NodeID) + 0x380)
#Define CANOPEN_PDO4_RX (NodeID)    ((NodeID) + 0x480)

#Define CANOPEN_PDO1_TX (NodeID)    ((NodeID) + 0x200)
#Define CANOPEN_PDO2_TX (NodeID)    ((NodeID) + 0x300)
#Define CANOPEN_PDO3_TX (NodeID)    ((NodeID) + 0x400)
#Define CANOPEN_PDO4_TX (NodeID)    ((NodeID) + 0x500)
```

**Example for calling up the macro:**

CANOPEN_PDO2_RX (64)

⇨ The resulting CAN identifier is: 2C0h = 40h + 280h

**Default CAN Identifier Distribution**

For CANopen® the following CAN identifier distribution is predefined. In this case, the node number is embedded in the identifier.

| 11-bit identifier (binary) | Identifier (decimal) | Identifier (hexadecimal | Function |
|---|---|---|---|
| 000000000000 | 0 | 0 | Network Management |
| 000100000000 | 128 | 80h | Synchronization |
| 0001xxxxxxxx | 129 - 255 | 81h - FFh | Emergency |
| 0011xxxxxxxx | 385 - 511 | 181h - 1FFh | PDO1 (tx) |
| 0100xxxxxxxx | 513 - 639 | 201h - 27Fh | PDO1 (rx) |
| 0101xxxxxxxx | 641 - 767 | 281h - 2FFh | PDO2 (tx) |
| 0110xxxxxxxx | 769 - 895 | 301h - 37Fh | PDO2 (rx) |
| 0111xxxxxxxx | 897 - 1023 | 381h - 3FFh | PDO3 (tx) |
| 1000xxxxxxxx | 1025 - 1151 | 401h -47Fh | PDO3 (rx) |
| 1001xxxxxxxx | 1153 - 1279 | 481h - 4FFh | PDO4 (tx) |
| 1010xxxxxxxx | 1281 - 1407 | 501h - 57Fh | PDO4 (rx) |
| 1011xxxxxxxx | 1409 - 1535 | 581h - 5FFh | Send SDO |
| 1100xxxxxxxx | 1537 - 1663 | 601h - 67Fh | Receive SDO |
| 1110xxxxxxxx | 1793 - 1919 | 701h - 77Fh | NMT Error Control |
| xxxxxxxx = Node number 1 - 127 | | | |

**Parameter DataType**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Parameter Paramset**

The following parameters can be transferred to the function. Several parameters can be linked together using the Or function.

**CANOPEN_ASYNCPDORTRONLY**
Send asynchronous PDOs by receiving an RTR frame.

**CANOPEN_ASYNCPDO**
Send asynchronous PDO.

**CANOPEN_PDOINVALID**
PDO not sent.

**CANOPEN_NORTR**
PDO cannot be requested by RTR (Remote Request).

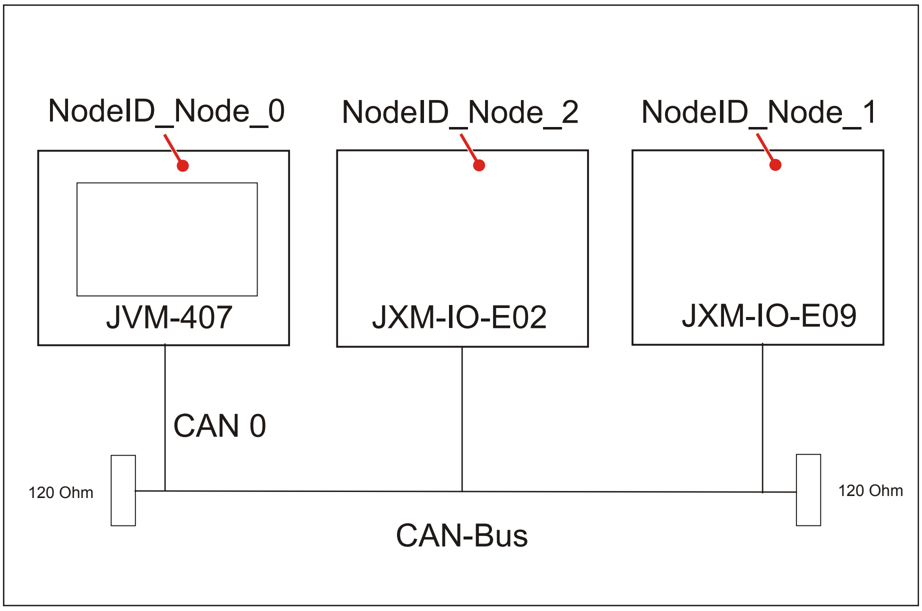
**CANOPEN_29BIT**
Use 29-bit identifier
Default: 11-bit identifier

**Using this Function**

```
Result := CanOpenAddPDOTx (
    0,
    842,
    0,
    CANOPEN_DWORD,
    sizeof(var_Data_1_of_Node_3),
    var_Data_1_of_Node_3,
    1000,
    100,
    CANOPEN_ASYNCPDO | CANOPEN_NORTR);
```

**JetSym STX Program**

JVM-407 sends process data to two CANopen® devices with the node ID 74 and 112. After running the program and for changes, the JVM-407 sends cyclic PDO telegrams every 3,000 ms (Event Time). As a maximum, the PDO telegram is sent every 10 ms (Inhibit Time).




```
#Include "CanOpen.stxp"

Const
    // CAN no.
    CAN_CONTROLLER_0 = 0;
    // Node ID Node_1
    NodeID_Node_0 = 10;
    // Node ID Node 4
    NodeID_Node_1 = 74;
    // Node ID Node 5
    NodeID_Node_2 = 112;
    // Event_Time in ms
    Event_Time = 3000;
    // Inhibit time in ms
    Inhibit_Time = 100;
End_Const;

Var
    Data_1_of_Node_1: Int;
    Data_2_of_Node_1: Int;
    Data_1_of_Node_2: Byte;
End_Var;


Task main autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';
```

```
// Initialization CAN 0
CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// Send data per PDO
CanOpenAddPDOTx(CAN_CONTROLLER_0,
CANOPEN_PDO2_TX(NodeID_Node_1), 0, CANOPEN_DWORD,
sizeof(Data_1_of_Node_1), Data_1_of_Node_1 Event_Time,
Inhibit_Time, CANOPEN_ASYNCPDORTRONLY | CANOPEN_NORTR);
CanOpenAddPDOTx(CAN_CONTROLLER_0,
CANOPEN_PDO2_TX(NodeID_Node_1), 4, CANOPEN_DWORD,
sizeof(Data_2_of_Node_1), Data_2_of_Node_1, Event_Time,
Inhibit_Time, CANOPEN_ASYNCPDORTRONLY | CANOPEN_NORTR);
CanOpenAddPDOTx(CAN_CONTROLLER_0,
CANOPEN_PDO3_TX(NodeID_Node_2), 0, CANOPEN_BYTE,
sizeof(Data_1_of_Node_2), Data_1_of_Node_2, Event_Time,
Inhibit_Time, CANOPEN_ASYNCPDO | CANOPEN_NORTR);

// All devices on the CAN bus have the status of PREOPERATIONAL
// Set all devices on the CAN bus to OPERATIONAL status
CanOpenSetCommand(CAN_CONTROLLER_0,
CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES, CAN_CMD_NMT),
CAN_NMT_START);

//As from now, PDO telegrams will be transmitted.
//      ...
//      ...
//      ...

End_Task;
```

## CANopen® Object Directory for JVM-407

**Supported Objects**

The following objects are supported by the operating system for JVM-407:

| Index (hex) | Object (code) | Object name | Type | Attribute |
|---|---|---|---|---|
| 1000 | VAR | Device Type | Unsigned32 | ro |
| 1001 | VAR | Error Register | Unsigned8 | ro |
| 1002 | VAR | Manufacturer Status | Unsigned32 | ro |
| 1003 | ARRAY | Pre-defined Error Field | Unsigned32 | ro |
| 1008 | VAR | Manufacturer Device Name | String | const |
| 1009 | VAR | Manufacturer Hardware Version | String | const |
| 100A | VAR | Manufacturer Software Version | String | const |
| 100B | VAR | Node ID | Unsigned32 | ro |
| 1017 | VAR | Producer Heartbeat Time | Unsigned16 | rw |
| 1018 | RECORD | Identity | Identity | ro |
| 1200 | RECORD | Server 1 - SDO Parameter | SDO Parameter | ro |
| 1201 | RECORD | Server 2 - SDO Parameter | SDO Parameter | rw |
| 1203 | RECORD | Server 3 - SDO Parameter | SDO Parameter | rw |
| 1203 | RECORD | Server 4 - SDO Parameter | SDO Parameter | rw |

**Device Type Object (Index 0x1000)**

The structure of the "Device Type Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x1000 | 0 | 0x0000012D | Device Type (Read-Only) |

**Error Register Object (Index 0x1001)**

The structure of the "Error Register Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x1001 | 0 | 0 | Error Register (Read-Only) |

This object implements the CANopen® Error Register functionality.

Bit 0 – Generic Errors

None of the other bits are currently in use.

**Pre-defined Error Field Object (Index 0x1003)**

The structure of the "Pre-defined Error Field Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x1003 | 0 | 0 | Number of errors entered in the Array's Standard Error Field |
| | 1 | 0 | Most recent error<br>0 indicates no error |
| | 2 ... 254 | - | Earlier Errors |

This object shows a history list of errors that have been detected by the JVM-407. The maximum length of the list is 254 errors. The list content is deleted on restart.

**Composition of the Standard Error Field**

2-byte LSB: Error Code

2-byte MSB: Additional information

**Manufacturer Device Name Object (Index 0x1008)**

The structure of the "Manufacturer Device Name Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x1008 | 0 | JVM-407 | Hardware name |

**Manufacturer Hardware Version Object (Index 0x1009)**

The structure of the "Manufacturer Hardware Version Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x1009 | 0 | | OS version of the device |

**Manufacturer Software Version Object (Index 0x100A)**

The structure of the "Manufacturer Software Version Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x100A | 0 | | Software version of the application program that runs on the JVM-407 |

The entry in this index is made via the parameter "SWVersion" of the STX function CanOpenInit ().

**Node ID Object (Index 0x100B)**

The structure of the "Node ID Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x100B | 0 | | Own Node ID |

**Producer Heartbeat Time Object (Index 0x1017)**

The structure of the "Producer Heartbeat Time Object" is shown in the following table.

| Index | Sub-Index | Default | Description |
|---|---|---|---|
| 0x1017 | 0 | 1,000 [ms] | Heartbeat time |

# 7 SAE J1939 STX API

**Introduction**

This chapter describes the STX functions of the SAE J1939 STX API.

**The SAE J1939 Standard**

SAE J1939 is an open standard for networking and communication in the commercial vehicle sector. The focal point of the application is the networking of the power train and chassis. The J1939 protocol originates from the international Society of Automotive Engineers (SAE) and works on the physical layer with CAN high-speed according to ISO 11898.

**Application**

These STX functions are used in communication between the controller JVM-407 and other ECUs in the vehicle. As a rule, engine data e.g. rpm, speed or coolant temperature are read and displayed.

**Documentation**

The key SAE J1939 specifications are:

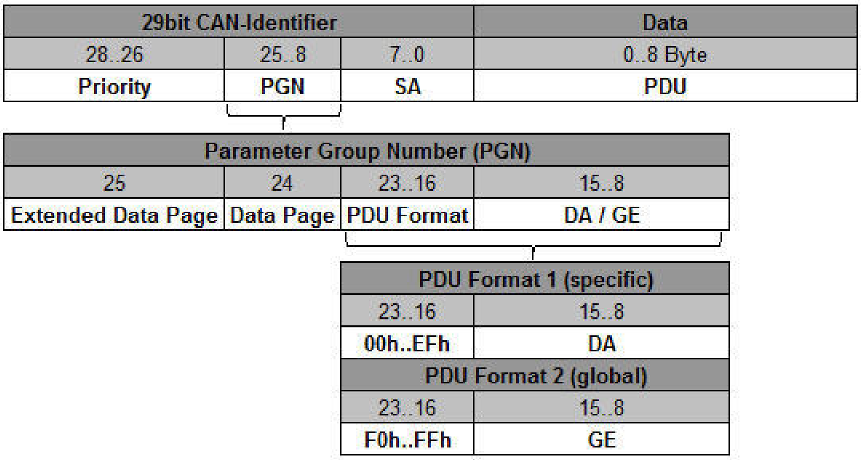
- J1939-11 - Information on the physical layer
- J1939-21 - Information on the data link layer
- J1939-71 - Information on the application layer vehicles
- J1939-73 - Information on the application layer range analysis
- J1939-81 - Network management

**Contents**

## Content of a J1939 Message

**Content of a J1939 Message**

The following diagram shows the content of a J1939 message:




| Abbreviation | Description |
|---|---|
| DA | Destination Address |
| GE | Group Extensions |
| PDU | Protocol Data Unit |
| PGN | Parameter Group Number |
| SA | Source Address |

**Meaning of the Parameter Group Number (PGN)**

The PGN is a number defined in the SAE J1939 standard that groups together several SPNs into a meaningful group. The PGN is part of the CAN identifier. The 8-byte data (PDU) contain the values of individual SPNs.

The example below shows a PGN 65262 (0xFEEE):

**PGN 65262 Engine Temperature 1 - ET1**

| Part of the PGN | Value | Remarks |
|---|---|---|
| Transmission Repetition Rate | 1 s | |
| Data Length | 8 | |
| Extended Data Page | 0 | |
| Data Page | 0 | |
| PDU Format | 254 | |
| PDU Specific | 238 | PGN Supporting Information |
| Default Priority | 6 | |
| Parameter Group Number | 65262 | in hex: 0xFEEE |

| Start position | Length | Parameter name | SPN |
|---|---|---|---|
| 1 | 1 byte | Engine Coolant Temperature | 110 |
| 2 | 1 byte | Engine Fuel Temperature 1 | 174 |
| 3 - 4 | 2 bytes | Engine Oil Temperature 1 | 175 |
| 5 - 6 | 2 bytes | Engine Turbocharger Oil Temperature | 176 |
| 7 | 1 byte | Engine Intercooler Temperature | 52 |
| 8 | 1 byte | Engine Intercooler Thermostat Opening | 1134 |

## STX Function SAEJ1939Init

**Introduction**

Calling up the SAEJ1939Init () function initializes one of the CAN busses (not CAN 0 as this is reserved for CANopen®) available for the J1939 protocol. From then on, the JVM-407 has the SA (Source Address) assigned by the function parameter mySA. It thus has its own device address on the bus.

**Function Declaration**

```
Function SAEJ1939Init (
    CANNo:Int,
    mySA:Byte,
) :Int;
```

**Function Parameters**

The function SAEJ1939Init () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| mySA | Own source address | 0 ... 253 |

**Return Value**

This function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | OK |
| -1 | Error when checking parameters |
| -3 | Insufficient memory for SAE J1939 |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Using this Function**

Initializing the CAN-Bus 1. The JVM-407 has Node-SA 20 (0x14).

The JVM-407 can now send messages with the set SA (and only these messages).

```
Result := SAEJ1939Init(1, 20);
```

| | |
|---|---|
| **Address Claiming** | Address Claiming has not been implemented. |

## STX Function SAEJ1939SetSA

**Introduction**

Calling up the function SAEJ1939SetSA changes the own SA (Source Address) during runtime.

**Function Declaration**

```
Function SAEJ1939SetSA (
    CANNo:Int,
    mySA:Byte,
) :Int;
```

**Function Parameters**

The function SAEJ1939SetSA () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| mySA | New SA | 0 ... 253 |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Using this Function**

The SA is changed during runtime.

```
Result := SAEJ1939SetSA(1, 20);
```

**Important Note**

Messages are immediately sent/received with the new SA.

## STX Function SAEJ1939GetSA

**Introduction**

By calling up the function SAEJ1939GetSA, you can determine the own SA (Source Address).

**Function Declaration**

```
Function SAEJ1939GetSA (
    CANNo:Int,
    ref mySA:Byte,
) :Int;
```

**Function Parameters**

The function SAEJ1939GetSA () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| mySA | SA currently set | 0 ... 253 |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Using this Function**

This function returns the currently set SA.

```
Result := SAEJ1939SetSA(1, actual_SA);
```

## STX Function SAEJ1939AddRx

**Introduction**

Calling up the function SAEJ1939AddRx () prompts the JVM-407 to receive a specific message. This message is sent from another bus node. The address of this bus node is transferred to this function as a bySA parameter. If the message is not sent, the value received last remains valid. Cyclical reading continues until the function SAEJ1939Init () is called up again.

**Function Declaration**

```
Function SAEJ1939AddRx (
    CANNo:Int,
    IPGN:Long,
    bySA:Byte,
    BytePos:Int,
    BitPos:Int,
    DataType:Int,
    DataLength:Int,
    const ref VarAddr,
    ref stJ1939:TJ1939Rx
    EventTime: Int,
    InhibitTime: Int,
) :Int;
```

**Function Parameters**

The function SAEJ1939AddRx () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| IPGN | PGN<br>Parameter Group Number | 0 ... 0x3FFFF |
| bySA | Source Address of message sender | 0 ... 253 |
| BytePos | Starting position of bytes of data to be received | 1 ... n |
| BitPos | Starting position of bits of data to be received | 1 ... 8 |
| DataType | Data type of data to be received | 1 ... 3, 10 ... 16 |
| DataLength | Volume of data for the global variable VarAddr | |
| VarAddr | Global variable into which the received value is entered | |
| TJ1939Rx | Control structure | |
| EventTime | Time lag between two telegrams (> Inhibit Time) | Default Value: 1,000 ms |
| InhibitTime | Minimum time lag between two telegrams received (< EventTime) | Default Value: 100 ms |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter DataType**

Data types can include the following.

| Byte types | Bit types | SAEJ1939 |
|---|---|---|
| 1 | - | SAEJ1939_UNSIGNED8<br>SAEJ1939_BYTE |
| 2 | - | SAEJ1939_UNSIGNED16<br>SAEJ1939_WORD |
| 4 | - | SAEJ1939_UNSIGNED32<br>SAEJ1939_DWORD |
| n | - | SAEJ1939_STRING |
| - | 1 | SAEJ1939_1BIT |
| - | 2 | SAEJ1939_2BIT |
| - | 3 | SAEJ1939_3BIT |
| - | 4 | SAEJ1939_4BIT |
| - | 5 | SAEJ1939_5BIT |
| - | 6 | SAEJ1939_6BIT |
| - | 7 | SAEJ1939_7BIT |

**Control Structure TJ1939Rx**

```
TJ1939Rx: Struct
// Status of received message
            byStatus       : Byte;
// Priority of received message
            byPriority     : Byte;
           End_Struct;
```

**Using this Function**

```
Result := SAEJ1939AddRx (
    1,
    0xFEEE,
    0x00,
    2
    0
    SAEJ1939_BYTE,
    sizeof(var_Fueltemp),
    var_Fueltemp,
    struct_TJ1939Rx_EngineTemperatureTbl,
    1500,
    120);
```

**JetSym STX Program**

The device JVM-407 with the own SA of 20 wants to receive and display the current fuel temperature. The parameters InhibitTime and EventTime are not explicitly specified when calling up the function. In this case, the default values are used. The controller that measures the fuel temperature has the SA of 0. In practice, the address of the controller can be found in the engine manufacturer's documentation.

The fuel temperature has the SPN 174 and is a component (byte 2) of the PGN 65262 Engine Temperature 1.

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;

// PGN 65262 Engine Temperature 1
    Fueltemp  : Byte;
    EngineTemperatureTbl : TJ1939Rx;
End_Var;

Task main autorun

// Initializing CAN 1
bySAEJ1939Channel := 1;
own_Source_Address := 20;
SAEJ1939Init (bySAEJ1939Channel, own_Source_Address);

// Receive fuel temperature
SAEJ1939AddRx (bySAEJ1939Channel, 65262, 0x00, 2, 1, SAEJ1939_BYTE,
sizeof(Fueltemp), Fueltemp, EngineTemperatureTbl);

End_Task;
```

| | |
|---|---|
| **Engine Manufacturer's Manual** | For information on the data (priority, PGN, SA and data byte structure) refer to the manual provided by the engine manufacturer. |

## STX Function SAEJ1939AddTx

**Introduction**

Calling up the function SAEJ1939AddTx () prompts the device JVM-407 to cyclically send a specific message via the bus.

Cyclical sending continues until the function SAEJ1939Init () is called up again.

Date are sent once the Event Time has elapsed or the given variables have changed and Inhibit Time has elapsed.

**Function Declaration**

```
Function SAEJ1939AddTx (
    CANNo:Int,
    IPGN:Long,
    BytePos:Int,
    BitPos:Int,
    dataType:Int,
    DataLength:Int,
    const ref VarAddr,
    ref stJ1939:TJ1939Tx
    EventTime: Int,
    InhibitTime: Int,
) :Int;
```

**Function Parameters**

The function SAEJ1939AddTx () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| IPGN | PGN<br>Parameter Group Number | 0 ... 0x3FFFF |
| BytePos | Starting position of the byte of data to be sent | 1 ... n |
| BitPos | Starting position of the bit of data to be sent | 1 ... 8 |
| DataType | Data type of data to be sent | 1 ... 3, 10 ... 16 |
| DataLength | Volume of data for the global variable VarAddr | |
| VarAddr | Global variable into which the value to be sent is entered | |
| TJ1939Tx | Control structure | |
| EventTime | Time lag between two telegrams (> Inhibit Time) | Default Value: 1,000 ms |
| InhibitTime | Minimum time lag between two telegrams received (< EventTime) | Default Value: 100 ms |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter DataType**

Data types can include the following.

| Byte types | Bit types | SAEJ1939 |
|---|---|---|
| 1 | - | SAEJ1939_UNSIGNED8<br>SAEJ1939_BYTE |
| 2 | - | SAEJ1939_UNSIGNED16<br>SAEJ1939_WORD |
| 4 | - | SAEJ1939_UNSIGNED32<br>SAEJ1939_DWORD |
| n | - | SAEJ1939_STRING |
| - | 1 | SAEJ1939_1BIT |
| - | 2 | SAEJ1939_2BIT |
| - | 3 | SAEJ1939_3BIT |
| - | 4 | SAEJ1939_4BIT |
| - | 5 | SAEJ1939_5BIT |
| - | 6 | SAEJ1939_6BIT |
| - | 7 | SAEJ1939_7BIT |

**Control Structure TJ1939Tx**

```
TJ1939Tx : Struct
// Status of sent message
            byStatus      : Byte;
// Priority of sent message
            byPriority    : Byte;
          End_Struct;
```

**Using this Function**

```
Result := SAEJ1939AddTx (
    1,
    0xFEEE,
    0x00,
    2
    0
    SAEJ1939_BYTE,
    sizeof(var_Fueltemp),
    var_Fueltemp,
    struct_TJ1939Tx_EngineTemperatureTbl,
    1500,
    120);
```

**JetSym STX Program**

Redefining the priority: Priority value 0 has the highest priority, priority value 7 has the lowest priority. A message with priority 6 can be superseded by a message with priority 4 (if the messages are sent at the same time). The parameters InhibitTime and EventTime are not explicitly specified when calling up the function. In this case, the default values are used.

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;

// PGN 65262 Engine Temperature 1
    Fueltemp  : Byte;
    EngineTemperatureTbl : TJ1939Tx;
End_Var;

Task main autorun

// Initializing CAN 1
bySAEJ1939Channel := 1;
own_Source_Address := 20;
SAEJ1939Init (bySAEJ1939Channel, own_Source_Address);

// PGN 65262 Engine Temperature
// Set a new priority
EngineTemperatureTbl.byPriority := 6;
SAEJ1939AddTx (bySAEJ1939Channel, 65262, 0x00, 2, 1, SAEJ1939_BYTE,
sizeof(Fueltemp), Fueltemp, EngineTemperatureTbl);

End_Task;
```

| | |
|---|---|
| **Engine Manufacturer's Manual** | For information on the data (priority, PGN, SA and data byte structure) refer to the manual provided by the engine manufacturer. |

## STX Function SAEJ1939RequestPGN

**Introduction**

Calling up the function SAEJ1939RequestPGN () sends a request to the DA (Destination Address) following a PGN.

This function is not terminated until a valid value has been received or the timeout of 1,250 ms has elapsed.

To obtain the value of the requested message its receipt must be scheduled using the function SAEJ1939AddRx ().

This function must be constantly recalled in cycles.

**Function Declaration**

```
Function SAEJ1939RequestPGN (
    CANNo:Int,
    byDA:Byte,
    ulPGN:Long,
    byPriority:Byte,
) :Int;
```

**Function Parameters**

The function SAEJ1939RequestPGN () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| byDA | Destination Address<br>Address from which the message is requested | 0 ... 253<br>The own SA cannot be used |
| ulPGN | PGN<br>Parameter Group Number | 0 ... 0x3FFFF |
| byPriority | Priority | 0 ... 7<br>Default Value: 6 |

**Return Value**

This function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | Message has been received |
| -1 | Timeout, as no reply has been received |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Parameter DataType**

Data types can include the following.

| Byte types | Bit types | SAEJ1939 |
|---|---|---|
| 1 | - | SAEJ1939_UNSIGNED8<br>SAEJ1939_BYTE |
| 2 | - | SAEJ1939_UNSIGNED16<br>SAEJ1939_WORD |
| 4 | - | SAEJ1939_UNSIGNED32<br>SAEJ1939_DWORD |
| n | - | SAEJ1939_STRING |
| - | 1 | SAEJ1939_1BIT |
| - | 2 | SAEJ1939_2BIT |
| - | 3 | SAEJ1939_3BIT |
| - | 4 | SAEJ1939_4BIT |
| - | 5 | SAEJ1939_5BIT |
| - | 6 | SAEJ1939_6BIT |
| - | 7 | SAEJ1939_7BIT |

**Using this Function**

```
Result := SAEJ1939RequestPGN (
    1,
    0x00,
    0xFEE5,
    5);
```

**JetSym STX Program**

JVM-407 with own SA of 20 wants to request the PGN 65253 "Engine Hours" from an engine control unit with the SA 0. The SPN 247 "Engine Total Hours of Operation" should be read from this PGN. It is therefore necessary to register receipt of the SPN 247 by calling up the function SAEJ1939AddRx ().

The parameter "byPriority" is not explicitly specified when calling up the function. In this case, the default value is used.

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;

// PGN 65253 Engine Hours, Revolutions
    EngineTotalHours  : Int;
    EngineHoursTbl : TJ1939Rx;
End_Var;

Task main autorun

// Initializing CAN 1
```

```
bySAEJ1939Channel := 1;
own_Source_Address := 20;
SAEJ1939Init (bySAEJ1939Channel, own_Source_Address);

// Engine Hours, Revolutions  -- on Request
SAEJ1939AddRx (bySAEJ1939Channel, 65253, 0x00, 1, 0,
SAEJ1939_DWORD, sizeof(EngineTotalHours), EngineTotalHours,
EngineHoursTbl, 5000, 150);

// Required for a cyclical task
TaskAllEnableCycle ();
EnableEvents;

End_Task;


Task t_RequestPGN_5000 cycle 5000

Var
    Return_value : Int;
End_Var;

// Request total machine operating hours
Return_value := SAEJ1939RequestPGN (bySAEJ1939Channel, 0x00,
65253);

If Return_value Then
      Trace ('PGN Request failed');
End_If;

End_Task;
```

## STX Function SAEJ1939GetDM1

**Introduction**

Calling up the function SAEJ1939GetDM1 () requests the current diagnostics error codes (also see SAE J1939-73 No. 5.7.1). The corresponding PGN number is 65226. This function must be constantly recalled in cycles.

**Function Declaration**

```
Function SAEJ1939GetDM1 (
    CANNo:Int,
    bySA:Byte,
    ref stJ1939DM1stat:TJ1939DM1STAT
    ref stJ1939DM1msg:TJ1939DM1MSG
) :Int;
```

**Function Parameters**

The function SAEJ1939GetDM1 () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| bySA | Source Address of message sender | 0 ... 253<br>The own SA cannot be used |
| stJ1939DM1stat | lStatus<br>lMsgCnt<br>lBuffer | Lamp Status<br>Number of received messages<br>Size of variable stJ1939DM1msg |
| stJ1939DM1msg | lSPN<br>byOC<br>byFMI | Error Code<br>Error counter<br>Error Type |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |

| Device | CANMAX |
|---|---|
| JCM-620 | 2 |

**stJ1939DM1stat.lStatus**

**Default**: 0xFF00

| Type | Byte | Bit group | Description |
|---|---|---|---|
| Status | 1 | 8 - 7 | Malfunction Indicator Lamp Status |
| | | 6 - 5 | Red Stop Lamp Status |
| | | 4 - 3 | Amber Warning Lamp Status |
| | | 2 - 1 | Protect Lamp Status |
| Flash | 2 | 8 - 7 | Flash Malfunction Indicator Lamp |
| | | 6 - 5 | Flash Red Stop Lamp |
| | | 4 - 3 | Flash Amber Warning Lamp |
| | | 2 - 1 | Flash Protect Lamp |

| Type | Byte | Bit group Value | Description |
|---|---|---|---|
| Status | 1 | 00 | Lamps off |
| | | 01 | Lamps on |
| Flash | 2 | 00 | Slow Flash (1 Hz, 50 % duty cycle) |
| | | 01 | Fast Flash (2 Hz or faster, 50 % duty cycle) |
| | | 10 | Reserved |
| | | 11 | Unavailable / Do Not Flash |

**stJ1939DM1msg**

**Default Value:**
lSPN = 0
byOC = 0
byFMI = 0
For older controllers (grandfathered setting):
lSPN = 524287 (0x7FFFF)
byOC = 31 (0x1F)
byFMI = 127 (0x7F)

**Using this Function**

```
Result := SAEJ1939GetDM1 (
    1,
    0x00,
    stdm1stat_pow,
    stdm1msg_pow,);
```

**JetSym STX Program**

By calling up the function SAEJ1939GetDM1 (), the JVM-407 requests the current diagnostics error code (PGN 65226).

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;
    stdm1stat_pow : TJ1939DM1STAT;
    stdm1msg_pow : Array[10] of STJ1939DM1MSG;
    MyTimer : TTimer;
End_Var;

Task main autorun

// Initializing CAN 1
bySAEJ1939Channel := 1;
own_Source_Address := 20;
SAEJ1939Init (bySAEJ1939Channel, own_Source_Address);

TimerStart (MyTimer, T#2s);

Loop

When (TimerEnd (MyTimer)) Continue;

// Request the diagnostics error codes DM1 POW
stdm1stat_pow.lBuffer := sizeof (stdm1msg_pow);
SAEJ1939GetDM1 (bySAEJ1939Channel, 0x00, stdm1stat_pow,
stdm1msg_pow);

TimerStart (MyTimer, T#2s);

End_Loop;

End_Task;
```

## STX Function SAEJ1939GetDM2

**Introduction**

Calling up the function SAEJ1939GetDM2 () requests the diagnostics error codes that preceded the current one (also see SAE J1939-73 No. 5.7.2). The corresponding PGN number is 65227.

**Function Declaration**

```
Function SAEJ1939GetDM2 (
    CANNo:Int,
    bySA:Byte,
    ref stJ1939DM2stat:TJ1939DM2STAT
    ref stJ1939DM2msg:TJ1939DM2MSG
) :Int;
```

**Function Parameters**

The function SAEJ1939GetDM2 () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| bySA | Source Address of message sender | 0 ... 253<br>The own SA cannot be used |
| stJ1939DM2stat | lStatus<br>lMsgCnt<br><br>lBuffer | Lamp Status<br>Number of received messages<br>Size of variable stJ1939DM2msg |
| stJ1939DM2msg | lSPN<br>byOC<br>byFMI | Error Code<br>Error counter<br>Error Type |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |

| Device | CANMAX |
|---|---|
| JCM-620 | 2 |

**stJ1939DM2stat.lStatus**

**Default**: 0xFF00

| Type | Byte | Bit group | Description |
|---|---|---|---|
| Status | 1 | 8 - 7 | Malfunction Indicator Lamp Status |
| | | 6 - 5 | Red Stop Lamp Status |
| | | 4 - 3 | Amber Warning Lamp Status |
| | | 2 - 1 | Protect Lamp Status |
| Flash | 2 | 8 - 7 | Flash Malfunction Indicator Lamp |
| | | 6 - 5 | Flash Red Stop Lamp |
| | | 4 - 3 | Flash Amber Warning Lamp |
| | | 2 - 1 | Flash Protect Lamp |

| Type | Byte | Bit group Value | Description |
|---|---|---|---|
| Status | 1 | 00 | Lamps off |
| | | 01 | Lamps on |
| Flash | 2 | 00 | Slow Flash (1 Hz, 50 % duty cycle) |
| | | 01 | Fast Flash (2 Hz or faster, 50 % duty cycle) |
| | | 10 | Reserved |
| | | 11 | Unavailable / Do Not Flash |

**stJ1939DM2msg**

**Default Value:**
lSPN = 0
byOC = 0
byFMI = 0
For older controllers (grandfathered setting):
lSPN = 524287 (0x7FFFF)
byOC = 31 (0x1F)
byFMI = 127 (0x7F)

**Using this Function**

```
Result := SAEJ1939GetDM2 (
    1,
    0x00,
    stdm2stat_pow,
    stdm2msg_pow,);
```

**JetSym STX Program**

By calling up the function SAEJ1939GetDM2 (), the JVM-407 requests the current diagnostics error code (PGN 65227).

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;
    stdm2stat_pow : TJ1939DM2STAT;
    stdm2msg_pow : Array[10] of STJ1939DM2MSG;
End_Var;

// Initializing CAN 1
bySAEJ1939Channel := 1;
own_Source_Address := 20;
SAEJ1939Init (bySAEJ1939Channel, own_Source_Address);

// Required for a cyclical task
TaskAllEnableCycle ();
EnableEvents;

End_Task;


Task t_RequestPGN_5000 cycle 5000

Var
    Return_value  : Int;
End_Var;

// Request the diagnostics error codes DM2 POW
stdm2stat_pow.lBuffer := sizeof (stdm2msg_pow);
Return_value := SAEJ1939GetDM2 (bySAEJ1939Channel, 0x00,
stdm2stat_pow, stdm2msg_pow);

If Return_value Then
    Trace ('DM2 Request failed');
End_If;

End_Task;
```

## STX Function SAEJ1939SetSPNConversion

**Introduction**

Calling up the function SAEJ1939SetSPNConversion () determines the configuration of bytes in the message, which is requested using function SAEJ1939GetDM1 () or SAEJ1939GetDM2 (). In other words, it specifies the conversion method.

**Function Declaration**

```
Function SAEJ1939SetSPNConversion (
    CANNo:Int,
    bySA:Byte,
    iConversionMethod:Int,
) :Int;
```

**Function Parameters**

The function SAEJ1939SetSPNConversion () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| bySA | Source Address of message sender | 0 ... 253 |
| iConversionMethod | Conversion method | 1 ... 4<br>4: Automatic detection<br>2: Default |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Using this Function**

```
Result := SAEJ1939SetSPNConversion (
    1,
    0xAE,
    4);
```

# STX Function SAEJ1939GetSPNConversion

**Introduction**

Calling up the function SAEJ1939GetSPNConversion () ascertains the current conversion method set.

**Function Declaration**

```
Function SAEJ1939SetSPNConversion (
    CANNo:Int,
    bySA:Byte,
    iConversionMethod:Int,
) :Int;
```

**Function Parameters**

The function SAEJ1939GetSPNConversion () has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN channel number | 1 ... CANMAX |
| bySA | Source Address of message sender | 0 ... 253 |
| iConversionMethod | Conversion method | 1 ... 4<br>4: Automatic detection<br>2: Default |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Error when checking parameters |

**Parameter CANNo**

The value of the CANMAX parameter depends on the device. The following table provides information on this point.

| Device | CANMAX |
|---|---|
| JVM-407 | 2 |
| BTM 07 | 2 |
| BTM 012 | 1 - 2 |
| BTM 011 | n/a |
| JCM-350 | 4 |
| JCM-620 | 2 |

**Using this Function**

```
Result := SAEJ1939GetSPNConversion (
    1,
    0xAE,
    actual_conversion_method);
```

# 8 File System

**Introduction**

This chapter covers the file system of the HMI JVM-407. The file system enables access to files located on the internal flash disk, SD card or USB stick.

**Categories**

The file system distinguishes between system area with directories/files used by the operating system (OS), and application area which is available to the user.

**System Directories**

It is not possible to delete system directories. They will even survive formatting.

| Directory | Description |
|---|---|
| /System | ▪ System configuration<br>▪ System information |
| /SD | ▪ Root directory of the SD card |
| /USB | ▪ Root directory of the USB stick |

**Contents**

# 8.1 Properties

**Introduction**

This chapter covers the properties of the file system. The file system distinguishes between internal flash disk, SD card, and USB stick.

**General Properties**

The following properties apply to the internal flash disk, SD card, and USB stick:

- Maximum number of simultaneously opened files: 8
- Directory names are separated by a slash "/", not by a backslash "\".
- When a file is saved, date and time of the realtime clock of the HMI is assigned to it.
- Date, time, and/or file size are not available for all system files.

**Contents**

## Flash Disk - Properties

**Available Capacity**

The following disk space is available to the user:

| Parameter | Value |
|---|---|
| Flash disk size | 13184 KByte |

**Properties**

The internal flash disk has the following properties:

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

## SD Card - Properties

**Available Capacity**

The available capacity depends on the SD card used:

| Parameter | Value |
|---|---|
| Tested size | 8 MBytes ... 4 GBytes |

**Properties**

The SD card has the following properties:

- FAT-16 and FAT-32 compatible.
- The maximum path length is 260 characters.
- No case sensitivity.
- The following characters are not allowed in directory and file names: "/", "\", ":", "*", "?", """, "<", ">" and "|"
- No user/access administration.

## USB Stick - Properties

**Available Capacity**

The available capacity depends on the USB stick used:

| Parameter | Value |
|---|---|
| Tested size | 1 GByte ... 8 GBytes |

**Properties**

The USB stick has the following properties:

- FAT-16 and FAT-32 compatible.
- The maximum path length is 260 characters.
- No case sensitivity.
- The following characters are not allowed in directory and file names: "/", "\", ":", "*", "?", """, "<", ">" and "|"
- No user/access administration.

## 8.2 User Administration

**Introduction**

The file system for the internal flash disk offers the possibility to define authorization for access (locks) to directories, as well as to set up users with specific permissions (keys).

Users are not allowed to access directories and files for which they do not have the required key. In case of a 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 | Maximum 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**

Settings for user administration can be made in 3 files located in the directory "/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 SD cards.
- Once a file user administration has been transferred, its content can be read immediately. The settings only become active when the system is rebooted.

**Contents**

## User Administration

**Introduction**

The user administration for the file system of the JVM-407 is managed in the configuration file "/System/users.ini".

**Prerequisites**

If you want to use names for the keys, you must make them known to the JVM-407 beforehand. Therefore, set up the names first (*Setting up names for keys/locks* on page 163).

**User Administration**

Carry out the following steps for administering users:

| Step | Action |
|---|---|
| **1** | Establish an FTP connection to the JVM-407; when doing so, log in with administrator rights. |
| **2** | Open the file "/System/users.ini". |
| **3** | Make your changes to this file. |
| **4** | Save the changed file to the JVM-407. |
| **5** | Reboot the JVM-407. |

**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 used.
- In these sections values can be set which are then used by the file system.
- Blank lines can be inserted at will.
- 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 invalid or missing entry | no user account is created |
| **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 | Key for read accesses |
| Allowed values | 1 ... 31 (or corresponding names) |
| In case of missing entry | the user will not receive read keys |
| **WRITEKEYS** | |
| In the given example | openLock2,10,11 |
| Description | Key for write accesses |
| Allowed values | 1 ... 31 (or corresponding names) |
| In case of missing entry | the user will not receive write keys |
| **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. In the user administration only the password can be changed for these two users.

**As-Delivered Condition**

In as-delivered condition the content of the configuration file included in the HMI 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 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 this user, too.

**Predefined Keys**

Two out of the 31 keys have a predefined function:

| Lock / Key | Function |
|---|---|
| 1 | ■ IP configuration<br>■ User administration |
| 2 | ■ Operating system update of CPU |

## Assigning a Lock

**Introduction**

The configuration file "/System/flashdisklock.ini" is used to assign locks to directories located on the flash disk. 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 JVM-407 beforehand. Therefore, set up the names first (*Setting up names for keys/locks* on page 163).

**Assigning a Lock**

Carry out the following steps to assign a lock to a directory:

| Step | Action |
|---|---|
| 1 | Establish an FTP connection to the JVM-407; when doing so, log in with administrator rights. |
| 2 | Open the file "/System/flashdisklock.ini". |
| 3 | Make your changes to this file. |
| 4 | Save the changed file to the JVM-407. |
| 5 | Reboot the JVM-407. |

**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.
- Each directory is specified with its lock number in an individual line.
- Blank lines can be inserted at will.
- 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:

- Allowed lock numbers: 0 ... 31.
- Lock number 0: No lock is assigned to this directory. This directory can be accessed without any restrictions.
- Numbers or previously defined names can be used.

## Assigning Names to Locks/Keys

**Introduction**

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

**Assigning Names**

Carry out the following steps to assign names to keys/locks:

| Step | Action |
|---|---|
| 1 | Establish an FTP connection to the JVM-407; when doing so, log in with administrator rights. |
| 2 | Open the file "/System/keys.ini". |
| 3 | Make your changes to this file. |
| 4 | Save the changed file to the JVM-407. |
| 5 | Reboot the JVM-407. |

**Result:** 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.
- Blank lines can be inserted at will.
- 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.
- For a lock and its key the same name is used.

# 8.3 Reviewing the Flash Disk Capacity Used

**Introduction**

This chapter covers how you can review the used capacity of the user area located on the flash disk.

**Contents**

# Flash Disk Capacity Used

**Info File**

The capacity used of the user area located on the internal flash disk can be seen from the file "/System/flashdiskinfo.txt".

**Example**

In this example, the fictive capacity used of a flash disk in a JetControl 340 (4 MB) is shown:

```
Name  : flash disk
Date  : 25.11.2008
Time  : 15:04
Tracks: 64

Track  0: sectors: 128  (used:  81 / blocked:  47 / free:   0)
Track  1: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  2: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  3: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  4: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  5: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  6: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  7: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  8: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  9: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 10: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 11: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 12: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 13: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 14: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 15: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 16: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 17: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 18: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 19: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 20: sectors: 128  (used:  64 / blocked:  64 / free:   0)
Track 21: sectors: 128  (used:  85 / blocked:  43 / free:   0)
Track 22: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 23: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 24: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 25: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 26: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 27: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 28: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 29: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 30: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 31: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 32: sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track 33: sectors: 128  (used: 105 / blocked:   0 / free:  23)
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:   0 / blocked:   0 / free: 128)
Track  55:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  56:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  57:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  58:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  59:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  60:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  61:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  62:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  63:  sectors: 128  (used:   0 / blocked:   0 / free: 128)

Total:  sectors: 8192  (used: 4175 / blocked: 154 / free: 3863)

Used   :  2120900 byte
Blocked:    78232 byte
Free   :  1962404 byte
Total  :  4161536 byte
```

**Elements of Info File**

Tracks and sectors represent the administration units of the flash disk. The info file is structured accordingly and consists of the following elements:

| Element | Description |
|---|---|
| Name | Dedicated name of the flash disk |
| Date / Time | Point in time when the flash disk has been formatted last |
| Tracks | Total number of tracks |
| Track xx: sectors: 128 | Assignment of sectors of a track |
| Total: sectors: | Overall statistical data of sectors |

| Element | Description |
|---|---|
| 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 can not yet be used due to administrative reasons. |
| free | The sector is not occupied and can be used. |

## 8.4 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 chapters:

- *Operating System Update* on page 335
- *Application Program* on page 343

# 8.5 Formatting and Checking

**Introduction**

This chapter covers formatting and checking of the internal flash disk, SD card, and USB stick.

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 JVM-407 is booting, the OS checks the contents of the control register belonging to the file system. Depending on the value contained in this register the following functions are carried out:

- Formatting the flash disk
- Formatting the SD card
- Formatting the USB stick
- Checking the SD card
- Checking the USB stick

**Register Number**

The control register number of the file system is dependent on the controller:

| Controller | Register Number |
|---|---|
| JC-24x | 2936 |
| JM-D203-JC24x | 2936 |
| JC-340, JC-350, JC-360 | 202936 |
| JVM-407 | 202936 |

**Contents**

## Formatting the Flash Disk

**Introduction**

Sometimes it might be necessary to reformat the flash disk. This may be the case if an OS release has been transferred which has a different flash disk format. Or 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**

In order to cause the JVM-407 to format the internal flash disk proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the JVM-407 on. |
| **2** | Enter value -999720373 (0xc4697a4b) into the control register of the file system. |
| **3** | Switch the JVM-407 off. |
| **4** | Switch the JVM-407 on. |

**Result:** During the boot process of the JVM-407 the flash disk is formatted and the control register is set to 0.

## Formatting the SD Card

**Introduction**

Sometimes it might be necessary to reformat the SD card. This might be the case when information for flash disk administration has been destroyed.

**Consequences**

All files and directories on the SD card will be deleted!

**Formatting the SD Card**

In order to cause the JVM-407 to format the SD card proceed as follows:

| Step | Action |
|---|---|
| 1 | Switch the JVM-407 on. |
| 2 | Enter value -748362163 (0xd364e64d) into the JVM-407 register of the file system. |
| 3 | Switch the JVM-407 off. |
| 4 | Switch the JVM-407 on. |

**Result:** During the boot process of the JVM-407 the SD card is formatted and the control register is set to 0.

## Formatting the USB Stick

**Introduction**

Sometimes it might be necessary to reformat the USB stick. This might be the case when information for USB stick administration has been destroyed.

**Consequences**

All files and directories on the USB stick will be deleted!

**Formatting**

To format the USB stick proceed as follows:

| Step | Action |
|---|---|
| 1 | Power up the HMI. |
| 2 | Enter value (0x8f3d5185) into the control register of the file system. |
| 3 | De-energize the HMI. |
| 4 | Power up the HMI. |

**Result:** During the boot process of the HMI the USB card is formatted and the control register is set to 0.

## Checking the SD Card

**Introduction**

Sometimes it might be necessary to check the SD card for errors. This might be the case when the JVM-407 was switched off while accessing the SD card.

**Consequences**

- All files and directories on the SD card will be checked and errors, if any, will be fixed.
  Following such a check, the administrative structures on the SD card in consistent condition.
- Depending on the SD card size and the number of files and directories the boot process duration of the JVM-407 may extend to several minutes.

**Checking the SD Card**

In order to cause the JVM-407 to check the SD card proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the JVM-407 on. |
| **2** | Enter value 748371092 (0x2c9b3c94) into the JVM-407 register of the file system. |
| **3** | Switch the JVM-407 off. |
| **4** | Switch the JVM-407 on. |

**Result:** During the boot process of the JVM-407 the SD card is checked. The value in the control register remains unchanged so that the card is checked whenever the JVM-407 is rebooted.

**Restrictions**

This function only "repairs" the administrative structures on the SD card in order that it can be used further. However, it may happen that data of a file which has been written incompletely can't be restored in all cases.

## Checking the USB Stick

**Introduction**

Sometimes it might be necessary to check the USB stick for errors. This might be the case if the HMI was de-energized while it was accessing the USB stick.

**Consequences**

- All files and directories on the USB stick will be checked and errors, if any, will be fixed.
  Following such a check, the administrative structures on the USB stick are in consistent condition.
- Depending on the USB stick capacity and the number of files and directories to be checked the boot process of the HMI may take several minutes.

**Check**

To check the USB stick for errors proceed as follows:

| Step | Action |
|---|---|
| 1 | Power up the HMI. |
| 2 | Enter value (0x17dbd42a) into the control register of the file system. |
| 3 | De-energize the HMI. |
| 4 | Power up the HMI. |

**Result:** During the boot process of the JVM-407 the USB stick is checked. The value in the control register remains unchanged so that the stick is checked whenever the HMI is rebooted.

**Restrictions**

This function only "repairs" the administrative structures on the USB stick so that it can be used further. However, it may happen that data of a file, which has been written incompletely, can't be restored in all cases.

# 9 FTP Server

**Introduction**

The FTP server allows access to directories and files located either on an SD card, or on a flash disk integrated into the JVM-407 using an FTP client.

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

**FTP Clients**

Apart from the command line FTP client, which comes with many PC operating systems, graphic FTP tools can be used, as well.

**Number of Possible Connections**

The FTP server on the JVM-407 is able to manage 4 FTP connections simultaneously. That is, up to 4 FTP client programs can be connected with the JVM-407 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 perform 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**

## Login

**Login**

To have access to the file system via FTP, the FTP client must log in and provide its user name and password when starting the communication.

**As Delivered Condition**

In its original configuration the controller is delivered with two user accounts:

[USER1]
NAME=admin
PW=admin

[USER33]
NAME=system
PW=system

**Administration of Users**

Via user administration of the file system, the password can be modified and new users can be added.

**Related Topics**

- **User administration** on page 156

## Supported Commands

**Supported Commands**

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

| Command | Purpose |
|---|---|
| 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:<br>■ Type A with interpretation N<br>■ Type I<br>■ 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; must be followed by the command "RNTO" |
| RNTO | Indicates the new name of the file which has been specified by the command "RNFR" before. |
| PASV | The FTP server changes into "passive mode" |

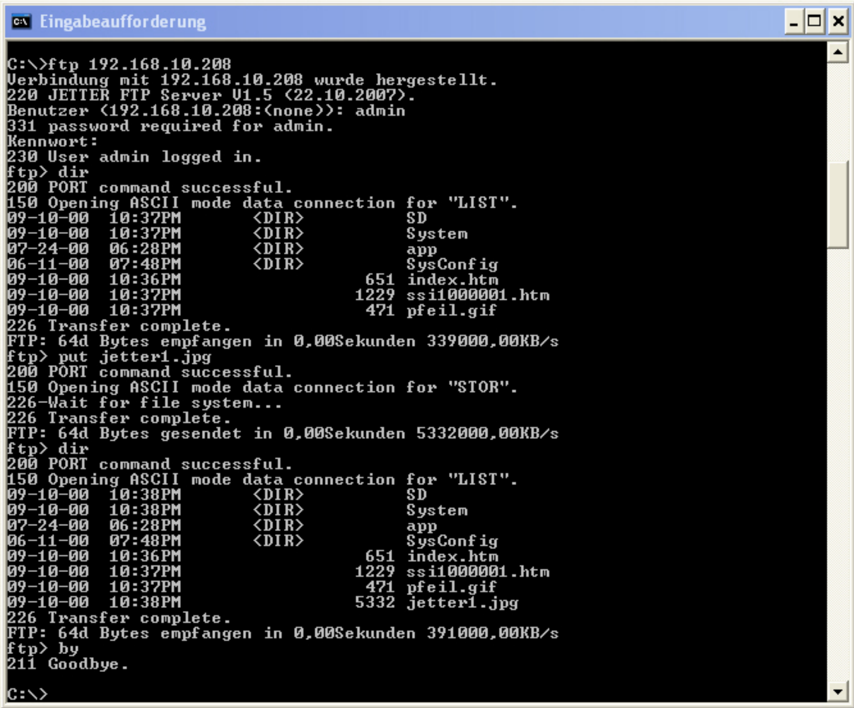
## Example: Windows FTP Client

**Task**

The following tasks are to be carried out using an FTP client, for example, the one which comes with Windows XP:

- Invoking the FTP client by opening a connection
- Loging 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"
- Re-displaying the content of the current directory using "dir"
- Terminating the session and the FTP client using "bye"

**Action**



# 10 HTTP Server

**Introduction**

The HTTP server can be accessed via standard browser. 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

**Enabling the HTTP Server Feature**

To enable the HTTP server feature in the controller the following requirements have to be met:

- When ordering the controller option -W has been selected.

If both requirements have been met, the corresponding bit in status register "Web" is set.

**Required Programmer's Skills**

To perform 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**

# 10.1 Server Side Includes

**Introduction**

Current realtime controller values can be displayed in an HTML page using the **Server Side Includes** (SSI) feature in the HTTP server.

**Rules**

A **name space tag** has to be specified at the beginning of the HTML page that is to contain the realtime controller values. This name space tag is for defining the **name space** used in the HTML page.

In the body section of the HTML page the **Data Tags** are specified.

**Updating Realtime Controller Values**

When the page is loaded into the browser, the HTTP server once replaces the data tags by current controller values.

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

**Contents**

## Name Space Tag

**Name Space Tag - Structure**

The **Name Space Tag** must be the first entry in the HTML file. Its structure is as follows:

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

with **NS** representing the **name space**. A character string with a maximum length of 63 characters can be chosen for the name space.

The **Name Space** 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, JW is used for Name Space.

## Inserting Realtime Controller Values

**Introduction**

Actual realtime controller values can be integrated into the parameters of 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 variables can be 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 variable value is 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 realtime controller value is multiplied |
| Comments | This operation is executed prior to adding the offset |
| Example | `factor="1.5"` |

| **offset** | |
|---|---|
| Description | Value which is added to the realtime controller value |
| Comments | This operation is executed after multiplication by the factor |
| Example | `offset="1000"` |

**Format Definition**

The representation of variables can be defined by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" allows to output leading zeroes. This option applies to the following register types: INT, INTX and REAL.
- Prefix "+" allows to output a sign. This option applies to the following register types: INT, and REAL.
- Prefixing a blank allows to output a space character for positive values. This option applies to the following 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<br>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:**

The content of register 1000250 is multiplied by 3.25, then, 500 is added to the product, and the result is displayed with sign and at least five integer positions. Leading zeros are added if necessary. Furthermore, three decimal positions are inserted.

**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<br>Flag = 1 --> Display: 1 |
| STRING | Flag = 0 --> Display: FALSE<br>Flag = 1 --> Display: TRUE |

Standard type: BOOL

**Example:**

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

**Result:**
The state of flag 100 is inserted 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<br>Input = 1 --> Display: 1 |
| STRING | Input = 0 --> Display: OFF<br>Input = 1 --> Display: ON |

Standard type: BOOL

**Example:**

```
<JW:DTAG name="I100000308" type="STRING" />
```

**Result:**
The state of input 100000308 on the CPU is inserted 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<br>Output = 1 --> Display: 1 |
| STRING | Output = 0 --> Display: OFF<br>Output = 1 --> Display: ON |

Standard type: BOOL

**Example:**

```
<JW:DTAG name="O100000308" />
```

**Result:**
The state of output 100000308 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 the number of which 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 the 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 the 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**

Current realtime controller values are to be inserted into an HTML page.

The HTML page is then to be displayed in a browser using the **Server Side Includes** feature of the HTTP server.

**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 maybe that way <JC:DTAG name="R201000" type="BOOL" />, if only
boolean queries are used.
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>For inputs and outputs by analogy: <JC:DTAG name="PI1000130"
type="BOOL" />
or <JC:DTAG name="100000205" type="STRING" />. </p>
<p>R201000 = <JC:DTAG name="R201000" type="INT"
format="+0##########" /> </p>
<p>Regards </p>
<p>Your JetControl</p>
</body>

</html>
```

# 11 Programming

**Purpose of this Chapter**

This chapter is for supporting you in programming the HMI JVM-407 in the following fields of activity:

- Programming additional functions

**Prerequisites**

To be able to program the HMI JVM-407 the following prerequisites must be fulfilled:

- The HMI is connected to a PC.
- The programming tool JetSym is installed on the PC.

**Contents**

## Abbreviations, Module Register Properties and Formats

**Abbreviations**

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

| Abbreviation | Meaning |
|---|---|
| R 100 | Register 100 |
| MR 150 | Module register 150 |

**Module Register Properties**

Each module register is characterized by certain properties. For many module registers most properties are identical. For example, their value after reset is 0. 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 following a reset | 0 or undefined (e.g. release number) |
| Takes effect | Immediately |
| Write access | Always |
| Data type | Integer |

**Number Formats**

The number formats used in this document are listed in the following table:

| Notation | Number Format |
|---|---|
| 100 | Decimal |
| 0x100 | Hexadecimal |
| 0b100 | Binary |

**JetSym Sample Programs**

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

| Notation | Meaning |
|---|---|
| `Var, When, Task` | Key words |
| `BitClear();` | Instructions |
| `100 0x100 0b100` | Constant numerical value |
| `// This is a comment` | Comments |
| `// ...` | Further program processing |

# 11.1 Memory Overview

**Introduction**

The JVM-407 features several types of program and data memories. There is volatile memory that requires power to maintain the stored information, and non-volatile memory which does not require power to maintain the stored information. 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 JVM-407 is powered up.

**Features**

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

**Memory Access**

- The user is not allowed to directly access the OS memory.
- Changes to the OS can be made by means of an OS update.

**Related Topics**

- **Updating the Operating System** on page 336

## File System Memory

**Introduction**

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

**Features**

- Internal flash disk and SD memory card
- Non-volatile
- Slow access: milliseconds up to seconds
- Limited number of write/delete cycles: approx. 1 million
- Internal flash disk size: 12.875 MBytes
- SD card size: 32 MByte to 4 GByte

**Memory Access**

- by operating system
- by JetSym
- via FTP connection
- by 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 HMI and is stored to it.

**Features**

- Stored as file within the file system
- Default directory: "/app"
- files may also be stored to other directories (or on SD card)
- Size: 256 KByte max.

**Memory Access**

- by operating system
- by JetSym
- via FTP connection
- by means of file commands from within the application program

**Related Topics**

- **Application Program** on page 343

## Memory for Volatile Application Program Variables

**Introduction**

Volatile variables are used to store data which may be discarded when the JVM-407 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 within the application program

**JetSym STX Program**

In the following program a global variable is incremented by 1 every 2 seconds:

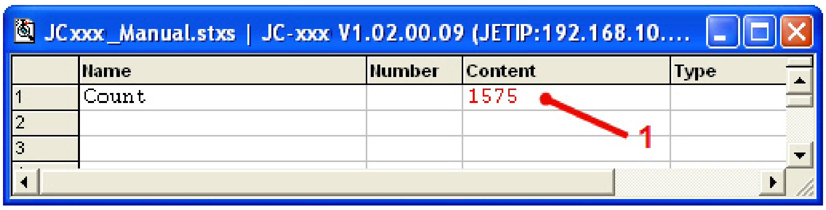
```
#Include "Platforms.stxp"

Var
    Count:    Int;
End_Var;

Task Increment 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 1 every 2 seconds. |

## Memory for Non-Volatile Application Program Registers

**Introduction**

Non-volatile registers are used to store data which must be maintained when the JVM-407 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: 6.000
- Register numbers: 1,000,000 through 1,005,999

**Memory Access**

- by JetSym
- by e-mail client
- by browser (via HTTP server)
- from HMIs
- from within the application program
- from other controllers/HMIs

**JetSym STX Program**

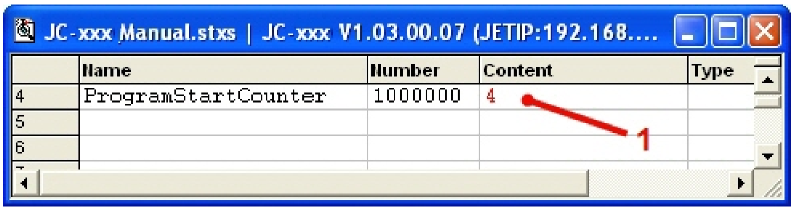
In the following program a register variable is incremented by 1 every time the application program is launched. Thus, it is used to count the number of program launches.

```
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 | Content | Description |
|---|---|---|
| **1** | Present content of the register variable | The content of the register variable is incremented by 1 every time the program is launched. |

## Memory for Non-Volatile Application Program Variables

**Introduction**

Non-volatile variables are used to store data which must be maintained when the JVM-407 is de-energized.

**Properties**

- Global variables which are assigned to permanent registers (%RL)
- Variables are stored in a compact way
- Size: 24,000 bytes
- Register numbers: 1,000,000 through 1,005,999

**Memory Access**

- by JetSym
- from HMIs
- from within the application program

**JetSym STX Program**

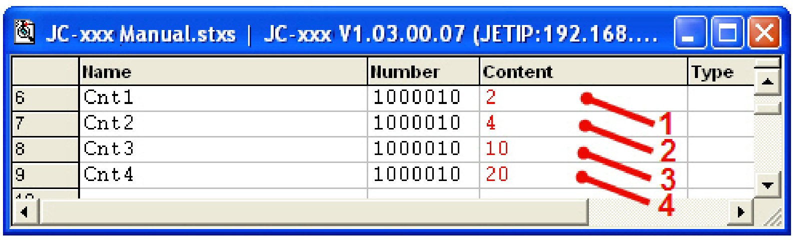
In the following program 4 non-volatile variables are incremented every second. The working range of the counters is between 0 and 255 (variable type: byte). For these 4 variables the 4 bytes of register 1000010 are used.

```
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 4 counters is byte, this will result in counter overflow after a relatively short time:




| Number | Content | Description |
|---|---|---|
| **1** | Present content of the variable Cnt1 | The content of the variable is incremented by 1 every second. |

| Number | Content | Description |
| --- | --- | --- |
| **2** | Present content of the variable Cnt2 | The content of the variable is incremented by 2 every second. |
| **3** | Present content of the variable Cnt3 | The content of the variable is incremented by 5 every second. |
| **4** | Present content of the variable Cnt4 | The content of the variable is incremented by 10 every second. |

## Special Registers

**Introduction**

Special registers are used to control OS functions and to 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: 100,000 through 999,999

**Memory Access**

- by JetSym
- via e-mail client
- by browser (via HTTP server)
- from HMIs
- from within the application program
- from other controllers

**JetSym STX Program**

In the following program 2 special registers are used. The first is the special register for status LEDs, the second is the special register for digipot values. In this task, the value is just copied from the special register containing digipot values to the special registers assigned to the status LEDS. If the application program is running on the HMI and the user turns the digipot, the value contained in the digipot special register is displayed by the status LEDs.

```
Var
    Digipot:    Int at %VL 363000;
    Status_LEDs:Int at %VL 362100;
End_Var;

Task Main Autorun

    Loop
        Status_LEDs:= Digipot;
    End_Loop;

End_Task;
```

## 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 through 36000

**Memory Access**

- by JetSym
- via e-mail client
- by browser (via HTTP server)
- from HMIs
- from within the application program

**JetSym STX Program**

The following program is for dimming the background lighting of the HMI if input In11 is set.

```
Var
    In11        :Bool at %XL 362100.10;
    //Background lighting
    BackgroundLighting    :Int at %VL 364000;
End_Var;

Task Main Autorun
    Loop
            // If In11 is set, then
            If In11 Then
                //dim background lighting
                Inc(BackgroundLighting);
                Delay(T#30ms);
            End_If;
    End_Loop;
End_Task;
```

# Flag

**Introduction**

Flags are 1-bit operands. This means they can either have the value TRUE or FALSE.

**Properties of User Flags**

- Global variables assigned to permanent addresses (%MX)
- Non-volatile
- Quantity: 256
- Flag numbers: 0 through 255

**Properties of Overlaid User Flags**

- Global variables assigned to permanent addresses (%MX)
- Non-volatile
- Overlaid by registers 1000000 through 1000055
- Quantity: 1,792
- Flag numbers: 256 through 2047

**Properties of Special Flags**

- Global variables assigned to permanent addresses (%MX)
- When the operating system is launched, special flags are initialized using default values.
- Quantity: 256
- Flag numbers: 2048 through 2303

**Memory Access**

- by JetSym
- by e-mail client
- by browser (via HTTP server)
- from HMIs
- from within the application program

**JetSym STX Program**

In the following program, a flag is set when the user presses key F1. If on an HMI key F2 is pressed, the flag is reset. As long as this flag is set, special register 361000 (Status LED) is incremented. Incrementing of the special register continues until the flag is reset.

```
Var
    Flag1:          Bool at %MX 1;
    Input_Button_1: Bool at %XL 361000.0;
    Input_Button_2: Bool at %XL 361000.1;
    Status_LEDs:    Int  at %VL 362100;
End_Var;

Task Main Autorun
    Flag1:= False;
    Loop
        If Input_Button_1 Then
            Flag1 := True;
        ElseIf Input_Button_2 Then
            Flag1:= False;
```

```
        End_IF;

        If Flag1 Then
            Inc(Status_LEDs);
            Delay(T#100ms);
        End_If;
    End_Loop;
End_Task;
```

# 11.2 Inputs and Outputs

**Introduction**

This chapter covers the programming of inputs and outputs, controls and ignition and switching off delay for the JVM-407.

**Contents**

# Function Keys

**Introduction**

The HMI JVM-407 has four function keys F1 to F4. The function keys are freely programmable.

**Special Registers**

In register 361000 of the JVM-407, there is a bit-coded mapping of the function keys which can be used for programming.

**JetSym STX Program**

**Prerequisites:**

So that the status LEDs are not also controlled via the JVM-407 inputs, the inputs IN1 to IN10 should not be set whilst running the sample program.

In the following sample program, the function keys are continuously interrogated in one task. Pressing one or more keys controls the status LEDs assigned in the program.

```
Var
    F_Button_Register: Int At %VL 361000;

    Status_Led_1: Bool At %XL 362100.0;
    Status_Led_2: Bool At %XL 362100.1;
    Status_Led_3: Bool At %XL 362100.2;
    Status_Led_4: Bool At %XL 362100.3;
End_Var;

Task Main Autorun
    F_Button_Register := 0;
    Loop
        If F_Button_Register.0 Then
             Status_Led_1 := True;
        Else Status_Led_1 := False;
        End_If;
        If F_Button_Register.1 Then
             Status_Led_2 := True;
        Else Status_Led_2 := False;
        End_If;
        If F_Button_Register.2 Then
             Status_Led_3 := True;
        Else Status_Led_3 := False;
        End_If;
        If F_Button_Register.3 Then
             Status_Led_4 := True;
        Else Status_Led_4 := False;
        End_If;
    End_Loop;
End_Task;
```

# Digipot

**Introduction**

The JVM-407 has a digipot with pushbutton feature, which offers a convenient input option. The following provides details of the digipot's special registers with a corresponding sample program.

**Digipot Registers**

The following special registers exist for the digipot:

| Registers | Description |
|---|---|
| 363000 | This register counts up and down when the digipot is rotated and contains the current reading. Here, the following applies:<br>▪ **Rotate digipot clockwise** = register incremented<br>▪ **Rotate digipot counter-clockwise** = register decremented |
| 363001 | Bit 0: 0 = Digipot not pressed<br>Bit 0: 1 = Digipot pressed |
| 363002 | The lower limit for the digipot reading is specified here. If the digipot is further rotated counter-clockwise, the register 363000 remains at this minimum value. |
| 363003 | The upper limit for the digipot reading is specified here. If the digipot is further rotated clockwise, the register 363000 remains at this maximum value. |

**JetSym STX Program**

In the following sample program, the background lighting for the JVM-407 is dimmed using the digipot. An upper and lower limit for the digipot is specified for this purpose. Pressing the digipot sets full background lighting.

```
Var
    Digipot_Count     :  Int At %VL 363000;
    Digipot_Limit_min:  Int At %VL 363002;
    Digipot_Limit_max:  Int At %VL 363003;
    Digipot_Button    :  Int At %VL 363001;
    BackgroundLighting:  Int At %VL 364000;
End_Var;

Task Main Autorun
    Digipot_Count := 0;
    Digipot_Limit_max := 17;
    Digipot_Limit_min := 0;
    Loop
      If Digipot_Button Then
                 BackgroundLighting := 255;
            Else BackgroundLighting := Digipot_Count*15;
      End_If
    End_Loop
End_Task;
```

# Digital Inputs and Outputs

**Introduction**

The HMI JVM-407 has the following inputs and outputs:

- 15 digital inputs. Ten of these have a fixed connection with status LEDs and five are freely programmable.
- 1 digital output, e.g. to control a bypass relay. However, outputs are always set simultaneously to enable provision of a higher current.

**Special Registers**

The following registers are available for the digital inputs and outputs:

| Register | Description |
|---|---|
| 362100 | Bit-coded mapping of digital inputs **IN1 - IN15**.<br>**IN1 - IN10** are linked to the JVM-407 status LEDs.<br>**Example:**<br>Bit 0 = 1: IN1 in and status LED 1 on. |
| 362200 | Bit 0 of the register is responsible for setting the digital output.<br>Bit 0 = 1: Digital output is set. |

**JetSym STX-Program**

In this sample program, the freely programmable input IN11 is continuously interrogated. If this input is set, then the 2 digital outputs are set, which serve to control e.g. a bypass relay.

```
Var
    IN11:    Bool At %XL 362100.10;
    // Digital outputs
    Output:  Bool At %XL 362200.0;
End_Var;

Task Main Autorun
    Loop
        // If In11 is set, then
        If IN11 Then
            // Set the digital outputs
            Output := True;
            Delay(T#100ms);
        End_If;
    End_Loop;
End_Task;
```

# Ignition and Switching Off Delay

**Introduction**

The ignition and shutdown function are detailed here.

**Special Registers**

The special register 361100 of the JVM-407 is responsible for prompting ignition. Here, the following applies:

| If ... | ... Then ... |
|---|---|
| Bit 0 = 0 | Ignition is switched on and voltage is applied to KL 15 ignition (+). |
| Bit 0 = 1 | Ignition is switched off and no voltage is applied to KL 15 ignition (+). |

**Default Ignition Function**

The HMI has the following default settings in connection with ignition:

| If ... | ...and... | ... Then ... |
|---|---|---|
| the power supply is connected to the HMI | the ignition is off | the HMI does not boot up. |
| the power supply is connected to the HMI | the ignition is on | the HMI boots up. |
| the HMI is running | the ignition is switched off (not the power supply) | then the HMI remains switched on. |

**Shutdown Function - Options**

Notwithstanding the default ignition function, the Shutdown function provides the following options:

- The HMI can be individually shut down.
- The HMI can be restarted.

**Function Declaration**

Function **Shutdown** (**Reboot:**Bool) :Bool;

**Function Parameters**

The Shutdown () function has the following parameters.

| Parameter | Description | Value |
|---|---|---|
| Reboot | System restart:<br>System shutdown: | True<br>False |

**Return Value**

The function transfers the following return values to the higher-level program.

| Return Value | |
|---|---|
| 0 | ok |
| -1 | Ignition is still switched on |

**Note**

If the ignition is still switched on, the device will not be switched off. However, a restart will always be performed and is not dependent on the ignition.

**JetSym STX Program**

In the sample program, the **Shutdown ()** function is executed after 3 seconds, if the ignition of the vehicle is switched off. The **Reboot** parameter for the **Shutdown ()** function has the value **false**. This means that the device is switched off.

```
Var

    Ignition: Int At %VL 361100;
End_Var;

Task Ign Autorun
    Loop
     When Ignition Continue;
            Delay(3000);
            Shutdown(False);
    End_Loop;
End_Task;
```

## 11.3 Realtime Clock (RTC)

**Introduction**

The JVM-407 is equipped with a timing circuit (realtime clock for date and time). This clock continues to work even when the JVM-407 is deenergized.

**Usage by OS**

The realtime clock is used by the OS for the following functions:

- File date and time when creating a file

**Restrictions**

When using the realtime clock the following restrictions have to be taken into account:

- When the JVM-407 is deenergized the power reserve is limited.
- The RTC has no automatic daylight savings time function

**Contents**

# Technical Data

**Technical Data - Real-Time Clock**

| Parameter | Description |
|---|---|
| Power reserve | 4 years |
| Deviation | Max. 1 minute per month |

**Behavior when the Power Reserve has Elapsed**

If the HMI has been de-energized for a longer period of time and the RTC power reserve has elapsed, it takes the following actions when re-booting:

| Stage | Description |
|---|---|
| **1** | During the boot process the HMI detects that the power reserve has elapsed. |
| **2** | Date and time are set to their default values:<br>Date: Saturday, January 01, 2000<br>Time: 0:00 a.m. |

**As-Delivered Condition**

In as-delivered condition the date is Saturday, 01 January, 2000.

## Sample Program for Real-Time Clock

**Task**

Actual date and time from the JVM-407 are to be displayed in JetSym.

**Solution**

An application program task reads out the realtime clock at regular intervals and outputs the readings properly formatted as trace message. These readings can be displayed in JetSym when trace mode has been activated.

**JetSym STX Program**

```
#Include "Platforms.stxp"

Type
    // structure of 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 one second
        Delay(T#1s);
        // copy actual time and date to buffer
        Dummy := RTCregs.Trigger;

        // show day of 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;
            // show date
            Trace(StrFormat(' , %2d.%02d.%4d , ',
                            RTCregs.Day,
                            RTCregs.Month,
                            RTCregs.Year + 2000));
            // show time (plus cr/lf)
            Trace(StrFormat('%2d:%02d:%02d$n',
                            RTCregs.Hour,
                            RTCregs.Minute,
                            RTCregs.Second));
        End_Loop;
    End_Task;
```

# 11.4 Runtime Registers

**Introduction**

The JVM-407 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 Runtime Registers

**Overview of Registers**

The following registers are used in this manual:

| Registers | Description |
|---|---|
| **R 201000** | Application time base in milliseconds |
| **R 201001** | Application time base in seconds |
| **R 201002** | Application time base in R 201003 * 10 milliseconds |
| **R 201003** | Application time base unit for R 201002 |
| **R 201004** | System time base in milliseconds |

**R 201000**

## Application time base in milliseconds

Every millisecond this register is incremented by 1.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (with overflow function) |

**R 201001**

## Application time base in seconds

Every second this register is incremented by 1.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (with overflow function) |

**R 201002**

## Application time base in application time base units

Every [201003] * 10 milliseconds this register is incremented by 1. Using the reset value in register 201003 of 10, this register is incremented every 100 milliseconds.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (with overflow function) |

**R 201003**

### Application time base unit for R 201002

This register contains the multiplier for runtime register R 201002.

| Register properties | |
|---|---|
| Values | 1 ... 2,147,483,647 (* 10 ms) |
| Value following reset | 10 (--> 100 ms) |
| Enabling Conditions | after at least 10 ms |

**R 201004**

### System time base in milliseconds

Every millisecond this register is incremented by 1.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (with overflow function) |
| Access | Read access |

## Sample Program - Runtime Registers

**Task**

Measure how much time it takes to store variable values to a file.

**Solution**

Before storing the values register 201000 is set to 0. Once the values have been stored, from this register can be seen how much time it took to store the values [in milliseconds].

**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
        // clear start flag
        WriteIt := False;
        // wait until start flag set by user
        When WriteIt Continue;
        // open file in write mode
        If FileOpen(File1, '/Test.dat', fWrite) Then
            // restart timer register
            MilliSec := 0;
            // write array data to file
            FileWrite(File1, DataArray,
                      SizeOf(DataArray));
            // capture time
            WriteTime := MilliSec;
            FileClose(File1);
            // show measured time
            Trace(StrFormat('Time : %d [ms]$n',
                                WriteTime));
        Else
            // show error message
            Trace('Unable to open file!$n');
        End_If;
    End_Loop;
End_Task;
```

# 11.5 Monitoring the Interface Activity

**Introduction**

Several servers for variables have been integrated into the HMI to make variables used within the HMI 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 places through these interfaces.

**Monitored Interface Activities**

The following interface activitites can be monitored:

- JetIP server via Ethernet interface
- STX debug server via Ethernet interface

**Application**

The monitoring function for interface activities can be used, amongst others, for the following scenarios:

- Plants requiring process visualization to ensure safe operation can be transferred into a save condition if communications fails.
- When the service technician connects an HMI, the application program automatically displays additional status information.

**Contents**

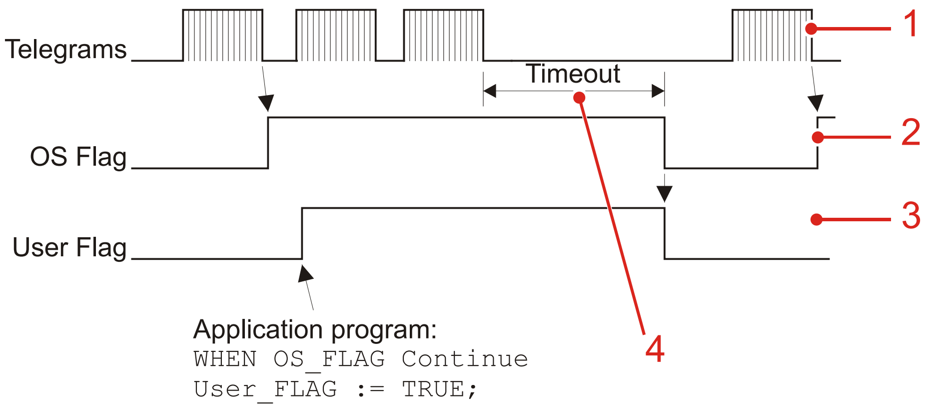
## Operating Principle

**Introduction**

The activity of a client communicating with a server in the JVM-407 can be monitored from the application program by means of two special flags and one special register per interface.

**Overview**

The diagram below shows the interdependence between interface activity and the two special flags, as well as the special register:




| Number | Element | Function |
|---|---|---|
| **1** | Telegrams | Requests from client to server |
| **2** | OS flag | OS flag set by the JVM-407 once a request has been received. |
| **3** | User flag | The user flag should be set in the application program once the OS flag has been set. This indicates that the connection has temporarily been disrupted even if the OS flag is reset very quickly. |
| **4** | Timeout | Time of inactivity after which both special flags are reset by the OS. This time can be set in a special register. |

**Description**

Interface activities are monitored as follows:

| Stage | Description |
|---|---|
| **1** | To activate monitoring mode the desired value is entered into the timeout register from within the application program. |
| **2** | When the JVM-407 receives the next telegram, it sets the corresponding OS flag. |
| **3** | Once the OS flag has been set, the corresponding user flag is set in the application program. |
| **4** | Each new telegram causes the timeout to restart. |
| **5** | If telegrams cease to arrive, both special flags are reset by the JVM-407 upon expiry of the timeout interval. |

| | |
|---|---|
| **6** | The application program detects that the special flags have been reset and takes appropriate action. |
| **7** | When further telegrams start to arrive, the JVM-407 sets the corresponding OS flag. The user flag, however, remains reset. |

# Programming

**Registers/Flags - Overview**

The following registers and flags are used in this manual:

### Timeout Registers

| Register | Interface | Application |
|---|---|---|
| **R 203000** | JetIP via Ethernet | ▪ Visualization<br>▪ Networking |
| **R 203005** | STX debug via Ethernet | ▪ JetSym via Ethernet |

### Special Flags

| Flag | Interface | Application |
|---|---|---|
| **F 2088** | JetIP via Ethernet | OS flag |
| **F 2089** | | User-defined flag |
| **F 2098** | STX debug via Ethernet | OS flag |
| **F 2099** | | User-defined flag |

**R 203000**

### Timeout in the case of JetIP via Ethernet

This register contains the timeout for the JetIP server via 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 debug via Ethernet

This register specifies the timeout for STX debug server via 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 OS flag of this interface is set by the HMI. |
| **3** | Set the corresponding user flag. |

**Timeout Detection**

To detect a timeout, proceed as follows:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Enable monitoring of interface activities (see above).</td></tr>
<tr><td>2</td><td>Wait until the user flag of this interface is reset by the HMI.<br>Result: A timeout has occurred.</td></tr>
<tr><td>3</td><td>Check the corresponding OS flag
<table>
<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>
</td></tr>
</table>

# 11.6 E-Mail

**Introduction**

E-mails are created using template files into which variable values are inserted as required when the e-mail is sent. E-mails are sent from the HMI 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 HMI JVM-407, and on how to create and send e-mails.

**Required Programmer's Skills**

To perform the functions described in this chapter, 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 of the HMI.
- The user must be familiar with IP networks.

**Contents**

## 11.6.1 Configuring the E-Mail Feature

**Introduction**

This chapter gives a description on how to configure the e-mail feature so as to allow sending of e-mails from within the application program.

During the boot process, the JVM-407 reads out configuration data from the file "/EMAIL/email.ini".

**Prerequisites**

When creating the configuration file, the following requirements have to be met:

- The IP address of the e-mail server must be 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* on page 79).
- The log-on and authentication parameters at the e-mail server must be 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 JVM-407 is based on the contents of the file "/EMAIL/email.ini". This file is read out only when the controller is booting.

**File Structure**

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

- These sections are for setting values which are then used by the e-mail client.
- Blank lines can be inserted as required.
- The following characters precede a comment line: "!", "#" or ";".

**Sections**

The configuration file contains up to 3 sections. Section [SMTP] is mandatory. The other sections have to be created only in case they are actually required.

| Section | Configuration Values |
|---|---|
| [SMTP] | ■ IP address and port number of SMTP server<br>■ Log-on parameters |
| [POP3] | ■ IP address and port number of POP3 server<br>■ Log-on parameters |
| [DEFAULT] | ■ Name of an e-mail template file containing default values |

## Section [SMTP]

**Introduction**

In this section the parameters are specified which are used to connect to 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 JVM-407 to log on at the SMTP server before an e-mail can be sent. During the logon process USER and PASSWORD have to be entered. JetControl 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;<br>can also be specified as name. |
| Allowed values | ■ > 1.0.0.0<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address |
| In case of illegal value or missing entry | E-mail feature will not be available |

| PORT | |
|---|---|
| In the given example | 25.000 |
| Description | Port number of SMTP server |
| Allowed values | ■ > 0<br>■ < 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 JVM-407 uses the entry contained in [FROM] |

| **USER** | |
|---|---|
| In the given example | JetControl0815 |
| Description | Log-on name for SMTP authentication. If this entry exists, a PASSWORD must be specified, 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, a USER must be specified, 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 the parameters are specified which are used to connect to the POP3 server.

This section is only needed if the e-mail server, to which the e-mails are to be sent, requires authentication through POP3-before-SMTP.

**Example:**

```
[POP3]
IP       = 192.168.40.1
PORT     = 25100
USER     = JetControl4711
PASSWORD = Pop3PassWord
```

**Authentication**

This type of authentication requires the JVM-407 to establish a connection to the POP3 server first. During this process USER and PASSWORD have to be entered. After that, the SMTP server allows to send e-mails for a given period of time (mostly 10 to 30 minutes).

**Configuration Values**

| **IP** | |
|---|---|
| In the given example | 192.168.40.1 |
| Description | IP address of POP3 server;<br>can also be specified as name. |
| Allowed values | ■ > 1.0.0.0<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ 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<br>■ < 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, a PASSWORD must be specified, 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, a USER must be specified, 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 the name of an e-mail template file is specified which contains default settings for e-mails. The settings made here will be used when sending an e-mail if the corresponding section in an e-mail template is missing.

**Example**

```
[DEFAULT]
MAILCFG = EmailDefaults.cfg
```

**Related Topics**

- **Structure of Template File** on page 235

## 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
```

## 11.6.2 Creating E-Mails

**Introduction**

This chapter gives a description on how to create e-mails so as to allow sending them from within the application program.

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 an e-mail for sending.

**File Name**

`email_#.cfg`

| Part of the name | Description |
|---|---|
| email_ | Constant prefix |
| # | Number of e-mail; value between 0 and 255 |
| .cfg | Constant file extension |

**Storage Location**

E-mail template files and the configuration file have to be stored to the same directory on the internal flash disk:

`/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 this section can be tagged with realtime controller values (refer to *Inserting Realtime Controller Values* on page 184).

[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]`<br>`JetControl@jetter.de` |

| **[TO]** | |
|---|---|
| Description | E-mail addressee |
| Comments | Several addressees are separated by ";". |
| Length | 255 characters |
| Example | `[TO]`<br>`service@mydomain.com` |

| [CC] | |
|---|---|
| Description | Additional e-mail addressee(s) |
| Comments | Several addressees are separated by ";". |
| Length | 255 characters |
| Example | `[CC]`<br>`service@mydomain.com;hotline@mydomain.com` |

| [SUBJECT] | |
|---|---|
| Description | Subject |
| Length | 255 characters |
| Example | `[SUBJECT]`<br>`Fatal Error` |

| [ATTACHMENT] | |
|---|---|
| Description | Complete name of the file to be attached |
| Comments | This file must be a text file. |
| Length | 511 characters |
| Example | `[ATTACHMENT]`<br>`/logfiles/error_report.log` |

| [MESSAGE] | |
|---|---|
| Description | E-mail message text |
| Comments | Text only message |
| Length | 65,535 characters |
| Example | `[MESSAGE]`<br>`Have a nice day !`<br>`JetControl.` |

## Inserting Realtime Controller Values

**Introduction**

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

**Tag Delimiters**

All tags start and end with defined strings. Between these tag delimiters variables can be defined:

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

**Defining Variables**

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** | |
|---|---|
| Function | Variable Name |
| Comments | Code letter followed by the variable number |
| Example | `name="R1000023"` |

| **type** | |
|---|---|
| Function | Variable type of notation |
| Example | `type="REAL"` |

| **format** | |
|---|---|
| Function | Representation format |
| Comments | Refer to format definition |
| Example | `format="+0####.###"` |

| **factor** | |
|---|---|
| Function | Factor by which the realtime controller value is multiplied |
| Comments | This operation is executed prior to adding the offset |
| Example | `factor="1.5"` |

| **offset** | |
|---|---|
| Function | Value which is added to the realtime controller value |
| Comments | This operation is executed after multiplication by the factor |
| Example | `offset="1000"` |

**Format Definition**

The representation of variables can be defined by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" allows to output leading zeroes. This option applies to the following register types: INT, INTX and REAL.
- Prefix "+" allows to output a sign. This option applies to the following register types: INT, and REAL.
- Prefixing a blank allows to output a space character for positive values. This option applies to the following 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<br>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:**

The content of register 1000250 is multiplied by 3.25. Then, 500 is added to the product, and the result is displayed with sign and at least five integer positions. Leading zeros are added if necessary. Furthermore, three decimal positions are inserted.

**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<br>Flag = 1 --> Display: 1 |
| STRING | Flag = 0 --> Display: FALSE<br>Flag = 1 --> Display: TRUE |

Standard type: BOOL

**Example:**

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

**Result:**

The state of flag 100 is inserted as string "T" or "F".

**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.

**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.

## 11.6.3 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 JVM-407 creates the e-mail based on the e-mail template file and inserts variable values if required.

**Processing by 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. However, only one e-mail function call can be carried out at a time. 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.

**Contents**

## Sending E-Mails Using the System Function

**Introduction**

A system function is used for sending e-mails.

**JetSym STX**

```
Systemfunction(110, &RegEmailNo, &RegResult);
```

| Parameter | Description |
|---|---|
| RegEmailNo | Number of the register that contains the number of the e-mail to be sent. This number is part of the file name of the e-mail template file. |
| RegResult | Number of the register to which the result of this function will be stored. |

**Return Value**

This function will produce one of the following return values:

| Result | Description | Possible error cause |
|---|---|---|
| **0** | No error | |
| **-1** | Insufficient memory | Operating system error |
| **-2** | FROM not defined | The e-mail template file is faulty or could not be located. |
| **-3** | TO not defined | Error in e-mail template file |
| **-4** | No connection to the e-mail server or error during data transfer to the e-mail server. | ■ No e-mail server available<br>■ An error occurred during configuration using file "/EMAIL/email.ini"<br>■ Data transfer error |
| **-10** | E-mail function not available. Bit 2 in register "Web status" not reset | Initialization error. For example, configuration file "/EMAIL/email.ini" does not exist or is faulty. |
| **-12** | Internal error | Operating system error |

## Sample Program

**E-Mail Template File**

The e-mail template file is stored to the JVM-407 under the name "/EMAIL/email_0.cfg".

```
[FROM]
JVM-407

[TO]
test1@test.mail

[CC]
test2@test.mail

[SUBJECT]
Test <JW:DTAG name="R1000000" />

[ATTACHMENT]
/System/config.ini

[MESSAGE]
Register 1000001 (int) = <JW:DTAG name="R1000001"
                          format="+0######" />
Register 1000001 (hex) = <JW:DTAG name="R1000001"
                          type="INTX" />
Text register: <JW:DTAG name="R1001000"
                type="STRING" />"
Float register: <JW:DTAG name="R1001900" type="REAL"
                 factor="2.35" offset="100" />
Flag 10: <JW:DTAG name="F10" type = "STRING" />
Output R[1000113] = <JW:DTAG name="PO1000113"
                     type="BOOL" />

Have a nice day...
JVM-407
```

**JetSym STX Program**

Set "bSend" to cause an e-mail to be sent.

```
Var
    Counter:    Int    At %VL 1000000;
    TestReg:    Int    At %VL 1000001;
    TextReg:    String At %VL 1001000;
    FloatReg:   Float  At %VL 1001900;

    RegEmail:   Int    At %VL 1000200;
    RegResult:  Int    At %VL 1000201;
    Send:       Bool;
End_Var;
```

```
Task SendMail Autorun
    Counter   := 0;
    TestReg   := 1234;
    TextReg   := 'Hello World !';
    FloatReg  := 20.5;
    RegEmail  := 0;
    Loop
        Send := False;
        When Send Continue;
        Inc(Counter);
        SystemFunction(110, &RegEmail, &RegResult);
    End_Loop;
End_Task;
```

## 11.6.4 Registers

**Introduction**

This chapter gives a description of those registers from which the status of e-mail processing can be seen.

**Contents**

# Overview of Registers

**Introduction**

The JVM-407 provides several registers from which the status of e-mail processing can be seen.

**Overview of Registers**

| Register(s) | Description |
|---|---|
| **202930** | Web status |
| **292932** | IP address of SMTP server |
| **292933** | IP address of POP3 server |
| **292934** | Port number of 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

In register "Web status" all available functions are displayed (bit-coded).

| 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 | |
|---|---|
| Access | Read access |
| Value following a reset | Depending on options purchased |

**R 292932**

### IP Address of 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 | |
|---|---|
| Access | Read access |
| Value following a 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 | |
|---|---|
| Access | Read access |
| Value following a reset | Depending on configuration |
| Takes effect | once R 202930.2 = 1 |

R 292934

## Port Number of 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 | |
|---|---|
| Access | Read access |
| Value following a 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 | |
|---|---|
| Access | Read access |
| Value following a reset | Depending on configuration |
| Takes effect | once R 202930.2 = 1 |

R 292937

## Status of E-Mail Processing

With the help of this registers 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 JVM-407 |
| | 2 | E-mail is being compiled and connection with the server is being established. |
| | 3 | E-mail was sent to the server |
| Access | Read access | |

**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 | None of the tasks is sending an e-mail |
| Value following a reset | 255 | |
| Access | Read access | |

# 11.7 Modbus/TCP

**Introduction**

This chapter describes the functions of the Modbus/TCP server and client integrated into JVM-407.

**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**

# 11.7.1 Modbus/TCP Server

**Introduction**

In the case of a valid license (Modbus/TCP feature is enabled) and after successful launch of the Modbus/TCP server, an external client can access registers, inputs and outputs.

This chapter covers the addressing process and describes the commands supported by the Modbus/TCP server.

**Number of Possible Connections**

4 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 lower-order 16 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 respective basic address for accessing registers, inputs and outputs can be 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 JVM-407 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 value 1000000.

Hence, registers 1000100 through 1000104 are read.

## Supported Commands - Class 0

| | |
|---|---|
| **fc 3** | **read multiple registers**<br>Reading register blocks:<br><br>The starting register number within JVM-407 is calculated as follows: Register number specified in the command plus the content of register 272702 *Register Offset*. |
| **fc 16** | **write multiple registers**<br>Writing register blocks<br><br>The starting register number within JVM-407 is calculated as follows: Register number specified in the command plus the content of register 272702 *Register Offset*. |

## Supported Commands - Class 1

**fc 1**

**read coils**

Reading outputs.

The output number within the JVM-407 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 JVM-407 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 JVM-407 is calculated as follows: Register number specified in the command plus the content of register 272702 *Register Offset*.

**fc 5**

**write coil**

Enabling/disabling an individual output.

The output number within the JVM-407 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 lower-order 16 bits of a register.

The starting register number within JVM-407 is calculated as follows: Register number specified in the command plus the content of register 272702 *Register Offset*.

## Supported Commands - Class 2

| | |
|---|---|
| **fc 15** | **force multiple coils** |
| | Enabling/disabling several outputs |
| | The output number within the JVM-407 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 JVM-407 is calculated as follows: Register number specified in the command plus the content of register 272702 *Register Offset*. |

## 11.7.2 Modbus/TCP Client

**Introduction**

The Modbus/TCP client included in JVM-407 supports only Class 0 Conformance. This means that 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 acyclical or cyclical transmission to a Modbus/TCP server using system functions.

**Number of Possible Connections**

Connections to 11 different Modbus/TCP servers may be opened at the same time.

**Acyclical Data Transmission**

System functions 65 and 67 (reading registers), as well as 66 and 68 (writing registers) can be used to establish a acyclical 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 (cyclical data transmission), this connection will be used. Setting-up and clearing-down the connection is, therefore, not required.

**Cyclical Data Transmission**

The configurable function **RemoteScan** is for cyclically transferring the inputs and outputs 20001 through 36000 that are combined in the 16-bit registers 278000 through 278999 from and to the configured 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 / Outputs**

| Registers | Inputs and Outputs |
|---|---|
| 278000 | 20001 ... 20016 |
| 278001 | 20017 ... 20032 |
| 278002 | 20033 ... 20048 |
| ... | ... |
| 278999 | 35985 ... 36000 |

These registers and inputs/outputs mapped to them are merely storage cells within the RAM. The registers are not directly mapped 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*. This "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 special functions for reading and writing registers (system functions 67 and 68), as well as for initializing RemoteScan (system function 85) have been provided. These special 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 lower-order 16 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**

# System Function 65: Acyclical Reading of Registers

**Introduction**

Using system function 65, a register block from a Modbus/TCP server is copied to the registers of the local memory.

**Important Notes**

- While this system function is being carried out, simultaneous calls of this function in other tasks are blocked until this function will be completed.
- While this system function is being executed, it is not advisable to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym. In the a.m. cases the connection remains open and it might happen that further communication is blocked.
- The IP address is always to be specified directly. It cannot be specified using names.

**Function Declaration**

```
Systemfunction(65, &StructModbusTCP, &RegResult);
```

| Parameter | Function |
|---|---|
| StructModbusTCP | Structure of the type MODBUS_TCP |
| RegResult | Number of the register to which the result of this function will be stored. |

**Type Declaration MODBUS_TCP**

```
Type
  MODBUS_TCP:
  Struct
    IPadress    : Int;
    Port        : Int;
    Timeout: Int;
    Source      : Int;
    Destination : Int;
    Values      : Int;
  End_Struct;
End_Type;
```

**Function Parameters**

| Parameter | Value | Comment |
|---|---|---|
| IPadress | IP-Address of Modbus/TCP Server | direct input |
| Port | 502 | |
| Timeout | in ms | |
| Source | remote | Register number of source |
| Destination | local | Register number of destination |
| Values | 1 ... 125 | Quantity of registers |

**Return Value**

This function will produce one of the following return values:

| Return Value | |
|---|---|
| 0 | No error |
| -1 or -2 | Error during connection set-up |
| -4 | Error during data transfer |
| -5 | Error message from Modbus/TCP Server |
| -8 | Timeout |
| -10 | No Modbus/TCP license |

**Related Topics**

- **Example of an Application** on page 265

## System Function 67: Acyclical Reading of Registers

**Introduction**

Using system function 67, a register block from a Modbus/TCP server is copied to the registers of the local memory.

Unlike system function 65, the Unit ID can be set here.

**Important Notes**

- While this system function is being carried out, simultaneous calls of this function in other tasks are blocked until this function will be completed.
- While this system function is being executed, it is not advisable to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym. In the a.m. cases the connection remains open and it might happen that further communication is blocked.
- The IP address is always to be specified directly. It cannot be specified using names.

**Function Declaration**

```
Systemfunction(67, &StructModbusTCP, &RegResult);
```

| Parameter | Function |
|---|---|
| StructModbusTCP | Structure of the type MODBUS_TCP |
| RegResult | Number of the register to which the result of this function will be stored. |

**Type Declaration MODBUS_TCP**

```
Type
  MODBUS_TCP:
  Struct
    IPadress     : Int;
    Port         : Int;
    Timeout: Int;
    Source       : Int;
    Destination : Int;
    Values       : Int;
    UnitID       : Int;
    Internal_1   : Int;
    Internal_2   : Int;
  End_Struct;
End_Type;
```

**Function Parameters**

| Parameter | Value | Comment |
|---|---|---|
| IPadress | IP-Address of Modbus/TCP Server | direct input |
| Port | 502 | |
| Timeout | in ms | |
| Source | remote | Register number of source |
| Destination | local | Register number of destination |
| Values | 1 ... 125 | Quantity of registers |
| UnitID | 0 ... 255 | Unit ID |
| Internal_1 | 0 | Internal usage |
| Internal_2 | 0 | Internal usage |

**Return Value**

This function will produce one of the following return values:

| Return Value | |
|---|---|
| 0 | No error |
| -1 or -2 | Error during connection set-up |
| -4 | Error during data transfer |
| -5 | Error message from Modbus/TCP Server |
| -8 | Timeout |
| -10 | No Modbus/TCP license |

## System Function 66: Acyclical Writing of Registers

**Introduction**

Using system function 66, the content of registers contained in the local memory is copied to the Modbus/TCP server as a register block.

**Important Notes**

- While this system function is being carried out, simultaneous calls of this function in other tasks are blocked until this function will be completed.
- While this system function is being executed, it is not advisable to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym. In the a.m. cases the connection remains open and it might happen that further communication is blocked.
- The IP address is always to be specified directly. It cannot be specified using names.

**Function Declaration**

```
Systemfunction(66, &StructModbusTCP, &RegResult);
```

| Parameter | Function |
|---|---|
| StructModbusTCP | Structure of the type MODBUS_TCP |
| RegResult | Number of the register to which the result of this function will be stored. |

**Type Declaration MODBUS_TCP**

```
Type
  MODBUS_TCP:
  Struct
    IPadress     : Int;
    Port         : Int;
    Timeout: Int;
    Source       : Int;
    Destination  : Int;
    Values       : Int;
  End_Struct;
End_Type;
```

**Function Parameters**

| Parameter | Value | Comment |
|---|---|---|
| IPadress | IP-Address of Modbus/TCP Server | direct input |
| Port | 502 | |
| Timeout | in ms | |
| Source | local | Register number of source |
| Destination | remote | Register number of destination |
| Values | 1 ... 125 | Quantity of registers |

**Return Value**

This function will produce one of the following return values:

| Return Value | |
|---|---|
| 0 | No error |
| -1 or -2 | Error during connection set-up |
| -4 | Error during data transfer |
| -5 | Error message from Modbus/TCP Server |
| -8 | Timeout |
| -10 | No Modbus/TCP license |

**Related Topics**

- **Example of an Application** on page 265

## System Function 68: Acyclical Writing of Registers

**Introduction**

Using system function 68, the content of registers contained in the local memory is copied to the Modbus/TCP server as a register block.

Unlike system function 66, the Unit ID can be set here.

**Important Notes**

- While this system function is being carried out, simultaneous calls of this function in other tasks are blocked until this function will be completed.
- While this system function is being executed, it is not advisable to carry out TaskBreak or TaskRestart for this task or to restart the program via JetSym. In the a.m. cases the connection remains open and it might happen that further communication is blocked.
- The IP address is always to be specified directly. It cannot be specified using names.

**Function Declaration**

```
Systemfunction(68, &StructModbusTCP, &RegResult);
```

| Parameter | Function |
|---|---|
| StructModbusTCP | Structure of the type MODBUS_TCP |
| RegResult | Number of the register to which the result of this function will be stored. |

**Type Declaration MODBUS_TCP**

```
Type
  MODBUS_TCP:
  Struct
    IPadress    : Int;
    Port        : Int;
    Timeout     : Int;
    Source      : Int;
    Destination : Int;
    Values      : Int;
    UnitID      : Int;
    Internal_1  : Int;
    Internal_2  : Int;
  End_Struct;
End_Type;
```

**Function Parameters**

| Parameter | Value | Comment |
|---|---|---|
| IPadress | IP-Address of Modbus/TCP Server | direct input |
| Port | 502 | |
| Timeout | in ms | |
| Source | local | Register number of source |
| Destination | remote | Register number of destination |
| Values | 1 ... 125 | Quantity of registers |
| UnitID | 0 ... 255 | Unit ID |
| Internal_1 | 0 | Internal usage |
| Internal_2 | 0 | Internal usage |

**Return Value**

This function will produce one of the following return values:

| **Return Value** | |
|---|---|
| 0 | No error |
| -1 or -2 | Error during connection set-up |
| -4 | Error during data transfer |
| -5 | Error message from Modbus/TCP Server |
| -8 | Timeout |
| -10 | No Modbus/TCP license |

## Example of an Application

**Task**

JetControl is to cyclically exchange I/O data with two Modbus/TCP servers on the network.

On external request, the content of a single register is to be sent to one of the two communication partners.

**Solution**

For cyclic data transmission the function "RemoteScan" is used. System functions 80 and 81 are executed one after the other.

The value contained in a single register is sent to the second communication partner in acyclical mode using system function 66.

**Action**

First, the configuration data are entered into the structures required for configuring the RemoteScan function. The starting address of these structures is transferred along with other data when system function 80 (InitRscan) is invoked. If initialization was successful, RemoteScan function is started via system function 81 (StartRscan) and cyclic communication sets in.

Then, the parameter structure for acyclic data transmission is prepared. Setting flag "Send" triggers a register block to be sent to a second communication partner one time.

**JetSym STX Program**

```
Type

    RSCAN_HEADER:
    Struct
        Protocol        :    Int;
        Units           :    Int;
    End_Struct;

    RSCAN_ELEMENT:
    Struct
        Ipadress        :    Int;
        Port            :    Int;
        UpdateRate      :    Int;
        OutRegs         :    Int;
        OutSource       :    Int;
        OutDestination  :    Int;
        InRegs          :    Int;
        InSource        :    Int;
        InDestination   :    Int;
        Status          :    Int;
        Timeout         :    Int;
    End_Struct;

    RSCAN_STATUS:
    Struct
        Status          :    Int;
        Error           :    Int;
        ErrCnt          :    Int;
```

```
    End_Struct;

    MODBUS_TCP:
    Struct
        Ipadress        :    Int;
        Port            :    Int;
        Timeout         :    Int;
        Source          :    Int;
        Destination     :    Int;
        Values          :    Int;
    End_Struct;
End_Type;

Const
    RscanRegs        = 1000100;
    RscanStatRegs    = 1001000;
    Elements         = 2;

    InitRscan        = 80;
    StartRscan       = 81;

    ProtModbusTCP    = 5;
    ModbusTCPPort    = 502;

    Rscan            = 0;
End_Const;

Var
    RemoteScan         : RSCAN_HEADER At %VL RscanRegs;

    RscanElements      : Array[Elements] Of RSCAN_ELEMENT At
                         %VL RscanRegs + Regsizeof(RSCAN_HEADER);

    RscanStatus        : Array[Elements] Of RSCAN_STATUS  At
                         %VL RscanStatRegs;

    ModbusTCP          : MODBUS_TCP At %VL 1000500;

    Result             : Int        At %VL 1000099;
    Send               : Bool       At %MX 1;
End_Var;


Task tRscan Autorun
    RemoteScan.Protocol := ProtModbusTCP;
    RemoteScan.Units    := Elements;

    // first communication unit
    RscanElements[0].Ipadress        := IP#192.168.10.211;
```

```
        RscanElements[0].Port             := ModbusTCPort;
        RscanElements[0].UpdateRate       := 50;
        RscanElements[0].OutRegs          := 3;
        RscanElements[0].OutSource        := 278000;
        RscanElements[0].OutDestination   := 20000;
        RscanElements[0].InRegs           := 3;
        RscanElements[0].InSource         := 21000;
        RscanElements[0].InDestination    := 278100;
        RscanElements[0].Status           := &RscanStatus[0];
        RscanElements[0].Timeout          := 20;

        // second communication unit
        RscanElements[1].Ipadress         := IP#192.168.10.150;
        RscanElements[1].Port             := ModbusTCPort;
        RscanElements[1].UpdateRate       := 20;
        RscanElements[1].OutRegs          := 5;
        RscanElements[1].OutSource        := 278300;
        RscanElements[1].OutDestination   := 20000;
        RscanElements[1].InRegs           := 10;
        RscanElements[1].InSource         := 25000;
        RscanElements[1].InDestination    := 278400;
        RscanElements[1].Status           := &RscanStatus[1];
        RscanElements[1].Timeout          := 200;

        Systemfunction(InitRscan, &RemoteScan, &Result);
        If Result > 0 Then
            Systemfunction(StartRscan, 0, &Result);
        End_If;

        ModbusTCP.Ipadress         := IP#192.168.10.212;
        ModbusTCP.Port             := ModbusTCPort;
        ModbusTCP.Values           := 1;
        ModbusTCP.Source           := 1040000;
        ModbusTCP.Destination      := 1050000;
        ModbusTCP.Timeout          := 100;
        Send                       := False;

        Loop
            When Send Continue;
            Systemfunction(66, &ModbusTCP, &Result);
            Send := False;
        End_Loop;

    End_Task;
```

# 11.8 User-programmable IP Interface

**The user-programmable IP interface**

The user-programmable IP interface allows to send or receive data via Ethernet interface on the JVM-407 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 exchange data 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 masks etc.)
- TCP (e.g. connection establishment/termination, data stream, data backup etc.)
- UDP (e.g. datagram, etc.)

**Restrictions**

For communication via user-programmable IP interface, ports which are already used by the operating system of the controller must NOT be used. 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 file /EMAIL/email.ini | 25, 110 | e-mail client |
| TCP | 502 | | Modbus/TCP Server |
| TCP, UDP | 1024 - 2047 | | various users |
| TCP, UDP | IP configuration | 50000, 50001 | JetIP |
| TCP | IP configuration | 52000 | Debug server |

**Contents**

## 11.8.1 Programming

**Introduction**

The user-programmable IP interface is used to exchange data between application program and network clients 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 JVM-407. Carry out the following steps to program this feature:

| Step | Action |
|---|---|
| 1 | Initializing the user-programmable IP interface |
| 2 | Establishing the connection(s) |
| 3 | Transferring data |
| 4 | Terminating the connection(s) |

**Technical Data**

Technical data of the user-programmable IP interface:

| Feature | Description |
|---|---|
| Number of connections | 20 |
| Maximum data size | 4,000 bytes |

**Restrictions**

In the application program, tasks serving the user-programmable IP interface should not be stopped through `TaskBreak` or restarted through `TaskRestart` while the JVM-407 is processing one of these functions. Failure to do so could result in the following errors:

- Connections are not opened
- Data loss during sending or receiving
- Connections, which should be terminated, remain established
- Connections, which should be used, are terminated

**Contents**

# Initializing the User-Programmable IP Interface

**Introduction**

The user-programmable IP interface must be initialized at least each time the application program is launched.

**Function Declaration**

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

**Return Value**

The following return value is possible:

| Result of the function | |
|---|---|
| 0 | always |

**Using the Function**

This function can be used and its return value be assigned to a variable for further utilization in the following way:

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

**Operating Principle**

The JVM-407 processes the function in the following steps:

| Stages | Description |
|---|---|
| 1 | All established connections of the user-programmable IP interface are terminated |
| 2 | All OS-internal data structures of the user-programmable IP interface are initialized |

**Related Topics:**


- **Establishing a connection** on page 272
- **Terminating a connection** on page 281
- **Sending data** on page 276
- **Receiving data** on page 278

## Establishing a Connection

**Introduction**

Before data can be sent or received, a connection has to be established. In doing so, it must be decided which transport protocol (TCP or UDP) is to be used and whether a client or a server should be established.

**Function Declaration**

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

**Function Parameters**

Description of function parameters:

| Parameter | Value | Comment |
|---|---|---|
| ClientServerType | Client = 1 = CONNTYPE_CLIENT<br>Server = 2 = CONNTYPE_SERVER | |
| IPType | UDP/IP = 1 = IPTYPE_UDP<br>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 is positive, the connection could have been established. If the return value is negative, an error occurred and the connection could not be established.

| Return Value | |
|---|---|
| > 0 | A positive return value must be stored to a variable, since it has to be passed on as handle with functions for receiving and sending data via this connection, as well as for terminating this connection. |
| -1 | Error during connection set-up |
| -2 | Internal error |
| -3 | Invalid parameter |
| -8 | Timeout |

**Using this Function with a TCP/IP Client**

This function can be used and its return value be assigned to a variable for further utilization in the following way if a client is to establish a TCP/IP connection to a server:

```
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. The following stages are taken when processing this function:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The JVM-407 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>2</td><td><table>
<tr><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</td><td>stage 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></td></tr>
</table>

**Using this Function with a TCP/IP Server**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if a server is to establish a TCP/IP connection to a client:

```
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. The following stages are taken when processing this function:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The JVM-407 sets up TCP/IP port 46000 for receiving connection requests</td></tr>
<tr><td>2</td><td>
<table>
<tr><th>If ...</th><th>... Then ...</th></tr>
<tr><td>the network client has established an connection</td><td>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</td></tr>
<tr><td>the connection has not been established and the timeout of 100 seconds has not elapsed</td><td>the system waits for a connection being established</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>
</td></tr>
</table>

**Using this Function with a UDP/IP Client**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if a client is to establish a UDP/IP connection:

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

**Functioning Principle with a UDP/IP Client**

As UDP is a connectionless type of communication, the controller simply opens a communication channel which is used to send data to a network client. The following stages are taken when processing this function:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The JVM-407 sets up a UDP/IP communication channel for sending data</td></tr>
<tr><td>2</td><td>
<table>
<tr><th>If ...</th><th>... Then ...</th></tr>
<tr><td>no error has occurred</td><td>the function is terminated and a positive value is returned as handle for further access to the connection</td></tr>
<tr><td>an error has occurred</td><td>the function is terminated and a negative value is returned</td></tr>
</table>
</td></tr>
</table>

**Using this Function with a UDP/IP Server**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if a server is to establish a UDP/IP connection:

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

**Functioning Principle with a UDP/IP Server**

As UDP is a connectionless type of communication, the server simply opens a communication channel over which a network client is able to receive data.The following stages are taken when processing this function:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The JVM-407 sets up a UDP/IP communication channel at port 46000 for receiving data</td></tr>
<tr><td>2</td><td><table><tr><th>If ...</th><th>... Then ...</th></tr><tr><td>no error has occurred</td><td>the function is terminated and a positive value is returned as handle for further access to the connection</td></tr><tr><td>an error has occurred</td><td>the function is terminated and a negative value is returned</td></tr></table></td></tr>
</table>

**Related Topics:**


- **Terminating a connection** on page 281
- **Sending data** on page 276
- **Receiving data** on page 278
- **Initializing the user-programmable IP interface** on page 271

## 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 can not 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 function parameters:

| Parameter | Value | Comment |
|---|---|---|
| IPConnection | Handle | Result of the function when establishing the connection |
| 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 | |
| DataLen | 1 .. 4,000 | Data block length in bytes |

**Return Value**

The following return values are possible:

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

**Using this Function with a TCP/IP connection**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if data are to be sent via TCP/IP connection:

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

**Functioning Principle with a TCP/IP Connection**

When using TCP/IP, data are sent via a previously established connection. Therefore, it is not required to specify the IP address and IP port and can be ignored in the function. The task stops at the command until data have been sent and acknowledgment has been received or an error has occurred.

**Using this Function with a UDP/IP Client**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if data are to be sent from a client via

UDP/IP connection:

```
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 2 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. There will be no acknowledgement that the data have been received by the remote network client.

**Related Topics:**


- **Initializing the user-programmable IP interface** on page 271
- **Establishing a Connection** on page 272
- **Terminating a connection** on page 281
- **Receiving data** on page 278

## Receiving Data

**Introduction**

Data can be sent via a previously established TCP/IP connection or via a UDP/IP connection of a server. Via UDP/IP connection of a client data can not be received, but only sent.

**Function Declaration**

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

**Function Parameters**

Description of function parameters:

| Parameter | Value | Comment |
|---|---|---|
| IPConnection | Handle | Return value when establishing the connection |
| IPAddr | Address of a variable for storing the sender's IP Address | Required only for UDP/IP server |
| IPPort | Address of a variable for storing the sender's IP port | 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**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if data are to be received via TCP/IP connection:

```
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 established connection. Therefore, it is not required to specify the IP address and IP port and can be ignored in the function. The task will pause at the command until the data are received or an error has occurred. In case of a TCP/IP connection, data are transmitted as data stream.

The JVM-407 processes the function in the following stages:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The JVM-407 waits until data have been received, but no longer than the specified timeout</td></tr>
<tr><td>2</td><td><table><tr><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></table></td></tr>
<tr><td>3</td><td><table><tr><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 data are buffered by the JVM-407 an can be retrieved from within the application by invoking the function several times</td></tr></table></td></tr>
<tr><td>4</td><td>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**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way if data are to be received from a server via UDP/IP connection:

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

**Functioning Principle with a UDP/IP Server**

The task will pause at the command until all of the data are received or an error has occurred. In case of a UDP/IP connection, data are transmitted as datagram.

The JVM-407 processes the function in the following stages:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The JVM-407 waits until all data of a datagram have been received, but no longer than the specified timeout</td></tr>
<tr><td>2</td><td><table><tr><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></table></td></tr>
<tr><td>3</td><td><table><tr><th>If ...</th><th>... Then ...</th></tr><tr><td>more data have been received than could have been copied into the receiving buffer (that is, if the sent datagram is too large)</td><td>these data are discarded</td></tr></table></td></tr>
<tr><td>4</td><td>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>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** on page 271
- **Establishing a Connection** on page 272
- **Terminating a connection** on page 281
- **Sending Data** on page 276

## 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 function parameters:

| Parameter | Value | Comment |
|---|---|---|
| IPConnection | Handle | Return value when establishing the connection |

**Return Value**

The following return values are possible:

| Return Value | |
|---|---|
| 0 | Connection terminated and deleted |
| -1 | Invalid handle |

**Using the Function**

This function can be invoked and its return value be assigned to a variable for further utilization in the following way:

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

**Related Topics:**


- **Establishing a Connection** on page 272
- **Sending Data** on page 276
- **Receiving Data** on page 278
- **Initializing the user-programmable IP interface** on page 271

## 11.8.2 Registers

**Introduction**

This chapter describes the registers of a JVM-407 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-940MC, JVM-407 | 350000 | 350000 ... 350007 |

**Determining Register Numbers**

In this chapter only the last figure of a register number is specified. To calculate the actually used register number the basic register number of the corresponding device must be added.

**Overview of Registers**

| Register(s) | Description |
|---|---|
| **MR 0** | Selection of a connection |
| **MR 1** | Type of connection |
| **MR 2** | Transport protocol |
| **MR 3** | IP address |
| **MR 4** | IP port |
| **MR 5** | Status |
| **MR 6** | Number of sent bytes |
| **MR 7** | Number of received bytes |

## Register Description

**Introduction**

Established connections are managed by the operating system in a list. Module register MR0 *Selection of a connection* is used to copy connection details into other registers of a register block.

**MR 0**

### Selection of a connection

Write access to this register is used to select connections and to display their details in the following registers. Read access 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.

| Module Register Properties | |
|---|---|
| Values | 0 ... 65,535 |

**MR 5**

### Status

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 |

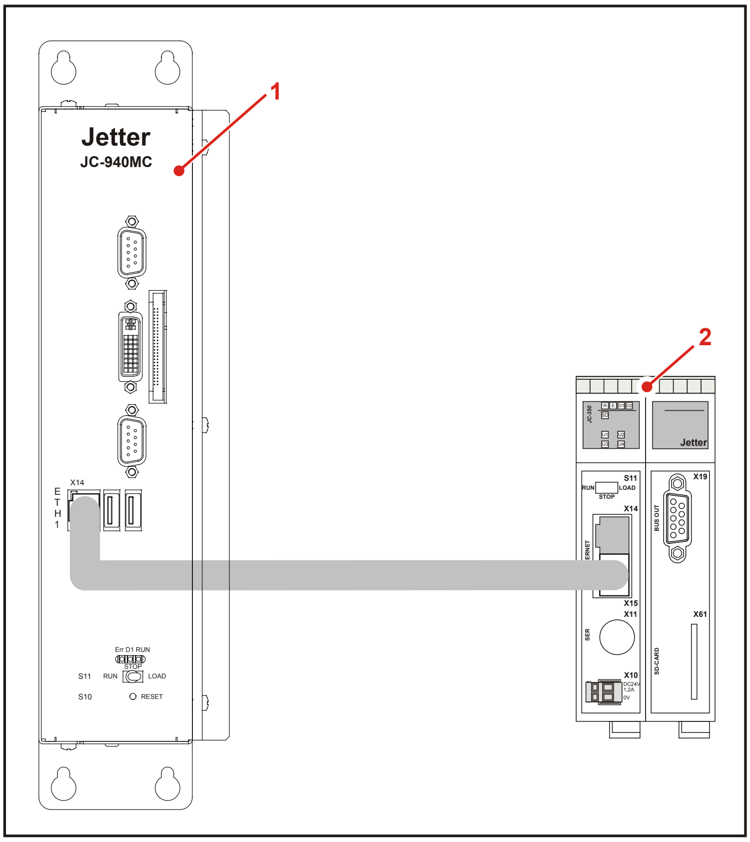
## 11.8.3 Sample Programs

**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 | Component | Function |
|---|---|---|
| **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.

**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* on page 287.

**JetSym STX Program**

```
Const
  TCP_PORT = 52100;
  MSG_LEN  = 4000;
End_Const;

Var
  // connection handle
  ConnHandle      : Int;

  // send 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;

  // receive error count
  RecvErrors      : Int;
  // send error count
  SendErrors      : Int;
  // valid communication counter
  CommCnt         : Int;

  AmountToReceive : Int;

  // dummy
  NotUsed         : Int;
End_Var;


Task TCPserver Autorun

  Var
```

```
  RecvTimer      : Timer;
  ReceiveCnt     : Int;
End_Var;

// connection timeout
ConnTimeOut := T#5s;
// receiption timeout
RecvTimeOut := T#5s;

// expected amount of data to receive
AmountToReceive := MSG_LEN;

// close all connections, init. data structures
ResConnInit := ConnectionInitialize();

If ResConnInit >= 0 Then

  Trace('Server running.$n');

  While (True) Do

    // try to connect
    ResConnCreate := ConnectionCreate
                     ( CONNTYPE_SERVER,
                       IPTYPE_TCP,
                       0,
                       TCP_PORT,
                       ConnTimeOut );

    If ResConnCreate > 0 Then

      Trace('Connection established.');
      // save connection handle
      ConnHandle := ResConnCreate;

      // loop, as long as connection established
      Loop

        // timeout for the complete data packet
        TimerStart(RecvTimeout, RecvTimeOut * 5);
        // init. receive data counter
        ReceiveCnt := 0;

        // loop until all expected data received or timeout
        While ReceiveCnt < AmountToReceive
            And Not TimerEnd(RecvTimer) Do

          ResConnReceive := ConnectionReceiveData
                            ( ConnHandle,
                              NotUsed,
                              NotUsed,
```

```
                                     RecvBuf[ReceiveCnt],
                                     SizeOf(RecvBuf),
                                     RecvTimeOut );

      If ResConnReceive > 0 Then
        // something received, increment counter
        ReceiveCnt := ReceiveCnt + ResConnReceive;
      Else
        // error on receive
        ResConnReceive := -1;
        // increment error counter
        inc(RecvErrors);
        // leave loop
        Exit;
      End_If;
    End_While;

    // here is the point to implement the server function;
    // in this example we simply return the received data
    If ReceiveCnt Then
      // copy from receive to send buffer
      MemCopy(SendBuf,RecvBuf,SizeOf(SendBuf));
      ResConnSend := ConnectionSendData
                       ( ConnHandle,
                         0,
                         0,
                         SendBuf,
                         ReceiveCnt );
      If ResConnSend < 0 Then
        // increment error counter
        Inc(SendErrors);
      End_If;
    End_If;

    If ResConnSend >= 0 And ResConnReceive >= 0 Then
      // no error --> increment OK counter
      Inc(CommCnt);
    Else
      // leave loop
      Exit;
    End_If;

  End_Loop;

  If ConnHandle > 0 Then
    // close connection
    ConnectionDelete(ConnHandle);
    // no longer valid
    ConnHandle := 0;
    Trace('Connection close.$n');
  End_If;
```

```
        End_If;

        // wait a little bit before trying to reconnect
        Delay(T#3s);

      End_While;

    Else
      Trace('ConnectionInitialize() failed, server stopped !$n');
    End_If;

  End_Task;
```

**Related Topics:**

- **Client** on page 292

## 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* on page 287.

**JetSym STX Program**

```
Const
  TCP_ADDR = IP#192.168.10.210;
  TCP_PORT = 52100;
  MSG_LEN  = 4000;
End_Const;

Var
  // connection handle
  ConnHandle       : Int;

  // send 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;

  // receive error count
  RecvErrors       : Int;
  // send error count
  SendErrors       : Int;
  // valid communication counter
  CommCnt          : Int;

  AmountToReceive  : Int;
  SendDelay        : Int;

  // dummy
  NotUsed          : Int;
End_Var;


Task TCPclient Autorun
```

```
Var
  RecvTimer    : Timer;
  ReceiveCnt     : Int;
End_Var;

// connection timeout
ConnTimeOut := T#5s;
// receiption timeout
RecvTimeOut := T#5s;

// expected amount of data to receive
AmountToReceive := MSG_LEN;
SendDelay       := T#500ms;

// close all connections, init. data structures
ResConnInit := ConnectionInitialize();

If ResConnInit >= 0 Then

  Trace('Client running.$n');

  While (True) Do

    // try to connect
    ResConnCreate  := ConnectionCreate
                     ( CONNTYPE_CLIENT,
                       IPTYPE_TCP,
                       TCP_ADDR,
                       TCP_PORT,
                       ConnTimeOut );

    If ResConnCreate > 0 Then

      Trace('Connection established.');
      // save connection handle
      ConnHandle := ResConnCreate;

      // loop, as long as connection established
      Loop

        ResConnSend := ConnectionSendData
                      ( ConnHandle,
                        0,
                        0,
                        SendBuf,
                        AmountToReceive );
        If ResConnSend < 0 Then
          // increment error counter
          Inc(SendErrors);
        End_If;
```

```
      // timeout for the complete data packet
      TimerStart(RecvTimer, RecvTimeOut * 5);
      // init. receive data counter
      ReceiveCnt := 0;

      // loop until all expected data received or timeout
      While ReceiveCnt < AmountToReceive
          And Not TimerEnd(RecvTimer) Do

        ResConnReceive := ConnectionReceiveData
                             ( ConnHandle,
                               NotUsed,
                               NotUsed,
                               RecvBuf[ReceiveCnt],
                               SizeOf(RecvBuf),
                               RecvTimeOut );

        If ResConnReceive > 0 Then
          // something received, increment counter
          ReceiveCnt := ReceiveCnt + ResConnReceive;
        Else
          // error on receive
          ResConnReceive := -1;
          // increment error counter
          Inc(RecvErrors);
          // leave loop
          Exit;
        End_If;
      End_While;

      If ResConnSend >= 0 And ResConnReceive >= 0 Then
        // no error --> increment OK counter
        Inc(CommCnt);
        Delay(SendDelay);
      Else
        // leave loop
        Exit;
      End_If;

    End_Loop;

    If ConnHandle > 0 Then
      // close connection
      ConnectionDelete(ConnHandle);
      // no longer valid
      ConnHandle := 0;
      Trace('Connection close.$n');
    End_If;

  End_If;
```

```
      // wait a little bit before trying to reconnect
      Delay(T#3s);

    End_While;

  Else
    Trace('ConnectionInitialize() failed, client stopped !$n');
  End_If;

End_Task;
```

**Related Topics:**

- **Server** on page 288

# 11.9 User-Programmable CAN-PRIM Interface

**CAN-PRIM Interface**

The user-programmable CAN-PRIM interface allows to send and receive CAN messages. When using this feature, the CAN messages are completely processed by the application program.

**Applications**

The user-programmable CAN-PRIM interface can be used for the following applications:

- Connection of modules with CAN interface

**Required Programmer's Skills**

To be able to program user-programmable CAN-PRIM interfaces basic knowledge of Controller Area Networks (CAN) is required. This knowledge includes:

- Structure of CAN messages

**Contents**

## User-programmable CAN-PRIM interface - Operating Principle

**Operating Principle**

The user-programmable CAN-PRIM interface uses message boxes for data exchange between CAN bus and application program. Each message box is able to accomodate a complete CAN message.

32 message boxes are available to the user. Each of these boxes can be configured as inbox or outbox with a specific CAN ID.

**Technical Data**

| Function | Description |
|---|---|
| CAN ID | 11-bit or 29-bit |
| RTR messages | are not supported |
| Number of message boxes | 32 |

## Restrictions Regarding the CAN-PRIM Interface

**Only CAN-0**

The CAN-PRIM interface of a JVM-407 is available only with CAN-0.

**Time Response**

The interval between two CAN messages received via CAN-PRIM interface must be at least 10 ms. If the interval is shorter, the HMI JVM-407 is not able to receive all CAN messages.

**Earmarked CAN IDs**

The following CAN IDs are earmarked as CANopen® is running in parallel:

| Earmarked CAN IDs | Description |
|---|---|
| 0x00 | With 11-bit NMT |
| 0x600 + node ID and 0x580 + node ID | SDO |
| 0x80 | Sync |
| 0x100 | Time Stop |
| 0x80 + node ID | Emergency message |
| 0x700 + node ID | Heartbeat |
| + related PDOs | In the application project |
| + IDs of other CANopen® nodes | |

# Programming the CAN-PRIM Interface

**Overview of Registers**

The following registers are used in this manual:

| Register | Description |
|---|---|
| **R 200010500** | CAN-PRIM status |
| **R 200010501** | CAN-PRIM command register |
| **R 200010502** | Box number |
| **R 200010503** | FIFO buffer filling level |
| **R 200010510** | Box status |
| **R 200010511** | Box configuration |
| **R 200010512** | CAN ID |
| **R 200010513** | Number of data bytes |
| **R 200010514**<br>**...**<br>**R 200010521** | Data byte 0<br>...<br>Data byte 7 |

**Initialization**

To initialize the CAN-PRIM interface configure the length of the CAN ID for all message boxes as follows:

| If CAN ID length... | ... Then ... |
|---|---|
| is 11 bits | R 200010501 = 8; |
| is 29 bits | R 200010501 = 9; |

**Configuring a Message Box for Sending**

To configure a message box for sending proceed as follows:

| Step | Action |
|---|---|
| **1** | Select message box<br>R 200010502 := Message box number; |
| **2** | Configure the message box as inbox<br>R 200010511 := 1; |
| **3** | Configure the CAN ID for receiving messages<br>R 200010512 := CAN ID; |
| **4** | Activate the box<br>R 200010501 := 1;<br>**Result if configuration was successful:**<br>Bit 0 = 1 in R 200010510 |

**Sending a CAN Message**

To send a CAN message proceed as follows:

| Step | Action |
|---|---|
| **1** | Select message box<br>R 200010502 := Message box number; |
| **2** | Number of bytes to be sent<br>R 200010513 := Number of bytes; |
| **3** | Writing the data bytes<br>R 200010514 := Data byte 0;<br>R 200010515 := Data byte 1;<br>...<br>R 200010521 := Data byte 7; |
| **4** | Send data from the selected message box<br>R 200010501 := 3;<br>**Result if sending was successful:**<br>Bit 3 = 0 in R 200010510 |

**Configuring a Message Box for Receiving**

To configure a message box for receiving proceed as follows:

| Step | Action |
|---|---|
| **1** | Select message box<br>R 200010502 := Message box number; |
| **2** | Configure the message box as inbox<br>R 200010511 := 0; |
| **3** | Configure the CAN ID for receiving messages<br>R 200010512 := CAN ID; |
| **4** | Activate the box<br>R 200010501 := 1;<br>**Result if configuration was successful:**<br>Bit 0 = 1 in R 200010510 |

**Receiving a CAN Message**

To receive a CAN message proceed as follows:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Check bit 1 NEWDAT in R 200010500
<table>
<tr><th>If ...</th><th>... Then ...</th></tr>
<tr><td>Bit 1 = 1 in R 200010500</td><td>a CAN message has been received.<br>Proceed with step 2</td></tr>
</table></td></tr>
<tr><td>2</td><td>Select the message box which has received a CAN message.<br>R 200010502 := R 200010504;</td></tr>
<tr><td>3</td><td>Check the message box for overflow.
<table>
<tr><th>If ...</th><th>... Then ...</th></tr>
<tr><td>Bit 2 = 1 in R 200010510</td><td>an overflow has occurred.</td></tr>
</table></td></tr>
<tr><td>4</td><td>Read the number of received bytes<br>Number of bytes = R 200010513;</td></tr>
<tr><td>5</td><td>Read the received bytes<br>Data byte 0 = R 200010514;<br>Data byte 1 = R 200010515;<br>...<br>Data byte 7 = R 200010521;</td></tr>
<tr><td>6</td><td>Acknowledge that the message has been received<br>R 200010501 := 4;</td></tr>
<tr><td>7</td><td>The message box is again ready to receive.</td></tr>
</table>

## Internal Processes of the CAN-PRIM Interface

**Introduction**

The CAN-PRIM interface processes the following tasks independently:

- Reception of CAN messages
- Sending of CAN messages
- Filtering of CAN messages on reception

**Internal Reception of CAN Messages**

The CAN-PRIM interface receives new messages in the following way:

<table>
<tr><th>Stage</th><th>Description</th></tr>
<tr><td>1</td><td>The CAN bus receives a valid CAN message.</td></tr>
<tr><td>2</td><td>The CAN ID matches the receiving mask.</td></tr>
<tr><td>3</td><td>The CAN ID matches the CAN ID of a message box which has been configured as inbox.</td></tr>
<tr><td>4</td><td><table>
<tr><th>If in R 200010510 of the message box ...</th><th>... Then ...</th></tr>
<tr><td>the NEW DAT bit = 0</td><td>the NEW DAT bit switches to 1<br>proceed with stage 5</td></tr>
<tr><td>the NEW DAT bit = 1</td><td>the OVERRUN bit switches to 1;<br>CAN message data are discarded.</td></tr>
</table></td></tr>
<tr><td>5</td><td>R 200010503 FIFO filling level is incremented.</td></tr>
<tr><td>6</td><td>The message box number is entered into R 200010504 FIFO data.</td></tr>
<tr><td>7</td><td>In R 200010500 CAN-PRIM Status the NEW DAT bit is set to 1.</td></tr>
</table>

## Register Description - CAN-PRIM Interface

**R 200010500**

**CAN-PRIM status**

R 200010500 allows to evaluate the status of the CAN-PRIM interface.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 1** | **NEW-DAT** | |
| | 1 = | At least one message box has received a new CAN message. |
| **Bit 2** | **ID length** | |
| | 0 = | The length of sent/received CAN IDs is 11 bits |
| | 1 = | The length of sent/received CAN IDs is 29 bits |

| Module register properties | |
|---|---|
| Type of access | Read access |
| Value after reset | CAN-PRIM interface is enabled. |

**R 200010501**

## CAN-PRIM command register

R 200010501 is used to transfer certain commands to the CAN-PRIM interface.

| CAN-PRIM Interface - Commands | |
|---|---|
| **1** | **Enabling the message box**<br>The selected message box in R 200010502 is enabled. When enabling the message box, the system checks whether the CAN ID of the box is reserved or not.<br>Result: Bit 0 = 1 in R 200010510 |
| **2** | **Disabling the message box**<br>The selected message box in R 200010502 is disabled.<br>Result: Bit 0 = 0 in R 200010510 |
| **3** | **Sending CAN messages**<br>A CAN message is sent containing the data of the selected message box. |
| **4** | **Clearing the NEW DAT bit**<br>This command is for clearing the NEW DAT bit in R 200010500 which enables the selected message box to receive CAN messages again.<br>Result: Bit 1 = 0 in R 200010510 |
| **5** | **Clearing the OVERRUN bit**<br>This command is for clearing the OVERRUN bit in R 200010510 of the selected message box.<br>Result: Bit 2 = 0 in R 200010510 |
| **6** | **Clearing the transmission error bit**<br>This command is for clearing the transmission error bit in R 200010510 of the selected message box.<br>Result: Bit 3 = 0 in R 200010510 |
| **7** | **Clearing the FIFO buffer**<br>This command is for clearing all entries in the FIFO buffer.<br>Result: R 200010503 = 0 |
| **8** | **Setting the default ID length to 11 bits**<br>The ID length for all CAN messages is set to 11 bits.<br>Result:<br>Bit 2 = 0 in R 200010500<br>R 200010506 = 0<br>R 200010507 = 0 |
| **9** | **Setting the default ID length to 29 bits**<br>The ID length for all CAN messages is set to 29 bits.<br>Result:<br>Bit 2 = 1 in R 200010500<br>R 200010506 = 0<br>R 200010507 = 0 |

| CAN-PRIM Interface - Commands | |
|---|---|
| **10** | **Checking message boxes for new messages**<br>The CAN-PRIM interface automatically checks the inbox for new messages. Command 10 is for extending the interval between checks. |

| Module register properties | |
|---|---|
| Type of access | CAN-PRIM interface is enabled. |

## R 200010502 Message box number

R 200010502 is for selecting a message box. The data contained in the message box can then be accessed via module registers R 200010510 through R 200010521.

| Module register properties | | |
|---|---|---|
| Values | Message box number: | 0 ... 15 |
| Type of access | Read access removes character | |
| Takes effect | if the CAN-PRIM interface is enabled. | |

## R 200010503 FIFO buffer filling level

R 200010503 shows whether new CAN messages have been received, as well as the number of messages.

| Module register properties | | |
|---|---|---|
| Values | Number of received messages: | 0 ... 16 |
| Type of access | Read access | |
| Takes effect | if the CAN-PRIM interface is enabled. | |

**R 200010504**

## FIFO data

R 200010504 shows which of the messages boxes has received a new CAN message. Read access to R 200010504 removes the value which has been read last from the FIFO buffer. This access decrements the value of R 200010503 by one.

| **Module register properties** | | |
|---|---|---|
| Values | No FIFO data available: | -1 |
| | Number of the message box containing new data: | 0 ... 15 |
| Type of access | Read access removes characters | |
| Value after reset | -1 | |
| Takes effect | if the CAN-PRIM interface is enabled. | |

**R 200010506**

## Global receiving mask

The global receiving mask is for filtering the bits of the received CAN-ID. If the bit of the global receiving mask is set, the received bit of the CAN-ID is compared with the global receiving ID.

| **Module register properties** | | |
|---|---|---|
| Values | in the case of 11-bit CAN IDs | 0 ... 0x7FF |
| | in the case of 29-bit CAN IDs | 0 ... 0x1FFFFFFF |
| Bit = 0 | Bit is not compared with R 200010507. | |
| Bit = 1 | Bit is compared with R 200010507. | |
| Takes effect | if the CAN-PRIM interface is enabled. | |

**R 200010507**

## Global receiving ID

The global receiving ID and R 200010506 *Global receiving mask* are for setting a CAN ID range which is then forwarded to the CAN-PRIM interface.

| **Module register properties** | | |
|---|---|---|
| Values | in the case of 11-bit CAN IDs | 0 ... 0x7FF |
| | in the case of 29-bit CAN IDs | 0 ... 0x1FFFFFFF |
| Takes effect | if the CAN-PRIM interface is enabled. | |

**R 200010510**

## Box status

R 200010510 allows to evaluate the status of a message box.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Valid** | |
| | 1 = | The message box is enabled |
| **Bit 1** | **NEW-DAT** | |
| | 1 = | The message box has received a CAN message. Reception of additional CAN messages is blocked. |
| **Bit 2** | **OVERRUN** | |
| | 1 = | The message box has received a new CAN message while NEW-DAT was 1. |
| **Bit 3** | **Sending error** | |
| | 1 = | An error has occurred when sending a CAN message from this message box. |

| Module register properties | |
|---|---|
| Type of access | Read access |
| Takes effect | if the CAN-PRIM interface is enabled. |

**R 200010511**

## Box configuration

R 200010511 is for configuring the message box.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Outbox/inbox** | |
| | 0 = | Outbox |
| | 1 = | Inbox |

| Module register properties | |
|---|---|
| Takes effect | if the CAN-PRIM interface is enabled. |

**R 200010512**

## CAN ID

In the case of an outbox, a CAN message is sent using the CAN ID.
In the case of an inbox, only CAN messages with this CAN ID are received.

| **Module register properties** | | |
|---|---|---|
| Values | in the case of 11-bit CAN IDs | 0 ... 0x7FF |
| | in the case of 29-bit CAN IDs | 0 ... 0x1FFFFFFF |
| Takes effect | if the CAN-PRIM interface is enabled and the message box is disabled, i.e. if in MR 10510 bit 0 = 0. | |

**R 200010513**

## Number of data bytes

In the case of an outbox, a CAN message is sent with this number of data bytes.
In the case of an inbox, the number of received data bytes is entered.

| **Module register properties** | | |
|---|---|---|
| Values | Number of data bytes: | 0 ... 8 |
| Takes effect | if the CAN-PRIM interface is enabled. | |

**R 200010514 ...**
**R 200010521**

## Data bytes 0 through 7

In the case of an outbox, a CAN message is sent with these data bytes.
In the case of an inbox, the received data bytes are entered.

| **Module register properties** | | |
|---|---|---|
| Values | Data of data bytes: | 0 ... 255 |
| Takes effect | if the CAN-PRIM interface is enabled. | |

## CAN-PRIM Interface - Sample Program

**Task**

CAN messages with CAN IDs 0x200 are to be sent via CAN-PRIM interface. On receipt, a CAN message with CAN ID 0x277 is to be sent.

**Solution**

The data are sent and received via CAN-PRIM interface. To this end, a message box is configured as inbox for CAN ID 0x200. A second message box is configured as outbox with CAN ID 0x277.

**Configuration**

In this example, the CAN-PRIM interface of a JVM-407 is used.

**Configuring the JetSym STX Program**

```
Type
    TYPE_JC_CAN_PRIM:
    Struct
        State        : Int At 0*SizeOf(Int);
        Command      : Int At 1*SizeOf(Int);
        BoxNumber    : Int At 2*SizeOf(Int);
        FifoNumData  : Int At 3*SizeOf(Int);
        FifoData     : Int At 4*SizeOf(Int);
        GlobalMask   : Int At 6*SizeOf(Int);
        GlobalID     : Int At 7*SizeOf(Int);
        BoxState     : Int At 10*SizeOf(Int);
        BoxConfig    : Int At 11*SizeOf(Int);
        BoxCanId     : Int At 12*SizeOf(Int);
        BoxDLC       : Int At 13*SizeOf(Int);
        BoxData      : Array[8] of Int At 14*SizeOf(Int);
    End_Struct;
End_Type;

Var
    CanPrim          : TYPE_JC_CAN_PRIM At %VL 200010500;
    Data             : Array[8] of Int;
End_Var;
```

```
Task main Autorun

    // 11-bit CAN ID
    CanPrim.Command := 8;

    // Selecting box 0
    CanPrim.BoxNumber := 0;
    // Configuring the box for ID 0x200
    CanPrim.BoxCanId := 0x200;
    // Configuring box as inbox
    CanPrim.BoxConfig := 0;
    // Enabling the box
    CanPrim.Command := 1;
    If
        BitClear(CanPrim.BoxState, 0)
    Then
        // CAN ID already used by system bus
    End_If;

    // Selecting box 1
    CanPrim.BoxNumber := 1;
    // Configuring the box to ID 0x2FF
    CanPrim.BoxCanId := 0x2FF;
    // Configuring box as outbox
    CanPrim.BoxConfig := 1;
    // Enabling the box
    CanPrim.Command := 1;
    If
        BitClear(CanPrim.BoxState, 0)
    Then
        // CAN ID is already used by CAN system bus
    End_If;
End_Task;
```

**JetSym STX Program - Receiving Data**

```
// Waiting for new CAN messages
When
    BitSet(CanPrim.State, 1)
Continue;

// Reading box number out of FIFO buffer and selecting box
CanPrim.BoxNumber := CanPrim.FifoData;

// Checking for overrun
If
    BitSet(CanPrim.BoxState, 2)
Then
    // Acknowledging overrun
    CanPrim.Command := 5;
End_If;

// Copying received data
Data[0] := CanPrim.BoxData[0];
Data[1] := CanPrim.BoxData[1];

// Resetting the NEW-DATA bit to be able to receive
// new messages in this box
CanPrim.Command := 4;
```

**JetSym STX Program - Sending Data**

```
// Selecting box 1
CanPrim.BoxNumber := 1;

// Number of data bytes = 2
CanPrim.BoxDLC := 2;
// Entering the data to be sent
CanPrim.BoxData[0] := 12;
CanPrim.BoxData[1] := 25;

// Starting to send the CAN message
CanPrim.Command := 3;

// Checking for errors
If
    BitSet(CanPrim.BoxState, 3)
Then
    // Acknowledging errors
    CanPrim.Command := 6;
End_If;
```

# 12 Automatic Copying of Controller Data

**Introduction**

This chapter describes the AutoCopy feature which allows to copy data within the JVM-407 and/or between the JVM-407 and an FTP server. To this end, a command file has to be created which is then stored along with the data to the SD card or a USB stick. This command file is automatically processed by the controller during the boot process.

**Functions Within the Local File System**

The following functions can be performed:

- Storing registers and flags to a file
- Restoring registers and flags from a file
- Creating directories
- Deleting directories
- Copying files
- Deleting files

**Functions Within the File System of an FTP Server**

The following functions can be performed:

- Copying files from the FTP server
- Copying files to the FTP server
- Deleting files
- Changing directories
- Creating directories
- Deleting directories

**Areas of Application**

This function can be used in systems where remote maintenance is not feasible, no PC is available or the operator is not able (or should not be allowed) to make modifications to the plant. This function includes the following:

- Modification to the application program
- Modification to user data
- Modification to the controller configuration
- Operating system update (JVM-407, network nodes)
- Duplication of a control system

**Prerequisites**

The following requirements must be met:

- the programmer must be familiar with the file system of the JVM-407
- the programmer must have basic knowledge in the area of FTP application

**Names**

In this description "Complete Name" means the name of the file or directory including its complete path.

**Contents**

## 12.1 Operating Principle

**Introduction**

This chapter describes how the AutoCopy funcion is started and how it is executed by the JVM-407.

**Contents**

## Activating the AutoCopy Feature

**Introduction**

The AutoCopy function can only be executed while the JVM-407 is booting.

**Prerequisites**

The command file has been created and stored to the SD card or USB stick.

| | Value | Comment |
|---|---|---|
| File Name | autocopy.ini | All lower case letters |
| Directory - SD | /SD/ | Root directory on the SD Card |
| Directory - USB | /USB/ | Root directory on USB stick |

**Activating the AutoCopy Feature**

To start AutoCopy proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the device OFF. |
| **2** | Insert the SD card completely into the SD slot or insert the USB stick into the USB port. |
| **3** | Keep the keys F1 and F3 pressed. |
| **4** | Switch the device ON. |
| **5** | Wait until the following message appears: **Start operating system in STOP mode**. |

**Result:** The device is booting in AutoCopy mode.

# Executing AutoCopy Commands

**Introduction**

During the boot process in AutoCopy mode the device executes the commands contained in the command file.

**Restrictions**

In AutoCopy mode, the following restrictions apply as regards the functions of the device JVM-407:

- The application program is not executed
- No communication with the JVM-407 possible

**Executing AutoCopy Commands**

When executing AutoCopy commands, the OS of the JVM-407 proceeds as follows:

| Stage | Description |
|---|---|
| **1** | The device loads the file "/SD/autocopy.ini" from the SD card or from the USB stick. |
| **2** | It reads the values from section [OPTIONS] |
| **3** | The device 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 device processes the other commands in ascending order up to the number given in section [OPTIONS] |
| **n+1** | The device calculates the statistic values for all command results and writes them into the log file. |

# Terminating AutoCopy Mode

**Introduction**

The AutoCopy mode can only be exited by booting the JVM-407.

**Terminating AutoCopy Mode**

Once the AutoCopy function is completed, proceed as follows to exit the AutoCopy mode:

| Step | Action |
|---|---|
| **1** | Remove the SD card or the USB stick. |
| **2** | Press any key on the device. |

**Result:** The device reboots.

# 12.2 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 values can be set which are then used by the AutoCopy function.
- Blank lines can be inserted as required
- The following characters precede a comment line: "!", "#" or ";"

**Sections**

The command file has two section types:

- In section [OPTIONS] the basic settings are made. It exists only once.
- In the sections [COMMAND_#] the commands to be executed are specified. The number of commands is limited to 128.

**Contents**

## Section [OPTIONS]

**Introduction**

This section contains the basic settings of the AutoCopy function. It exists only once, preferably at the beginning of the file.

**Example**

```
[OPTIONS]
CommandCount = 14
LogFile      = /SD/autocopy.log
LogAppend    = 1
```

**Elements of this Section**

This 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 | /SD/autocopy.log |
| Description | Complete name of the log file |
| Allowed values | ■ All allowed file names<br>■ Directory exists |
| Illegal values | ■ Invalid file name<br>■ Nonexistent directory |
| In case of illegal value or missing entry | No log file will be created. |
| **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.<br>■ 1 = Append file to existing one. If no file exists, a new log file is created. |
| Illegal values | ■ < 0<br>■ > 1 |
| In case of illegal value or missing entry | A new log file will be created. |

## Command Sections

**Introduction**

In these sections commands can be specified which are then executed by the AutoCopy function of the JVM-407.

**Example**

```
[COMMAND_1]
Command = DirCreate
Path    = /Homepage
ErrorAsWarning = 1

[COMMAND_2]
Command     = FileCopy
Source      = /SD/Index.htm
Destination = /Homepage/index.htm

[COMMAND_3]
Command        = FtpConnect
ServerAddr     = 192.168.123.45
UserName       = admin
Password       = admin
```

**Section Names**

The section names consist of the string `COMMAND_` followed by a number which indicates the number of the entry `CommandCount` given in 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 hold only one command. That is, for each command a separate section has to be created.

**Troubleshooting**

When an error occurs while a command is being processed, the corresponding entry in the log file is made. For each command the user can set, whether the error is entered into the log file as `Error` or as `Warning`. This setting is made through the optional parameter `ErrorAsWarning`:

| ErrorAsWarning | Entry in 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"`).
- The function parameter for the file on the FTP server may contain the path to this file if this feature is supported by the file system. If this feature is not supported, the corresponding directory must be set using the command `FtpDirChange(...)`.
- The file system of a JVM-407 PLC supports both options.

**Available Commands in the Local File System**

The following commands are available for access to the local file system:

**Command = DirCreate**

| | |
|---|---|
| Function | This command is for creating a subdirectory |
| Parameter name | Path |
| Parameter value | Complete name of the directory |
| Allowed values | ■ All valid directory names<br>■ Existing higher-level directories |
| Illegal values | ■ Invalid directory names<br>■ Nonexistent higher-level directory<br>■ Name of an already existing directory |
| In case of an illegal value | The directory will not be created and the error message will be entered into the log file |
| Example | `[COMMAND_1]`<br>`Command = DirCreate`<br>`Path    = /sub1`<br><br>`[COMMAND_2]`<br>`Command = DirCreate`<br>`Path    = /sub1/sub2` |

**Command = DirRemove**

| | |
|---|---|
| Function | This command is for deleting a subdirectory |
| Parameter name | Path |
| Parameter value | Complete name of the directory |
| Allowed values | ■ All valid directory names<br>■ An empty directory |
| Illegal values | ■ Invalid directory names<br>■ Directory is not empty |
| In case of an illegal value | The directory will not be deleted and the error message will be entered into the log file |
| Example | `[COMMAND_8]`<br>`Command = DirRemove`<br>`Path    = /sub1/sub2` |

**Command = FileCopy**

| | |
|---|---|
| Function | 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<br>■ The destination directory does exist |
| Illegal values | ■ Invalid file name<br>■ Nonexistent source file<br>■ Nonexistent destination directory |
| In case of an illegal value | The file will not be copied and the error message will be entered into the log file |
| Example | `[COMMAND_1]`<br>`Command     = FileCopy`<br>`Source      = /SD/OS/JC-340_1.04.0.03.os`<br>`Destination = /System/OS/op_system.os`<br><br>`[COMMAND_2]`<br>`Command     = FileCopy`<br>`Source      = /SD/Manual.pdf`<br>`Destination = /sub1/Manual.pdf` |

**Command = FileRemove**

| | |
|---|---|
| Function | This command is for deleting a file. |
| Parameter name | Path |
| Parameter value | Complete name of the file |
| Allowed values | All allowed file names |
| Illegal values | Invalid file name |
| In case of an illegal value | The file will not be deleted and the error message will be entered into the log file |
| Example | `[COMMAND_5]`<br>`Command = FileRemove`<br>`Path    = /sub1/Manual.pdf` |

**Command = DaFileRead**

| | |
|---|---|
| Function | This command is for transferring register values and flag states from a data file to the JVM-407 |
| Parameter name | DaFile |
| Parameter value | Complete name of the data file |
| Allowed values | All allowed file names for data files |
| Illegal values | ■ Invalid file name<br>■ Nonexistent data file |
| In case of an illegal value | The date will not be transferred to the controller and the error message will be entered into the log file |
| Example | `[COMMAND_12]`<br>`Command = DaFileRead`<br>`DaFile      = /SD/Data/MyTestData.da` |

**Command = DaFileWrite**

| | |
|---|---|
| Function | 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 file |
| Allowed values | ■ All allowed file names for data files<br>■ The destination directory does exist |
| Illegal values | ■ Invalid file name<br>■ Nonexistent destination directory |
| In case of an illegal value | The file will not be created and the error message will be entered 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 file which may exist and create a new one.<br>■ 1 = Append file to existing one. If no file exists, create a new data file. |
| Illegal values | ■ < 0<br>■ > 1 |
| In case 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 | ■ Registers<br>■ Flag |
| Illegal values | Values other than "Register" or "Flag" |
| In case of an illegal value | The file will not be created and the error message will be entered 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 JVM-407 |
| Illegal values | Invalid numbers |
| In case of an illegal value | The file will not be created and the error message will be entered 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 JVM-407 which are equal to or greater than the value for "First". |
| Illegal values | ■ Invalid numbers<br>■ Numbers less than "First" |
| In case of an illegal value | Only one value (First) is stored |

| | |
|---|---|
| Example | `[COMMAND_11]`<br>`Command = DaFileWrite`<br>`DaFile      = /SD/MyTestData2.da`<br>`Append      = 0`<br>`Type        = Register`<br>`First       = 1000000`<br>`Last        = 1000000`<br><br>`[COMMAND_12]`<br>`Command = DaFileWrite`<br>`DaFile      = /SD/MyTestData2.da`<br>`Append      = 1`<br>`Type        = Flag`<br>`First       = 10`<br>`Last        = 20`<br><br>`[COMMAND_13]`<br>`Command = DaFileWrite`<br>`DaFile      = /SD/MyTestData2.da`<br>`Append      = 1`<br>`Type        = Register`<br>`First       = 1000001`<br>`Last        = 1000999` |

**Available Commands for Access via FTP**

The following commands are available for access via network using FTP:

| **Command = FtpConnect** | |
|---|---|
| Function | 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<br>▪ Name which can be resolved through DNS |
| Illegal values | ▪ IP address other than tat of the FTP server<br>▪ 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 | Connection will not be established and the error message will be entered into the log file |
| Example | `[COMMAND_1]`<br>`Command = FtpConnect`<br>`ServerAddr = 192.168.123.45`<br>`UserName   = admin`<br>`Password   = admin` |
| Comment | Only one connection with an FTP server can be established at a time. If a connection to another FTP server is to be established, the JVM-407 terminates the existing connection beforehand. |

| **Command = FtpFileRead** | |
|---|---|
| Function | 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<br>■ The destination directory does exist |
| Illegal values | ■ Invalid file name<br>■ Nonexistent source file<br>■ Nonexistent destination directory |
| In case of an illegal value | The file will not be copied and the error message will be entered into the log file |
| Example | `[COMMAND_8]`<br>`Command      = FtpFileRead`<br>`ServerFile   = /app/cantest/cantest.es3`<br>`ClientFile   = /SD/cantest3.es` |

| **Command = FtpFileWrite** | |
|---|---|
| Function | Copying 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<br>■ The destination directory does exist |
| Illegal values | ■ Invalid file name<br>■ Nonexistent source file<br>■ Nonexistent destination directory |
| In case of an illegal value | The file will not be copied and the error message will be entered into the log file |
| Example | `[COMMAND_5]`<br>`Command     = FtpFileWrite`<br>`ServerFile  = /System/OS/op_system.os`<br>`ClientFile  = /SD/OS/JC-340_1.09.0.00.os` |

| **Command = FtpFileRemove** | |
|---|---|
| Function | This command is for deleting a file in the FTP server |
| Parameter name | ServerFile |
| Parameter value | Complete name of the file |
| Allowed values | All allowed file names |
| Illegal values | Invalid file name |
| In case of an illegal value | The file will not be deleted and the error message will be entered into the log file |

| | |
|---|---|
| Example | `[COMMAND_9]`<br>`Command    = FtpFileRemove`<br>`ServerFile = /sub1/Manual.pdf` |

**Command = FtpDirChange**

| | |
|---|---|
| Function | Changing the working directory in FTP server |
| Parameter name | ServerDir |
| Parameter value | Complete name of the directory |
| Allowed values | All valid directory names |
| Illegal values | Invalid directory names |
| In case of an illegal value | The directory will not be changed and the error message will be entered into the log file |
| Example | `[COMMAND_12]`<br>`Command = FtpDirChange`<br>`ServerDir   = /Data/MyTestData` |

**Commando = FtpDirCreate**

| | |
|---|---|
| Function | This command is for creating a subdirectory in FTP server |
| Parameter name | ServerDir |
| Parameter value | Complete name of the directory |
| Allowed values | ■ All valid directory names<br>■ Existing higher-level directories |
| Illegal values | ■ Invalid directory names<br>■ Nonexistent higher-level directory<br>■ Name of an already existing directory |
| In case of an illegal value | The directory will not be created and the error message will be entered into the log file |
| Example | `[COMMAND_6]`<br>`Command     = FtpDirCreate`<br>`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. |

**Commando = FtpDirRemove**

| | |
|---|---|
| Function | This command is for removing a subdirectory in FTP server |
| Parameter name | ServerDir |
| Parameter value | Complete name of the directory |
| Allowed values | ■ All valid directory names<br>■ An empty directory |
| Illegal values | ■ Invalid directory names<br>■ Directory is not empty |
| In case of an illegal value | The directory will not be removed and the error message will be entered into the log file |

Example

```
[COMMAND_8]
Command     = FtpDirRemove
ServerDir   = /Data/MyTestData
```

## Example of a Command File

**Task**

New functions are to be added to an installed JVM-407. To this end, the following modifications have to be made to the configuration:

- Operating system update
- New application program
- New values for some of the registers

**Solution**

The required files are copied to an SD card and a command file for the AutoCopy function is created. This SD card along with a short instruction sheet is sent to the customer. Once the update is completed, the customer returns the card.

**SD Card Contents**

The SD card contains the following files:

- The file "autocopy.ini"
- The new OS
- A .da file containing the new register values
- A file "start.ini" and a .es3 file containing the new application program

Following execution the log file "autocopy.log" has been added.

**Command File**

```
[OPTIONS]
CommandCount = 6
LogFile      = /SD/autocopy.log
LogAppend    = 0

# update operating system of JVM-407
[COMMAND_1]
Command     = FileCopy
Source      = /SD/OS/JVM4xx_1.15.1.00.os
Destination = /System/OS/op_system.os

# Creating user program directories
# Probably already present - but to be sure ...
[COMMAND_2]
Command        = DirCreate
Path           = /app
ErrorAsWarning = 1

[COMMAND_3]
Command     = DirCreate
Path        = /app/userprogtest

# Copying user program start file
[COMMAND_4]
Command     = FileCopy
Source      = /SD/UserProgs/start.ini
```

```
Destination = /app/start.ini

# Copying user program
[COMMAND_5]
Command     = FileCopy
Source      = /SD/UserProgs/userprogtest.es3
Destination = /app/userprogtest/userprogtest.es3

# Setting registers and flags
[COMMAND_6]
Command     = DaFileRead
DaFile      = /SD/UserData/MyTestData.da
```

# 12.3 Log File

**Introduction**

This chapter covers the structure and contents of the log file into which the results of each command are entered.

**Contents**

## File Contents

**Introduction**

The log file is a plain text file. The command file defines whether a log file is to be created. And whether it is to be created from scratch or whether the entries are to be appended to an existing log file.

**Example**

```
JetControl AutoCopy log file 07.11.2008 09:14:09

 1: Ok    - FileCopy   /SD/OS/JC-340_1.04.0.00.os
                         /System/OS/op_system.os (345740 byte)
 2: Warning - DirCreate  /app
 3: Ok    - DirCreate  /app/userprogtest
 4: Ok    - FileCopy   /SD/UserProgs/start.ini
                         /app/start.ini (63 byte)
 5: Ok    - FileCopy   /SD/UserProgs/userprogtest.es3
                         /app/userprogtest/userprogtest.es3
                         (169 byte)
 6: Error   - DaFileRead /SD/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, short statistics on command processing.

In the above example, an error message occurs (which will be entered as warning) when trying to create the directory "/app" as this directory already exists. When reading the DA file an error occurs, too. The corresponding error message is entered into the log file.

# 12.4 Data Files

**Introduction**

This chapter covers data files where register and flag values are stored.

**Contents**

## File Format

**Format**

The file is structured as follows:

- 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 registers |
| QS | Floating-point registers |

**Example**

```
SD1001
; Data File - Jetter AG
;
; Register 1000000 ... 1000005
RS     1000000    12345
RS     1000001    2
RS     1000002    -1062729008
RS     1000003    502
RS     1000004    50
RS     1000005    3
QS     1009000    3.14
;
; Flag 10 ... 13
FS     10     0
FS     11     1
FS     12     1
FS     13     0
```

# 13 Operating System Update

**Introduction**

Jetter AG are continuously striving to enhance the operating systems for HMIs. Enhancing means adding new features, upgrading existing functions and fixing bugs.

This chapter describes how to update the operating system.

**Downloading an Operating System**

You can download operating systems from the Jetter AG homepage at **www.jetter.de http://www.jetter.de**. You get to the OS files by clicking on the quick link "Operating System Download" located on the website of the corresponding HMI.

**Contents**

# 13.1 Updating the Operating System of the HMI

**Introduction**

This chapter describes how an OS update of the JVM-407 is carried out. There are several options to transfer the OS file to the device:

- from within the programming tool JetSym
- via FTP connection
- from an SD Card
- from a USB stick
- from within the application program

**Contents**

## Operating System Update from within JetSym

**Introduction**

The programming tool JetSym offers an easy way to transfer an OS file to the JVM-407.

**Prerequisites**

- An OS file for the JVM-407 must be available.
- There must be a UDP/IP and a TCP/IP connection between programming tool and IP port of the JVM-407. The number of this port must have been entered into the configuration memory as IP basic port number.
- During booting, the JVM-407 is waiting for the OS update, or the OS is already running.
- Make sure that the JVM-407 is not switched off during OS update.

**Updating the Operating System**

To update the OS proceed as follows:

| Step | Action |
|---|---|
| 1 | In JetSym, call up the "Build" menu and select item "Update OS..." there, or click in the configuration window of the hardware manager on "OS Update".<br>**Result:** The file selection box opens. |
| 2 | Select the desired OS file here.<br>**Result:** In JetSym, a confirmation box opens. |
| 3 | Start the OS upload by clicking the button "Yes". |
| 4 | Wait until the update process is completed. |
| 5 | Reboot the JVM-407 to launch the new operating system. |

## Operating System Update by Means of FTP

**Introduction**

Using an FTP client an OS file can be transferred to the controller.

**Prerequisites**

- An OS file for the controller JVM-407 must be available.
- An FTP connection to the controller must be possible.
- The login parameters for a user with administrator or system rights must be at hand.
- The operating system is running.
- Make sure that the controller is not switched off during OS update.

**Updating the Operating System**

To update the OS proceed as follows:

| Step | Action |
|---|---|
| 1 | Establish an FTP connection to the controller. |
| 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 | Reboot the controller to launch the new operating system. |

## Automatic OS Update from SD Card and USB stick

**Reference:** An automatic OS update of the HMI from SD card or USB stick can be carried out using the AutoCopy function. For a detailed description refer to *AutoCopy* on page 312.

# 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 controller from within an OS file.

**Prerequisites**

- An OS file for the controller JVM-407 must be available in the file system of the controller.
- The operating system of the controller and the application program are running.
- Make sure that the controller is not switched off during OS update.

**Updating the Operating System**

To update the OS from within 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 | Reboot the controller to launch the uploaded operating system (for example by entering a value into the system command register). |

**Sample Program**

```
Var
    SourceName:             String[100];
    DestinationName:        String[100];
    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;
```

# 14 Application Program

**Introduction**

This chapter explains how the application program is stored to the JVM-407 and how the user selects the program 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 JVM-407 file system.

**Contents**

## Loading an Application Program

**Introduction**

The application program is loaded and executed by the file system either on relaunch of the application program through JetSym or on re-boot of the JVM-407.

**Loading Process**

The application program is loaded by the JVM-407's OS as follows:

| Stage | Description |
|---|---|
| **1** | The OS reads the file "/app/start.ini" from the internal flash disk. |
| **2** | The OS reads out the path to the application program from the entry "Project". |
| **3** | The OS reads out the program name from the entry "Program". The path is relative to the directory "/app". |
| **4** | The OS loads the application program from the file <Project>/<Program>. |

# Application Program - Default Path

**Introduction**

When uploading the application program from JetSym to the JVM-407, it is stored as file to the internal flash disk. Path and file name are entered 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 "*.es3" to it. Path and file names are always converted into lower case letters.

**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. This path is relative to "/app". |
| Program | Name of the application program file |

**Example:**

```
[Startup]
Project = test_program
Program = test_program.es3
```

**Result:** The application program is loaded from the file "/app/test_program/test_program.es3".

**Related Topics**

- **Storing the Application Program to the SD Card** on page 346

## Storing the Application Program to an SD Card

**Introduction**

When uploading the application program from JetSym to the JVM-407, the default path for the application program is used. If the application program is to be read from the SD card or an USB stick, the user has to configure this option.

The procedure is the same if you wish to store the application program to a different directory of the internal flash disk.

**Prerequisites**

Since the JVM-407'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 SD card or the USB stick**

This is how the JVM-407 is to be configured if you wish to store the application program to the SD card:

| Step | Action |
|---|---|
| 1 | Create the desired directory on the SD card or the USB stick. |
| 2 | Store the application program created by JetSym to this directory. |
| 3 | Enter the path to the application program file and the program name into the file "/app/start.ini" on the controller's internal flash disk. |

**Result:** When the application program is relaunched, it is loaded from SD card or USB stick.

**File "/app/start.ini"**

This file is a text file with one section holding two entries:

| Entry | Description |
|---|---|
| [Startup] | Section name |
| Project | Path to the application program. This path is relative to "/app". |
| Program | Name of the application program file |

**Example - SD Memory Card**

```
[Startup]
Project = /SD/TestProgram
Program = Test1.es3
```

**Example - USB Stick**

```
[Startup]
Project = /USB/TestProgram
Program = Test1.es3
```

**Result:** The application program is loaded from the file "Test1.es3" located in the directory "TestProgram" on SD card ("/SD/TestProgram/Test1.es3") and on USB stick ("/USB/TestProgram/Test1.es3").

**Related Topics:**


- **Application Program - Default Path** on page 345

# 15 Quick Reference JVM-407

## OS version

This quick reference gives an overview of registers and flags used in connection with HMIs JVM-407, BTM 07, BTM 09, BTM 09V and BTM 012 with OS version 1.17.1.00.

## General Overview - Registers

| | |
|---|---|
| 100000 ... 100999 | Electronic Data Sheet (EDS) |
| 101000 … 101999 | Configuration |
| 102000 … 102999 | Real-Time Clock (RTC) |
| 104000 ... 104999 | Ethernet |
| 106000 ... 106999 | CAN |
| 107000 ... 107999 | SD Memory Card |
| 108000 ... 108999 | CPU/backplane |
| 200000 ... 209999 | General system registers |
| 210000 ... 219999 | Application program |
| 230000 ... 239999 | Networking via JetIP |
| 260000 ... 269999 | Remote scan |
| 270000 ... 279999 | Modbus/TCP |
| 290000 ... 299999 | E-mail |
| 310000 ... 319999 | File system / data files |
| 350000 ... 359999 | User-programmable IP Interface |
| 360000 ... 369999 | Display |
| 1000000 ... 1059999 | JC-360: Application registers (remanent; Int/Float) |

## General Overview - I/Os

| | |
|---|---|
| **Entry keys** | |
| 361000 ... 361007 | Bit-coded mapping of entry keys |
| **LED** | |
| 362000 ... 362006 | Bit-coded mapping of LEDs |
| **I/Os** | |
| 362100 | Bit-coded mapping of status LEDs |
| 362200 | Bit-coded mapping of relay |

## General Overview - Flags

| | |
|---|---|
| 0 ... 255 | Application flags (remanent) |
| 256 ... 2047 | overlaid by registers 1000000 through 1000055 |
| 2048 ... 2303 | Special Flags |

## MAC Address

| | |
|---|---|
| 100801 | MAC Address (Jetter) |
| 100802 | MAC Address (device) |

## Configuration

| | |
|---|---|
| | **From file /system/ config.ini** |
| 101100 | IP address |
| 101101 | Subnet mask |
| 101102 | Default gateway |
| 101103 | DNS server |
| 101132 | Host name suffix type |
| 101133 ... 101151 | Host name (register string) |
| 101164 | JetIP port number |
| 101165 | STX debugger port number |
| | **Used by the system** |
| 101200 | IP address |
| 101201 | Subnet mask |
| 101202 | Default gateway |
| 101203 | DNS server |
| 101232 | Host name suffix type |
| 101233 … 101251 | Host name (register string) |
| 101264 | JetIP port number |
| 101265 | STX debugger port number |

## Realtime clock (RTC)

| | |
|---|---|
| | **Direct access** |
| 102911 | Seconds |
| 102912 | Minutes |
| 102913 | Hours |
| 102914 | Weekday (0 = Sunday) |
| 102915 | Day |
| 102916 | Month |
| 102917 | Year |
| | **Buffer access** |
| 102921 | Seconds |
| 102922 | Minutes |
| 102923 | Hours |
| 102924 | Weekday (0 = Sunday) |
| 102925 | Day |
| 102926 | Month |
| 102927 | Year |
| 102928 | Read/write trigger |

## Ethernet

| | |
|---|---|
| | **IP** |
| 104531 | current IP address (rw) |
| 104532 | current subnet mask (rw) |
| 104533 | current default gateway (rw) |

## CAN

| | |
|---|---|
| 106000 | Baud rate CAN 0 |
| 106001 | Node ID CAN 0 |
| 106100 | Baud rate CAN 1 |
| 106101 | Node ID CAN 1 |
| 106200 | Baud rate CAN 2 |
| 106201 | Node ID CAN 2 |

## SD memory card

| | |
|---|---|
| 107000 | Bit 0 = 1: Card available<br>Bit 1 = 1: Card ready |
| 107001 | 1 = card is read-only<br>(only applies if reg. 107000 = 3) |
| 107002 | Size in MBytes |
| 107003 | Baud rate in MBaud |

## CPU Hardware

| | |
|---|---|
| 108015 | Backup voltage (e.g. of the clock)<br>0 = Data invalid<br>1 = Power supply OK<br>You can confirm the register by entering 1, if the power supply has been recovered. |

## USB Data Carrier

| | |
|---|---|
| 109000 | Bit 0 = 1: Data carrier exists<br>Bit 1 = 1: Data carrier is ready |
| 107001 | 1 = data carrier is read-only<br>(only applies if reg. 109000 = 3) |
| 107002 | Size in MBytes |

## CPU

| | |
|---|---|
| 108002 | all LED on/off (bit-coded)<br>Bit 1: LED E |
| 108004 | LED E<br>0 = off<br>3 = on |
| 108015 | Application status<br>2 = RUN<br>3 = STOP |

## General System Registers

| | |
|---|---|
| 200000 | OS version (Major * 100 + Minor) |
| 200001 | Application program running (Bit 0 = 1) |
| 200008 | Error register (identical to 210004) |
| 200168 | Bootloader version (IP format) |
| 200169 | OS version (IP format) |
| 201000 | Runtime registers in milliseconds (rw) |
| 201001 | Runtime registers in seconds (rw) |
| 201002 | Runtime registers in reg. 201003 Units (rw) |
| 201003 | * 10 ms units for reg. 201002 (rw) |
| 201004 | Runtime registers in milliseconds (ro) |
| 202930 | Web status (bit-coded)<br>Bit 0 = 1: FTP server available<br>Bit 1 = 1: HTTP server available<br>Bit 2 = 1: E-mail available<br>Bit 3 = 1: Data file function available<br>Bit 4 = 1: Modbus/TCP has been licensed<br>Bit 5 = 1: Modbus/TCP available<br>Bit 6 = 1: Ethernet/IP available |
| 202936 | Control register file system<br>0xc4697a4b: Formatting the Flash Disk<br>0xd364e64d: Formatting the SD Card<br>0x2c9b3c94: Checking the SD Card<br>0x8f3d5185: Formatting the USB data carrier<br>0x17dbd42a: Checking the USB data carrier |
| 202960 | Password for system command register (0x424f6f74) |
| 202961 | System Command Registers |
| 202980 | Error history: Number of Entries |
| 202981 | Error history: Index |
| 202982 | Error history: Item |
| 203000 | Interface Monitoring: JetIP |
| 203001 | Interface Monitoring: SER |
| 203005 | Interface Monitoring: Debug server |
| 203100 ... 203107 | 32-bit overlay - Flag 0 ... 255 |
| 203108 ... 203123 | 16-bit overlay - Flag 0 ... 255 |
| 203124 ... 203131 | 32-bit overlay - Flag 2048 ... 2303 |
| 203132 ... 203147 | 16-bit overlay - Flag 2048 ... 2303 |
| 209700 | System logger: global enable |
| 209701 ... 209739 | Enable system components |

## Application Program

| | |
|---|---|
| 210000 | Application program running (Bit 0 = 1) |
| 210001 | JetVM version |
| 210004 | Error register (bit-coded)<br>Bit 1: Error JX3 bus<br>Bit 2: Error JX2 bus<br>Bit 8: illegal jump<br>Bit 9: illegal call<br>Bit 10: illegal index<br>Bit 11: illegal Opcode<br>Bit 12: divide by 0<br>Bit 13: stack overflow<br>Bit 14: stack underflow<br>Bit 15: stack invalid<br>Bit 16: Error when loading the application program<br>Bit 24: Cycle time overrun<br>Bit 25: Tasklock timeout<br>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 count within an execution unit |
| 210051 | ID of the execution unit just processed |
| 210056 | Required total cycle time in µs |
| 210057 | Calculated total cycle time in µs |
| 210058 | Maximum time slice per task in µs |
| 210060 | Task ID (for reg. 210061) |
| 210061 | Task priority for task [reg. 210060] |
| 210063 | Length of Scheduler Table |
| 210064 | Index in Scheduler Table |
| 210065 | Task ID in Scheduler Table |
| 210070 | Task ID (for reg. 210071) |
| 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 reg. 210077) |
| 210077 | Timer value in milliseconds |
| 210100 ... 210199 | Task status |
| 210400 ... 210499 | Task program address |
| 210600 | Task ID of a cyclic task (for reg. 210601) |
| 210601 | Processing time for a cyclic task in 1/10 of a percent |
| 210609 | Tasklock timeout in ms<br>-1: Monitoring disabled |
| 210610 | Time overrun (bit-coded, Bit 0 -> Timer 0 etc.) |

## Networking via JetIP

| | |
|---|---|
| 230000 | JetIP/TCP Server: Number of open connections |
| 230001 | JetIP/TCP Server: Mode |
| 230002 | JetIP/TCP Server: Time |
| 232708 | Timeout in milliseconds |
| 232709 | Response time in milliseconds |
| 232710 | Number of network errors |
| 232711 | Error code of the last access<br>0 = No error<br>1 = Timeout<br>3 = Error message of the remote station<br>5 = Invalid network address<br>6 = Invalid number of registers<br>7 = Invalid interface number |
| 232717 | Max. number of retries |
| 232718 | Number of retries |

## Modbus/TCP

| | |
|---|---|
| 272702 | Register offset |
| 272704 | Input offset |
| 272705 | Output offset |
| 278000 ... 278999 | 16-bit I/O register; overlaid by virtual I/O 20001 through 36000 |

## E-mail

| | |
|---|---|
| 292932 | IP address of SMTP server |
| 292933 | IP address of POP3 server |
| 292934 | Port number of SMTP server |
| 292935 | Port Number of POP3 server |
| 292937 | Status of E-Mail Processing |
| 292938 | E-Mail Task ID |

## File system / data file function

| | |
|---|---|
| 312977 | Status of file operation |
| 312978 | Task ID |

## User-Programmable IP Interface

| | **Reading the IP-PRIM connections 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 status |
| 350006 | Number of bytes sent |
| 350007 | Number of bytes received |

## Application Registers

| | |
|---|---|
| 1000000 ... 1005999 | 32 bit integer (remanent) |

## CAN-PRIM register

| | |
|---|---|
| 200010500 | CAN-PRIM status |
| 200010501 | CAN-PRIM command register |
| 200010502 | Message box number |
| 200010503 | FIFO level |
| 200010504 | FIFO data |
| 200010506 | Global receiving mask |
| 200010507 | Global receiving ID |
| 200010510 | Box status |
| 200010511 | Box configuration |
| 200010512 | CAN ID |
| 200010513 | Number of data bytes |
| 200010514 ... 200010521 | Data bytes |

## Display

**Function keys**

| | |
|---|---|
| 361000 ... 361007 | Bit-coded mapping of function keys<br>e.g. bit 0: 1 = key 1 is pressed |

**Ignition (IGN)**

| | |
|---|---|
| 361100 | Bit 0:<br>0 = Ignition switched on<br>1 = Ignition switched off |

**LEDs for keys**

| | |
|---|---|
| 362000 ... 362006 | Bit-coded mapping of LEDs<br>e.g. bit 0: 1 = LED key 1 on |

**I/O (IN1 ... IN15 and OUT)**

| | |
|---|---|
| 362100 | Bit-coded mapping of status LEDs<br>e.g. bit 0: 1 = IN1 on |
| 362200 | Bit-coded mapping of output<br>e.g. bit 0: 1 = OUT on |

**Digipot**

| | |
|---|---|
| 363000 | Current count value |
| 363001 | Digipot - Enter function |
| 363002 | Minimum count value |
| 363003 | Maximum count value |

**Display**

| | |
|---|---|
| 364000 | Backlighting |
| 364001 | Keys night-lighting |
| 364003 | Brightness sensor |

**Video**

is displayed by default on object 14000 (rectangle)

| | |
|---|---|
| 364200 | Video input (Input) |
| 364201 | Video input external Mux (only BTM 07) |
| 364202 | Video type<br>1 = composite<br>2 = svideo |
| 364203 | Video format<br>1 = PAL<br>2 = NTSC |
| 364204 | Video Options<br>Bit 0: 1 = Interlaced<br>Bit 1: 1 = Mirror vertical |
| 364210 | Video input brightness |
| 364211 | Video input contrast |
| 364212 | Video input saturation |
| 364220 | Video Output ID (Rectangle ID in IOP) |
| 364230 | Video Input Source X |
| 364231 | Video Input Source Y |
| 364232 | Video Input Source Width |
| 364233 | Video Input Source Height |

**Visualization**

| | |
|---|---|
| 365000 ... 365029 | Name of IOP file |
| 365050 ... 365079 | Name of language |
| 365100 | Language selection according to ID |
| 365200 | Number of available languages |
| 365201 | Current selection for the Info Register |
| 365202 | Info Register Default ID = 1 |
| 365203 | Info Register size of IOP file |
| 365210 ... 365240 | Info Register file name of IOP file |
| 365260 ... 365289 | Info Register name of language |

**System status**

| | |
|---|---|
| 367000 | HAL name |
| 367010 | Backup battery / Battery full (> 2 V) |

## Special Flags - Network

| | |
|---|---|
| 2075 | Error in networking via JetIP |

## Special flags - interface monitoring

| | |
|---|---|
| 2088 | OS flag - JetIP |
| 2089 | User flag - JetIP |
| 2090 | OS flag - SER |
| 2091 | User flag - SER |
| 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 User 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 Functions

| | |
|---|---|
| 4 | BCD to HEX conversion |
| 5 | HEX to BCD conversion |
| 20 | Square Root |
| 21 | Sine |
| 22 | Cosine |
| 23 | Tangent |
| 24 | Arc Sin |
| 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 |
| 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 |
| 90 | Writing data file |
| 91 | Appending data file |
| 92 | Reading data file |
| 96 | Deleting data file |
| 110 | E-mail feature |
| 150 | Configuring NetCopyList |
| 151 | Deleting NetCopyList |
| 152 | Sending NetCopyList |

# Appendix

**Introduction**

This appendix contains electrical and mechanical data, as well as operating data.

**Contents**

# A: Technical Data

**Introduction**

This chapter contains information on electrical and mechanical data, as well as on operating data of the JVM-407.

**Contents**

# Technical Data

## Technical Data - Electrical System: Power Supply

| Parameter | Description |
|---|---|
| Rated voltage U BATT | DC 12 V or DC 24 V |
| Permissible voltage range | 9 ... 32 VDC |
| Input current without camera | typ. 650 mA for DC 12 V |
| Input current without camera | typ. 320 mA for DC 24 V |
| Power consumption without camera | 7.8 W |

## Camera Connection

| Parameter | Description |
|---|---|
| Voltage | DC 12 V or<br>U BATT, if U BATT < DC 13 V |
| Current | max. 1 A |

## Display

| Parameter | Description |
|---|---|
| Display | 7" TFT LCD flat screen |
| Brightness | LED backlight (white) 300 cd/$m^2$ |
| Display resolution | 800 x 480 pixels (WVGA) |

## Keys, Digipot

| Parameter | Description |
|---|---|
| Keys | 4 illuminated silicone keys with night-lighting |
| Digipot | 16-position digital potentiometer with ENTER function |

## USB Stick

| Parameter | Description |
|---|---|
| Memory size | up to 8 GBytes |
| Supply voltage | 5 V, max. 150 mA |
| Short-circuit proof | yes,<br>Short-circuit current: ~ 1 A |

## Memory Configurations

| Parameter | Description |
|---|---|
| Number of remanent registers | 6.000 |
| Remanent memory for variables | 24,000 bytes |
| Flash disk | 12.875 MBytes |

**Battery**

| Parameter | Desription |
|---|---|
| Operating life | up to 4 years |
| Battery type | CR1225 (lithium button cell) |
| Voltage | 3 V |
| Capacity | 48 mAh |

**Technical Data - Real-Time Clock**

| Parameter | Description |
|---|---|
| Power reserve | 4 years |
| Deviation | Max. 1 minute per month |

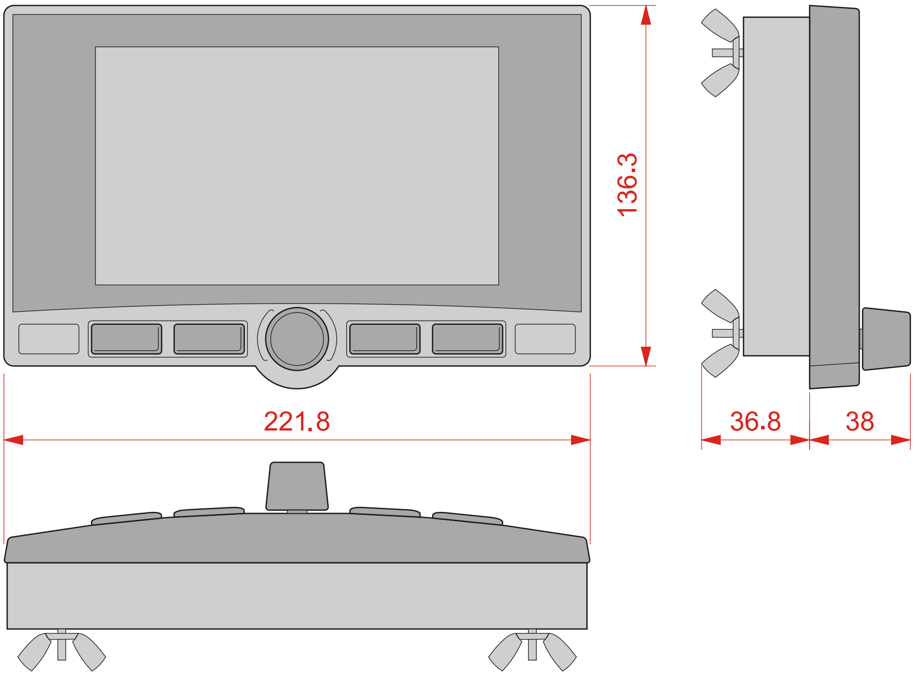
# Physical Dimensions

**Introduction**

This chapter details the physical dimensions of the JVM-407 and the conditions for installation.

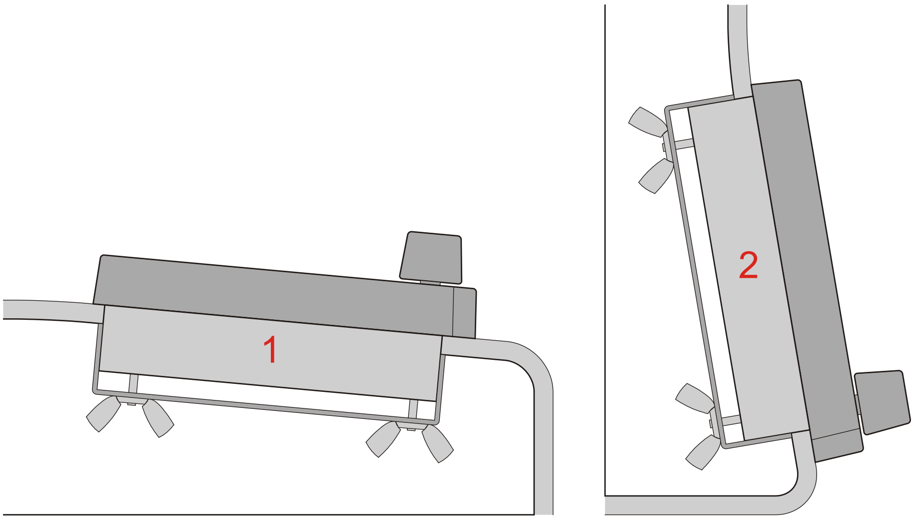
**Physical Dimensions**

The diagram shows the dimensions of the JVM-407.




**Permissible Installation Positions**

The diagram shows the positions permitted for installation.

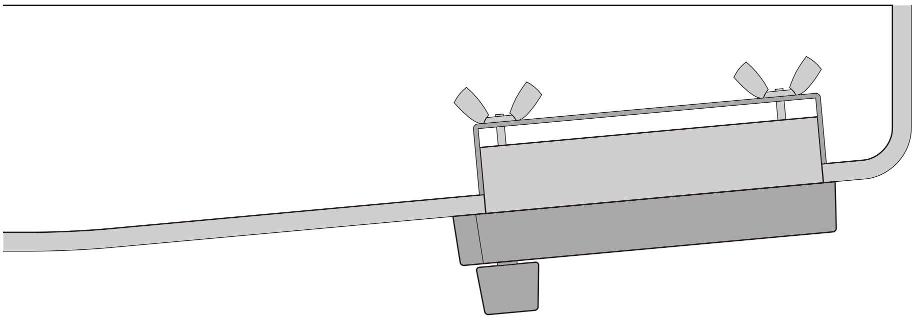


Explanations are as follows:

| Number | Permissible Installation Positions |
|---|---|
| 1 | horizontally or tilted |
| 2 | vertical or tilted |

**Prohibited Installation Positions**

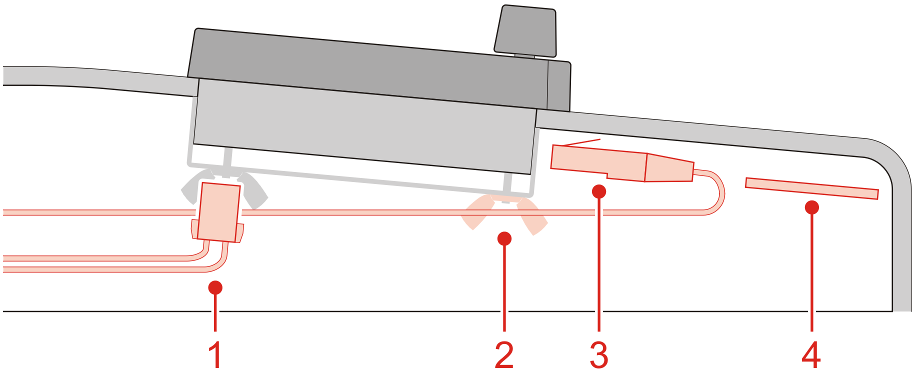
The diagram shows the positions prohibited for installation.



The rear panel of the HMI JVM-407 has no moisture protection, particularly against spray or water droplets. If the installation location cannot be guaranteed to be moisture-free, this method of installation (see diagram above) is prohibited. The accumulation of moisture and water droplets in the device can lead to current leakages and corrosion.

**Space Required for Installation and Service**

The diagram shows the space required for the HMI JVM-407.




Ensure there is enough space around the housing for servicing requirements.

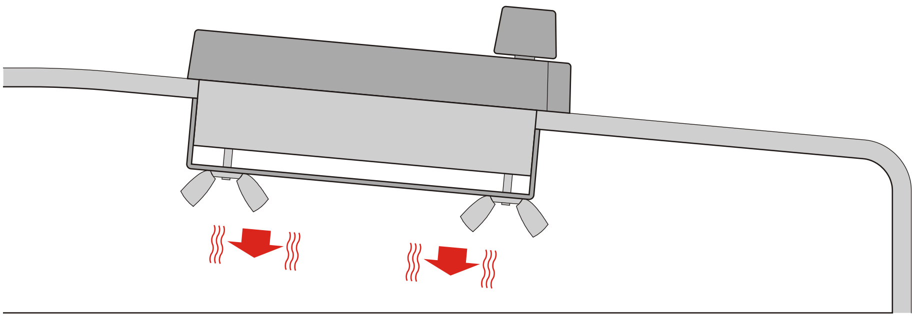
- It should be possible to disconnect the connector at any time.
- It should be possible to exchange the SD card at any time.
- It must be possible to easily loosen the wing nut on the SD card locking device.

Explanations are as follows:

| Number | Description |
|---|---|
| **1** | Connectors for CANopen®, video, power supply, inputs and outputs |
| **2** | Wing nut to secure the SD card |
| **3** | Network connector |
| **4** | SD memory card |

**Space Required to Protect Against Overheating**

The diagram indicates the safe distance to protect against overheating.



Please note:

- The JVM-407 increases the temperature of the environment as a result of heat emission under load.
  Power consumption is 7.8 W.
- The JVM-407 operates without interruption at an ambient temperature of up to +65 °C.

Consider the heat emission from the device, in particular when installing it in a critical environment:

- in the vicinity of the fuel tank
- in the vicinity of the fuel pipe
- in the vicinity of flammable vehicle components
- in the vicinity of thermally malleable vehicle components

**Installation Location**

The JVM-407 must be installed in the driver's cab.

# Operating Parameters - Environment and Mechanics

## Environment

| Parameter | Value | Standard |
|---|---|---|
| Operating temperature range | -20 ... +65 °C | |
| Storage temperature range | -30 ... +80 °C | DIN EN 61131-2<br>DIN EN 60068-2-1<br>DIN EN 60068-2-2 |
| Air humidity | 10 ... 95 %<br>Do not use a steam jet or other such devices to clean the JVM-407. | DIN EN 61131-2 |
| Climate test | Humid heat | DIN EN 60068-2-30 |
| Pollution degree | 2 | DIN EN 61131-2 |
| Installation Location | The JVM-407 must be installed in the driver's cab. | |

## Mechanical Parameters

| Parameter | Value | Standard |
|---|---|---|
| Vibration resistance | Vibration, broadband noise | DIN EN 60068-2-6<br>Severity level 2 |
| Shock resistance | 25 g occasionally, 11 ms, sinusoidal half-wave, 3 shocks in the directions of all three spatial axes | DIN EN 60068-2-27 |
| Degree of protection<br>Installation directly in console | front panel: IP64<br>rear panel: IP10 | DIN EN 60529<br>including all changes to date |
| Degree of protection<br>mounted on support arm | front panel: IP64<br>rear panel: IP64 | DIN EN 60529<br>including all changes to date |

## Operating Parameters - EMC

**EMC - Emitted Interference**

As per Directive 72/245/EEC with all amendments up to 2009/19/EC checked and compliant.

**EMC - Interference Immunity**

| Parameter | Value | Standard |
|---|---|---|
| Interference immunity to conducted faults | compliant | Directive 72/245/EEC with all changes up to 2009/19/EC |
| Interference immunity to external magnetic field | 20 ... 1,000 MHz: 100 V/m<br>1,000 ... 2,000 MHz: 30 V/m | Directive 72/245/EEC with all changes up to 2009/19/EC |
| Load Dump | Impulse 5b 70 V | ISO 7637-2 |

# B: Index

## A



## C



## E



## F



## H



## I



## M



## N



## O



## P



## Q



## R



## S



## T



## U

Jetter


**Jetter AG**

Graeterstrasse 2

D-71642 Ludwigsburg

**Germany**

| | |
|---|---|
| Phone: | +49 7141 2550-0 |
| Phone - Sales: | +49 7141 2550-433 |
| Fax - Sales: | +49 7141 2550-484 |
| Hotline: | +49 7141 2550-444 |
| Internet: | http://www.jetter.de |
| E-Mail: | sales@jetter.de |


## Jetter Subsidiaries


### Jetter (Switzerland) AG

Münchwilerstrasse 19

CH-9554 Tägerschen

**Switzerland**

| | |
|---|---|
| Phone: | +41 71 91879-50 |
| Fax: | +41 71 91879-69 |
| E-Mail: | info@jetterag.ch |
| Internet: | http://www.jetterag.ch |

### Jetter UK Ltd.

Old Witney Road

Eynsham

OX29 4PU Witney

Great Britain

| | |
|---|---|
| Phone: | +44 1865 883346 |
| Fax: | +44 1865 883347 |
| E-Mail: | info@jetter.uk.com |
| Internet: | http://www.jetter.uk.com |

### Jetter USA Inc.

13075 US Highway 19 North

Florida - 33764 Clearwater

**U.S.A**

| | |
|---|---|
| Phone: | +1 727 532-8510 |
| Fax: | +1 727 532-8507 |
| E-Mail: | bschulze@jetterus.com |
| Internet: | http://www.jetter.de |