Jetter automation

User Manual

JVM-104 - HMI

60880105

We automate your success.

Variant: Jetter
Design: O01
Item #: 60880105
Revision 4.01.2
May 2015 / 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

Significance of this User Manual

This document is an integral part of the JVM-104:

- Keep this document in a way that it is always at hand until the JVM-104 will be disposed of.
- Pass this document on if the JVM-104 is sold or loaned/leased out.

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

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

This document contains important information on the following topics:

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

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

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

Table of Contents

1 Safety instructions

Introduction

This chapter informs the user of basic safety instructions. It also warns the user of residual dangers, if there are any.

Contents

Basic safety instructions

Introduction

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

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

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

Intended conditions of use

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

The device has been designed for use in commercial vehicles and mobile machines. The device JVM-104 is an HMI with integrated controller for exchange of data with peripheral devices.

The HMI JVM-104 meets the requirements of the European Automotive EMC Directive for electric/electronic subassemblies.

Operate the JVM-104 only within the limits set forth in the technical specifications. Because of its low operating voltage, the JVM-104 is classified as a SELV (Safety Extra-Low Voltage) system. Therefore, the HMI JVM-104 is not subject to the EU Low Voltage Directive.

Usage other than intended

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

The JVM-104 is no safety-related part as per Machinery Directive 2006/42/EC. This device is not qualified for safety-relevant applications and must, therefore, NOT be used to protect persons.

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

Personnel qualification

Depending on the life cycle of the product, the persons involved must possess specific qualifications. The qualifications required to ensure safe handling of the device at different phases of the product life cycle are listed below:

Product life cycle	Minimum qualification
Transport/storage:	Trained and instructed personnel with knowledge in handling electrostatically 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/ disposal:	Specialized personnel with training in electrical/automotive engineering, such as automotive mechatronics fitters

Modifications and alterations to the module	**For safety reasons, no modifications and changes to the device and its functions are permitted.** Any modifications to the device not expressly authorized by Jetter AG will result in a loss of any liability claims to Jetter AG. **The original parts are specifically designed for the device. Parts and equipment from other manufacturers have not been tested by Jetter AG and are, therefore, not released by Jetter AG.** The installation of such parts may impair the safety and the proper functioning of the device. Any liability on the part of Jetter AG for any damages resulting from the use of non-original parts and equipment is excluded.
Transport	The JVM-104 contains electrostatically sensitive components which can be damaged if not handled properly. To exclude damages to the JVM-104 during transport it must be shipped in its original packaging or in packaging protecting against electrostatic discharge. ▪ Use an appropriate outer packaging to protect the JVM-104 against impact or shock. ▪ In case of damaged packaging inspect the device for any visible damage. Inform your freight forwarder and Jetter AG.
Storing	When storing the JVM-104 observe the environmental conditions given in the technical specification.
Repair and maintenance	The operator is not allowed to repair the device. The device does not contain any parts that could be repaired by the operator. If the device needs repairing, please send it to Jetter AG.
Disposal	When disposing of devices, the local environmental regulations must be complied with.

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

The HMI JVM-104

The JetView of the mobile automation series 104 is a compact full-graphics HMI. The HMI JVM-104 is extremely versatile thanks to its compact design and the integrated controller. The JVM-104 has especially been designed for use in the harsh environment of commercial vehicles and mobile machines.

The HMI can be operated in all light conditions, due to the backlit keys and the light sensor, which automatically adapts the brightness of the display to the brightness of the surroundings.

Product features

The features of this product are listed below:

- Display: 3.5" TFT, 350 cd/m^2
- Resolution: QVGA (320 x 240 pixels)
- Touchscreen
- 4 function keys (lighted)
- 1 digipot with pushbutton function
- Adjustable background lighting
- Adjustable night-lighting
- Loudspeaker
  Volume: 83 dB at a distance of 10 cm at resonance frequency of 2,670 Hz
  Adjustable frequency and volume.
- Powerful programming language JetSym STX
- Fast ARM11 CPU
- Non-volatile registers 30,000
- RAM: 128 MBytes
- Flash memory: 512 MBytes
- 1 CAN-2.0B interface

Accessories

The accessories are provided in the fastening kit. It includes a fastening bracket, a sealing ring and the corresponding screws and nuts.

Item no.	Quantity	Description
60880138	1	Fastening kit

Scope of delivery

The following items are included in the scope of delivery of the JVM-104:

Item no.	Quantity	Description
10001018	1	HMI JVM-104
60879282	1	Installation manual

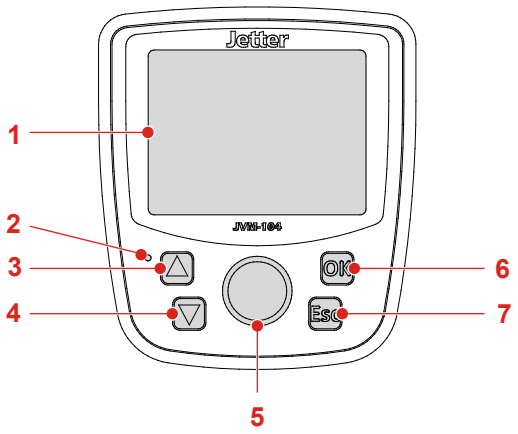
Parts and interfaces

Introduction

This chapter describes the parts and interfaces of the JVM-104.

Front panel of the JVM-104

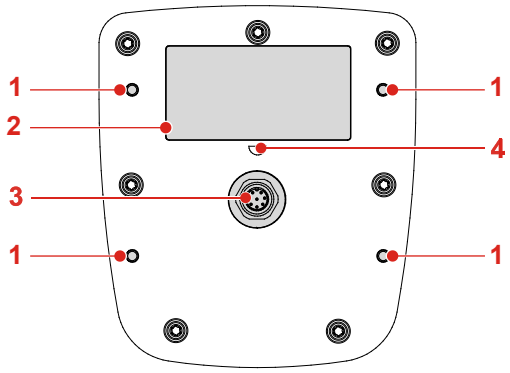
The HMI JVM-104 provides a touchscreen of an active surface of 3.5". The illustration shows the front panel of the HMI with all its control elements.



Number	Part	Description
1	TFT display	Active surface, touchscreen
2	Brightness sensor	Senses the surrounding brightness
3	Input key UP	Key with background lighting
4	Input key DOWN	Key with background lighting
5	Digipot	Rotary and pushbutton
6	Input key OK	Key with background lighting
7	Input key ESC	Key with background lighting

Rear panel of the JVM-104

The illustration shows the rear panel of the HMI with all its connections and the nameplate.



Number	Part	Description
1	Screw holes	For fastening the HMI. Max depth: 12 mm
2	Nameplate	
3	M12 male connector	
4	PV	Protective vent

Order reference

Order reference

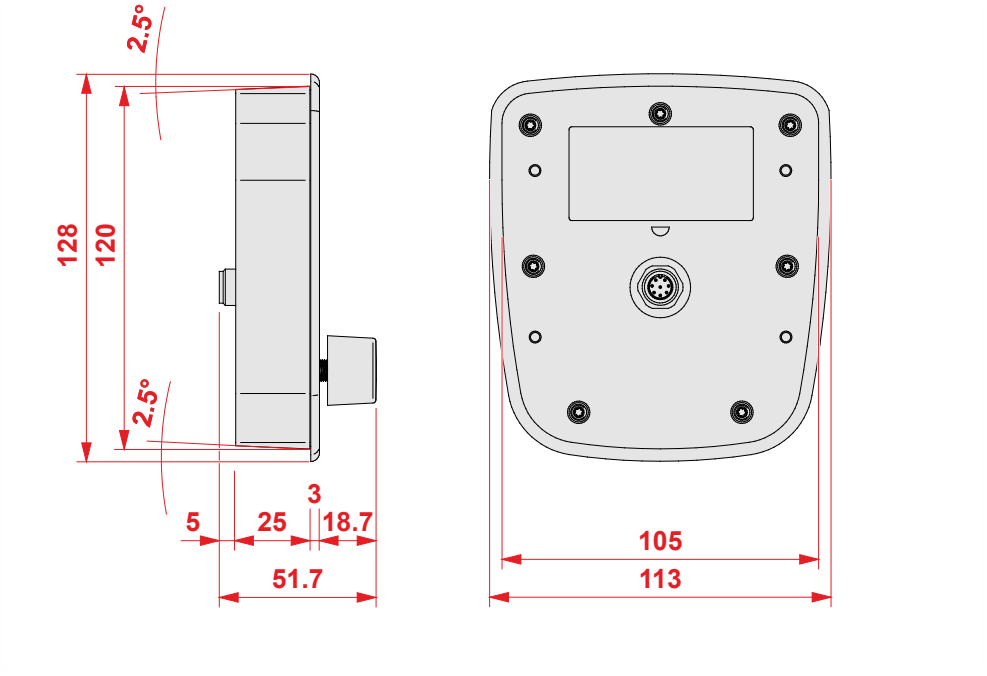
The HMI JVM-104 can be ordered from Jetter AG using the following item number:

Item no.	Order reference
10001018	JVM-104-K00-O01

Physical dimensions

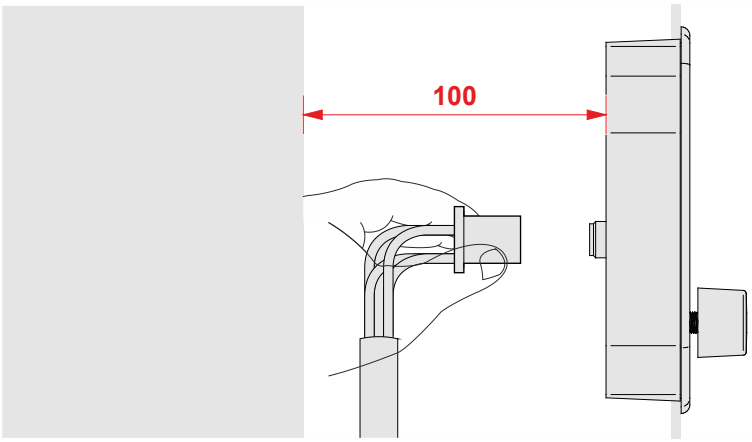
Physical dimensions

The illustration below shows the physical dimensions of the JVM-104 in millimeters.



Space required for installation and service

The illustration shows the space required for the HMI JVM-104. It is stated in millimeters.



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

- It should be possible to disconnect the connector at any time.

3 Identifying the JVM-104

Purpose of this chapter

This chapter supports you in retrieving the following information about the JVM-104:

- Hardware revision
- Electronic data sheet (EDS). Numerous manufacturing-related data are stored to the EDS.
- Software versions

Prerequisites

To be able to identify technical data about the HMI JVM-104 the following prerequisites must be fulfilled:

- The HMI is connected to a PC.
- The programming tool JetSym 5.1.2 or higher is installed on the PC.

Information for hotline requests

If you wish to contact the hotline of Jetter AG in case of a problem, please have the following information on the JVM-104 ready:

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

Contents

3.1 Identification by means of the nameplate

Introduction

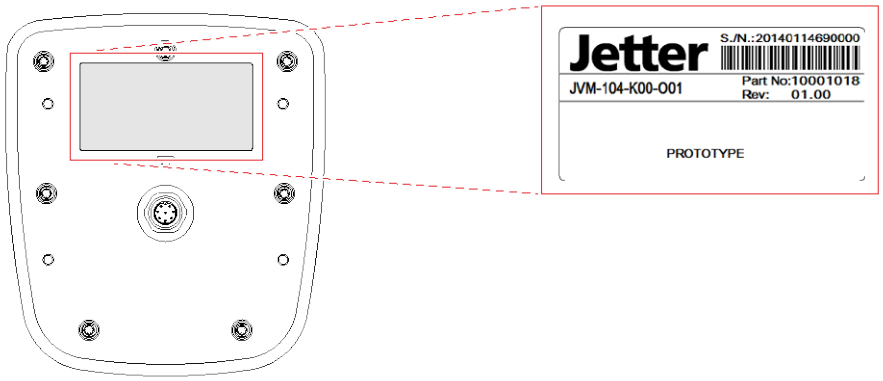
Each HMI JVM-104 can be identified by its nameplate attached to its enclosure. If you wish to contact the hotline of Jetter AG in case of a problem, please have information on the hardware revision and serial number ready.

Contents

Nameplate

Position of the nameplate

The location of the nameplate on the rear panel of the JVM-104 is shown below.



Nameplate

The nameplate of a JVM-104 contains the following information:

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

3.2 Version registers

Introduction

The operating system of the JVM-104 provides several registers which let you read out the version numbers of the OS and its components. If you wish to contact the hotline of Jetter AG in case of a problem, please have this information ready.

Contents

Software versions

Introduction

The JVM-104 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 JVM-104 is a four-figure value.

1	.	2	.	3	.	4

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

Released version

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

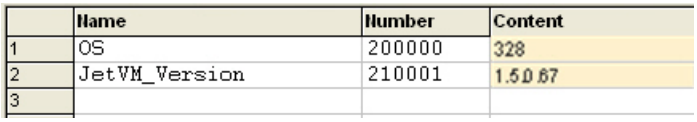
Overview of registers

The following registers let you read out the software versions:

Register	Description
200000	Operating system version
210001	Version of the STX interpreter for the STX application program (JetVM version)

Version numbers in JetSym setup

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



	Name	Number	Content
1	OS	200000	328
2	JetVM_Version	210001	1.5.0.67
3			

4 Mounting and installation of the JVM-104

Purpose of this chapter

This chapter describes the installation of the JVM-104 in the vehicle as regards the following points:

- Planning the wiring of a JVM-104
- Installation
- Configuration of the IP interface for the JVM-104

Contents

4.1 Interfaces

Introduction

The HMI JVM-104 is equipped with the following interface:

- M12 male connector

M12 male connector

The M12 connector has the following function:

- Power supply of the JVM-104
- CANopen® bus interface: CAN 1
- Recognition of the ignition

Contents

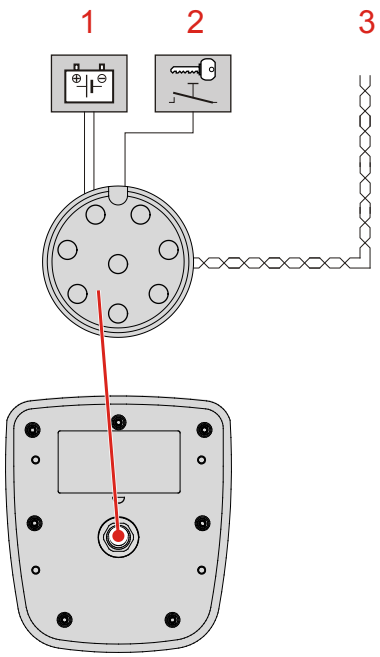
Example - Wiring

Introduction

The following examples shows how to connect a JVM-104.

Example

The illustration shows an example of a wiring layout.



Number	Description
1	Power supply (battery)
2	Ignition lock
3	CANopen® bus

Connecting the power supply

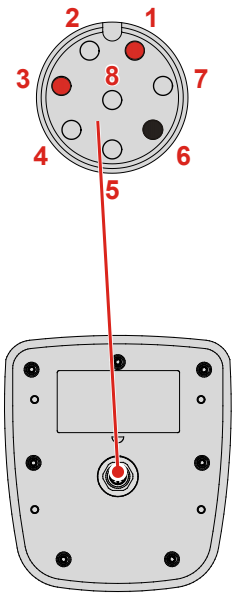
Purpose of the connector

This connector is also used for the following peripheral devices:

- Power supply
- Recognition of the ignition

Pinout of the power supply connector

The diagram shows the pinout of the power supply and ignition connector (viewing the cable side):



The pinout is as follows:

Pin	Description	Terminal number in vehicles
1	Power supply UB for logic circuits Voltage: DC 12 V or DC 24 V Power consumption: 2 A max.	Terminal # 30
3	Ignition (+)	Terminal # 15
6	Reference potential (GND)	Terminal # 31

Technical specifications - Power supply UB

Parameter	Description
Rated voltage	DC 12 V or DC 24 V
Permissible voltage range UB	DC 8 V ... DC 32 V, to ISO 7637
Permissible voltage range - Ignition	DC 5 V ... DC 32 V
Maximum current	2 A
Load dump protection	DC 70 V max.
Typical current consumption logic circuit (UB)	170 mA at DC 12 V 90 mA at DC 24 V
Power consumption	Approx. 2 W
Integrated protective functions	Protection against polarity reversal, overloading, voltage surges

Note on Ignition

To start the JVM-104, pin 3 (IGNITION FEED) must be connected with pin 1 (STANDARD FEED). The ignition control signal is issued when the key is in position *Ignition ON*.

Note on current consumption

When the JVM-104 is energized, the current consumption is temporarily higher. To guarantee reliable power-up of the JVM-104, supply at least three times as much power as would typically be needed.

Mating part

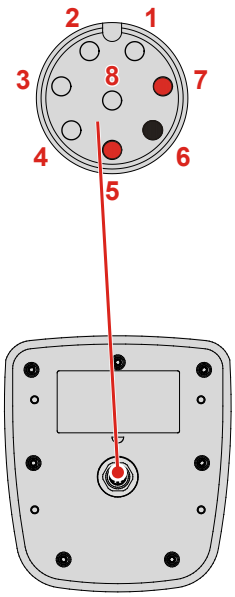
The following jack is a mating part to the M12 connector:

Manufacturer	e.g. BELDEN Lumberg automation
Manufacturer's item no.	RKCN 8/9
Wire size:	0.5 mm^2 (AWG 20)

CAN interface

Pinout of the CANopen® bus

The diagram shows the pinout of the connector for the CANopen® bus (viewing the cable side): Pin 6 for the reference potential is also color-coded.



The pinout is as follows:

Pin	Description
5	CAN_L
6	Reference potential (GND)
7	CAN_H

Technical specifications - CAN interface

Parameter	Description
Baud rate	250 kBaud ... 1 MBaud
Bus terminating resistor	None
External bus termination	120 Ω
Connector specifications	Twisted pair conductors, unshielded

Bus terminating resistor

The JVM-104 has not got an integrated bus terminating resistor.

Twisting

The CAN_L and CAN_H cable pairs must be twisted.

Specification - CAN bus cable

Parameter	Description
Core cross-sectional area	1000 kBaud: 0.25 … 0.34 mm^2 500 kBaud: 0.34 … 0.50 mm^2 250 kBaud: 0.34 … 0.60 mm^2
Cable capacitance	60 pF/m max.
Resistivity	1000 kBaud: 70 Ω/km max. 500 kBaud: 60 Ω/km max. 250 kBaud: 60 Ω/km max.
Number of cores	2
Twisting	CAN_L and CAN_H cables are twisted pairwise

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	Total stub length
1000 kBaud	25 m max.	0.3 m max.	1.5 m
500 kBaud	100 m max.	5 m max.	30 m
250 kBaud	250 m max.	10 m max.	60 m

Mating part

The following jack is a mating part to the M12 connector:

Manufacturer	e.g. BELDEN Lumberg automation
Manufacturer's item no.	RKCN 8/9
Wire size:	0.5 mm^2 (AWG 20)

4.2 Installing the JVM-104

Introduction

This chapter describes how to install the JVM-104.

Contents

Installation

Introduction

This chapter describes how the HMI JVM-104 is to be installed.

Selecting a place for installation

Select a suitable place for the device to be installed.
The place where the device is to be installed must meet the following requirements:

- 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 locations that do not meet the a.m. requirements.
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. Therefore, 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.
Uneven installation surfaces	The installation surface could become misshapen when fitting the HMI. Fastening is unstable and precarious.

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 unsuitable from an ergonomic point of view:

- 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

Accessories

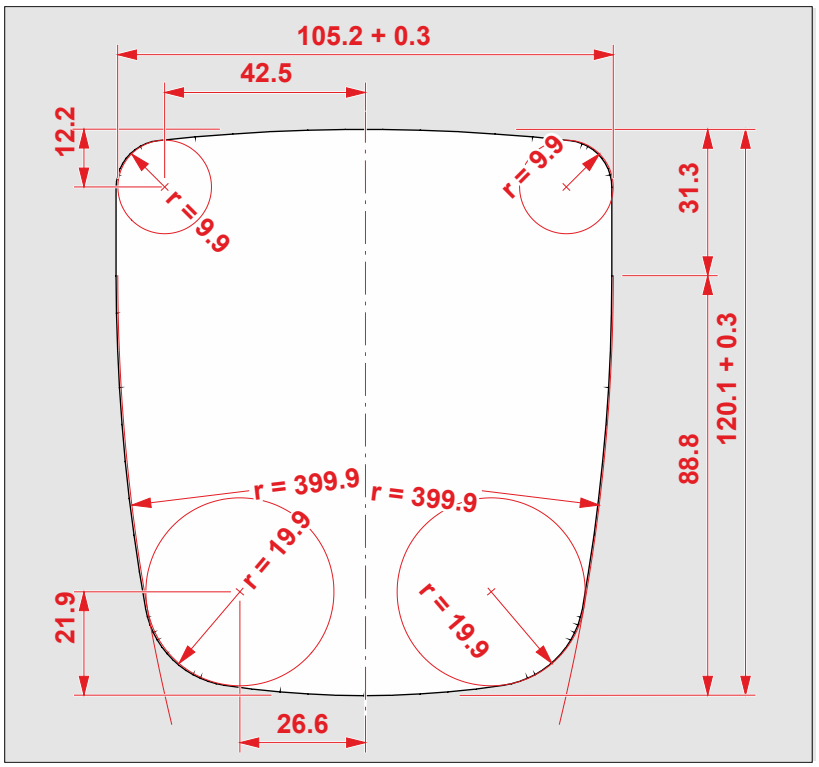
The accessories are provided in the fastening kit. It includes a fastening bracket, a sealing ring and the corresponding screws and nuts.

Item no.	Quantity	Description
60880138	1	Fastening kit

Preparing for installation

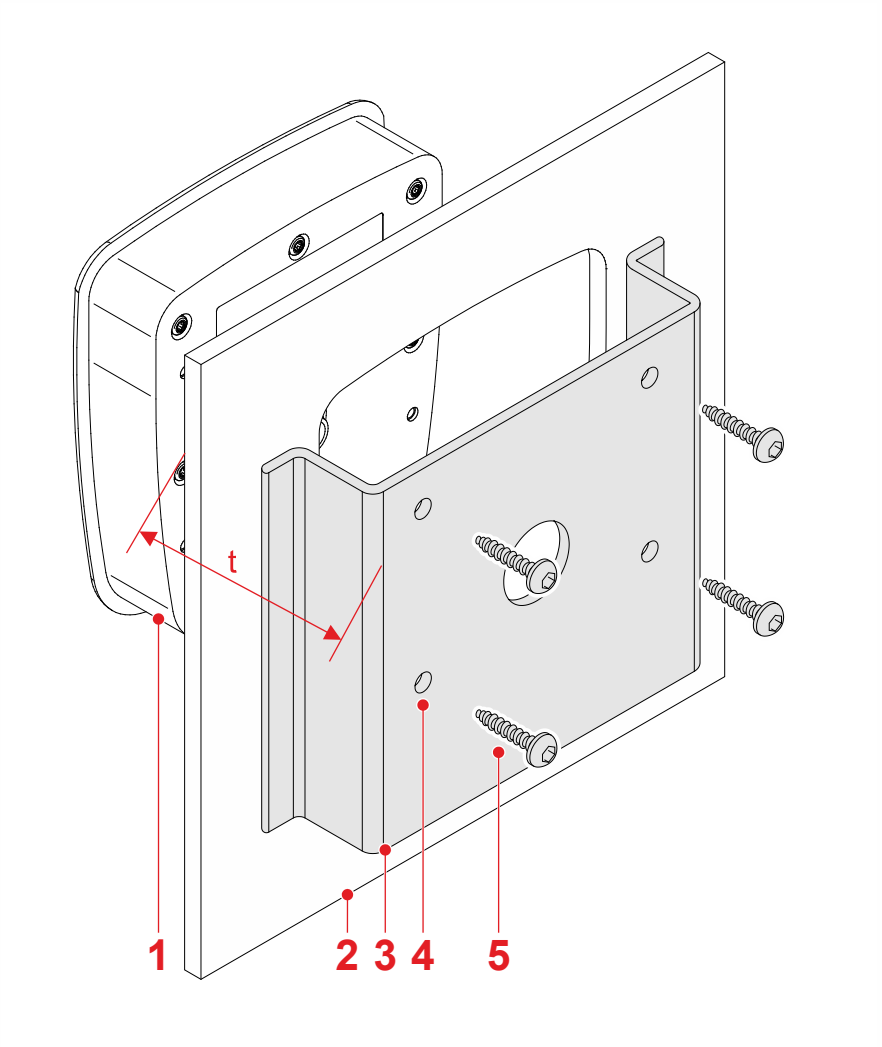
Make a fitting opening in the panel.

The illustration shows the shape of the opening and the dimensions in millimeters:



Installing the HMI

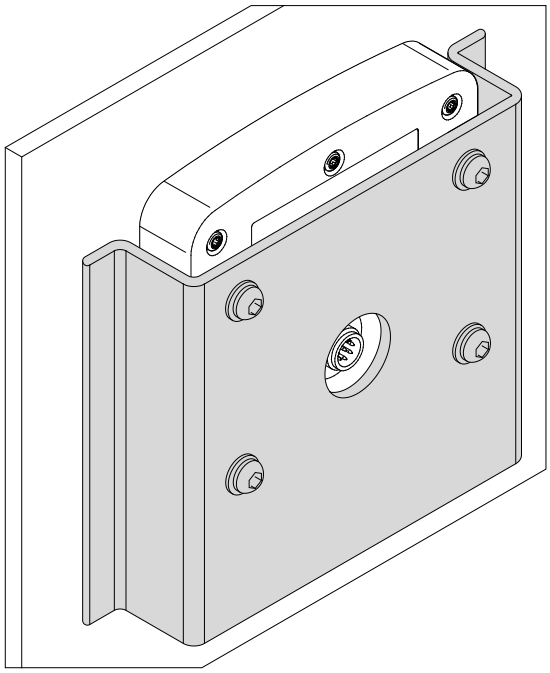
The illustration shows how to install the device.



Number	Description
1	JVM-104
2	Panel with opening for accommodating the HMI
3	Fastening bracket with opening for the connectors
4	4 x screw holes for screwing down the JVM-104
5	4 x self-tapping screw Screw size: 4 x 9 + t Tightening torque: 1.6 Nm ± 10 % Maximum screw-in depth: 12 mm

Step	Action
1	Insert the HMI into the front of the opening in the panel.
2	Hold the fastening bracket against the panel from the rear. To this end, the connectors must be seen through the opening of the fastening bracket.
3	Screw the HMI, together with the fastening bracket, onto the panel. The stud torque should be 1.6 Nm ± 10 %.

The illustration shows the installed HMI JVM-104.

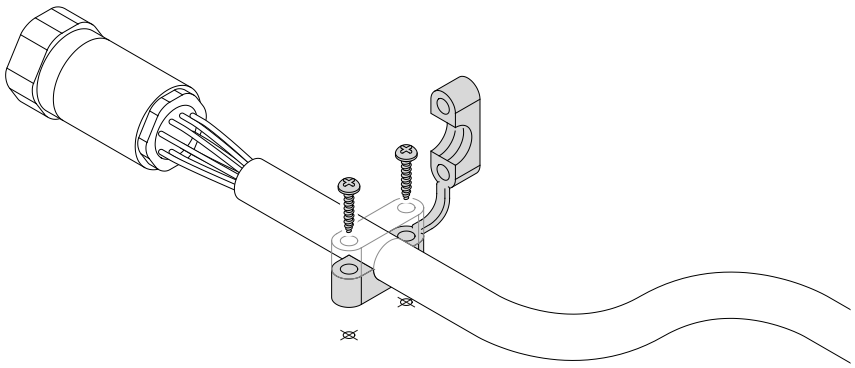


Installing the strain relief

Install strain reliefs for the connecting cables.

Take care to leave enough space for the connectors.

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



5 Initial commissioning

Purpose of this chapter

This chapter describes how to commission the JVM-104 and covers the following topics:

- Initial commissioning in JetViewSoft
- Initial commissioning in JetSym

JetViewSoft is a SCADA system and JetSym is a programming tool. Both have been developed by Jetter AG.

For more information refer to the Online Help in JetSym or JetViewSoft.

Minimum requirements

These instructions for initial commissioning apply to JetSym version 5.1.2 or higher and JetViewSoft version 4.0.2 or higher.

Contents

5.1 Preparatory work and first insight into programming with JetSym STX

Introduction

This chapter covers the preparatory work for commissioning the JVM-104. It also provides a first insight into the programming language JetSym STX.

Contents

Preparatory work for initial commissioning

Establishing a CAN connection

To be able to commission and program the JVM-104, complete the following activities first:

- Wire up the power supply units, ignition and CAN interface
- Connecting an USB CAN adapter between the controller and the PC
- Installing the respective adapter driver software

In order to commission the JVM-104, you don't have to connect any peripheral devices to it.

Default values on the CANopen® bus

The default values of the JVM-104 are listed below:

- CAN baud rate: 250 kBaud
- CANopen® node ID: 0x7F

Note

The device JVM-104 is not equipped with an internal (activatable) terminating resistor for the CAN bus.

Make sure that there is a terminating resistor of 120 Ω at both ends of the CAN bus.

Supported USB CAN adapters

The programming environment JetSym supports the following USB CAN adapters:

- **IXXAT Automation GmbH** (**http://www.ixxat.de http://www.ixxat.de**): For a list of currently supported hardware refer to the website of IXXAT Automation GmbH.
  The following driver versions are supported: VCI version 3.3, and VCI version 2.18
- **PEAK-System Technik GmbH** (**http://www.peak-system.com http://www.peak-system.com**): For a list of currently supported hardware refer to the website of PEAK-System Technik GmbH.
  The following driver versions are supported: Version 3.5.4.9547 or higher

Installing the USB CAN adapter

Prerequisites:

Before installing the driver software of the USB CAN adapter, **JetSym 4.3** or higher must be installed on the PC to be used.

To install the adapter proceed as follows:

Step	Action
1	Insert the USB CAN adapter into a USB port of your PC.
2	If the hardware installation assistant opens, terminate it.
3	Install the driver for the USB CAN adapter.
4	Install the corresponding JetSym driver depending on the USB CAN adapter used.

Step	Action	
	If ...	**... then ...**
	... you use an adapter by PEAK-Systems, ...	... proceed with step 5.
	... you do not use an adapter by PEAK-Systems, ...	... proceed with step 7.
5	Navigate in Windows Explorer to the folder **PcanDrv** located in the JetSym installation. Default location: *C:\Programme\Jetter\JetSym\Tools\PcanDrv*	
6	Execute the file **PcanDrv.exe**. Follow the steps of the installation routine.	
7	Plug the Sub-D connector of the adapter into the IN_CAN port of the JVM-104 (female Sub-D connector).	

Result: In the case of an error-free installation the CANopen® connection between PC and controller is established.

Requirement for power-up

The JVM-104 only powers up if the supply voltage +UB is applied to the ignition (+).

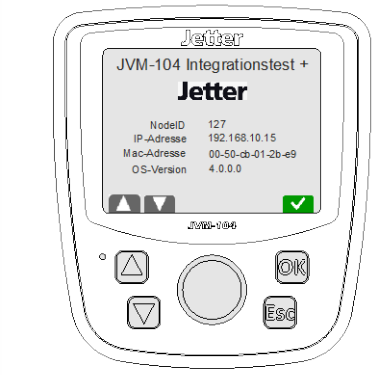
Skipping the application program during power-up

If, during power-up, you press the keys ▼ and **OK** simultaneously, you prevent the application program from being launched.

It may happen that the device does not react after power-up. This condition, however, lets you access the device using FTP or JetSym.

Default display

The default application program launched on the JVM-104 after power-up displays the following input mask on the display.



The node ID displayed is the address of the CANopen® bus 1 set in the JVM-104. This address can be set by the keys ▲ and ▼ .

Key ▲ increases the address in steps of 1.

Key ▼ decreases the address in steps of 1.

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

Programming in the programming language JetSym STX

Introduction

JetViewSoft lets you create visualization applications for use on the following platforms:

- PC systems
- HMIs for industrial applications
- HMIs for mobile applications

JetSym STX lets you access visualization objects and control their representation on the HMI. The programming language JetSym STX lets you program the HMI as if it were a controller. The compiled programs can be processed in the HMI without the need for an external controller. This is made possible by the STX interpreter and the graphical runtime environment JVER (JetView Embedded Runtime). Both form an integral part of the HMI's operating system.

JetSym STX program

The program below just causes an internal variable within a loop to be doubled to value 20. This example shows how JetSym STX can be used.

```
Task MiniExample AutoRun
Var
    i, j : Int;
End_Var;
    j := 1;
    // j is being run through within a loop up to value 1024
    For i := 1 To 10 Do
      j := j * 2;
    End_For;
End_Task;
```

Program location

When you load the compiled program to the HMI, JetSym creates in the directory *\App* a folder and names it after the project. JetSym stores the application program to this folder. The file name of the application program comprises of the project name and the extension **.es3**. Path and file names are always converted into lower case letters.

The file **start.ini** is automatically created on program download. It defines which application program is to be loaded.

5.2 Configuring a project for the ER-STX-CE platform

Introduction

This chapter describes how to create and configure in JetViewSoft and JetSym a visualization project for the ER-STX-CE platform.

Contents

Initial commissioning in JetViewSoft

Introduction

JetViewSoft lets you create visualization files for the JVM-104 and upload them to the HMI. This topic covers the following:

- Creating a project in JetViewSoft
- Making project settings
- Creating visualization files and uploading them to the HMI

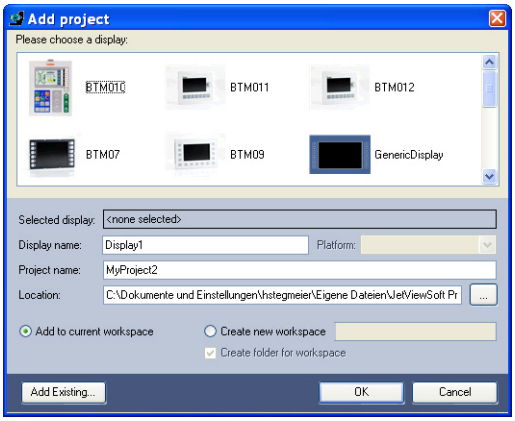
Prerequisites

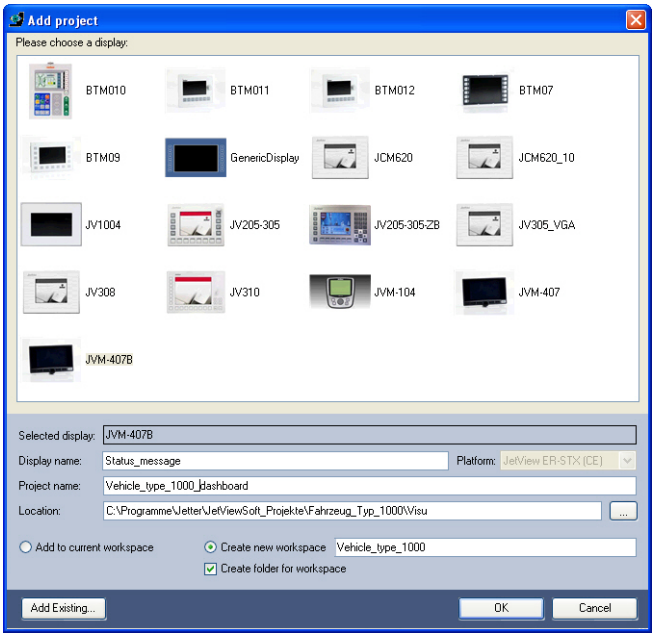
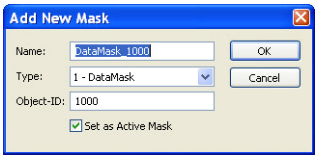
The following prerequisites must be fulfilled:

- JetViewSoft must be installed on the PC.
- JetViewSoft must be licensed (see Online Help in JetViewSoft).
- An active CAN connection between the PC and the HMI must be set up.

Creating a project

To create a new project for the HMI in JetViewSoft, proceed as follows:

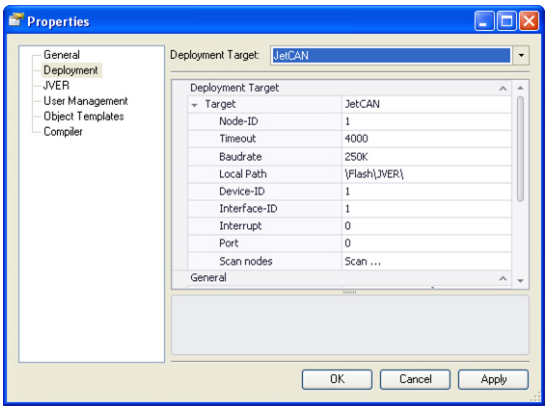
Step	Action
1	Start JetViewSoft
2	Open the **File** menu. Select menu item **New Project**. **Result:** The following dialog box opens: 
3	Select in **Selected display:** the HMI used. To do so, click on the image of the corresponding HMI.
4	In **Display name**, select a program-internal name for the HMI. You can add one or more HMIs to a project.
5	If you have got the possibility to make a selection: Select **Platform JetView ER-STX(CE).**
6	In **Project name**, enter the name of the project.
7	If necessary, change the project menu path under **Location**. For better clarity, the path should end with *\Visu*.
8	Enter the name of the workspace into **Workspace**.

Step	Action
⇨	The screenshot below shows an example of the completed dialog box:  **Important note:** The names must not contain blanks. Otherwise, it will be difficult to delete the visualization files at a later date.
9	Confirm your settings by clicking **OK**. **Result:** The dialog box closes and the **Add New Mask** dialog box opens. 
10	Enter the name of the first data mask into the box **Name**. Leave all other settings unchanged This mask is automatically the active mask when launching the HMI.
11	Confirm the settings by clicking **OK**.

Result: Creation of the project is completed.

Making the deployment settings

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

Step	Action
1	Open the menu **Project**. Select menu item **Properties**. **Result**: The dialog box of the same name opens.
2	Open the **Deployment** pane from the navigation panel on the left-hand side of the dialog box. 
3	Under **Deployment Target**, select **JetCAN**.
4	Click on the + sign next to **Target** to expand the setting options. Or just double-click **Target**.
5	Under **Node-ID,** enter the node ID of the HMI. The default node ID of a JVM-104 is **0x7F**.
6	Enter the baud rate into the box **Baudrate**. The default baud rate is **250K**.
7	Enter the project path *\App\projectname* into the box **Local Path**. In this case, projectname is a placeholder representing the actual name of the project (in lowercase letters).
8	Confirm your settings by clicking **OK**.

Result: The deployment settings have now been made and you can transfer the files to the HMI.

Transferring a project to the HMI

To create a JetViewSoft project and to transfer it to the HMI, proceed as follows:

Step	Action
1	Create a screen mask using the available objects (rectangles, ellipses, etc.). Once these objects have been transferred, they can be seen on the HMI.
2	Open the **File** menu. Select the menu item **Save all**.
3	Press the **[F7]** key to trigger the build process for this project. **Result:** JetViewSoft compiles the project files as long as no error occurs.
4	Open the menu **Build**. Select menu item **Deploy**. Or press the keyboard shortcut **[CTRL] + [F5]**. **Result:** JetViewSoft transfers the files to the HMI.
5	In order to make the HMI read in the visualization files, restart it.

Result: The files of your JetViewSoft project have been stored to the directory *\App\projectname* on the HMI. The HMI shows the start screen.

Missing visualization application

If there is no visualization application on the device, the display shows the following message:

The folder **Data** is empty. That is, there is no visualization application and no JVER (JetView Embedded Runtime) on the device. If JVER is not running (desktop background is visible), communication with JetSym is not possible.

Remedy: Use JetViewSoft to upload a visualization application to the device.

IOP file as visualization application on the HMI

In delivered condition, the HMI may already include a visualization application with an ***.iop** file stored to the folder **Data** .

This is also the case, if the CAN bus node ID must be set.

Result: The HMI will not display your visualization application.

Remedy:

Step	Action	
1	**If ...**	**... then ...**
	... the file **\App\visual.iop** or **\Data\visual.iop** exists, ...	... delete or rename this file.
2	**If ...**	**... then ...**
	... the file **\App\JetViewERS.cfg** exists, ...	... delete or rename this file.
⇨	The visualization application developed for the ER-STX-CE platform is displayed.	

Related topics

- **Initial commissioning in JetSym** (see page 50)

Creating and configuring a visualization project in JetSym

Introduction

The programming tool JetSym STX lets you create visualization applications for the HMI JVM-104. This topic covers the following:

- Creating a project in JetSym STX
- Configuring the controller hardware
- Including the visualization library JVER-STX
- Creating a program that can be compiled and transferred to the HMI

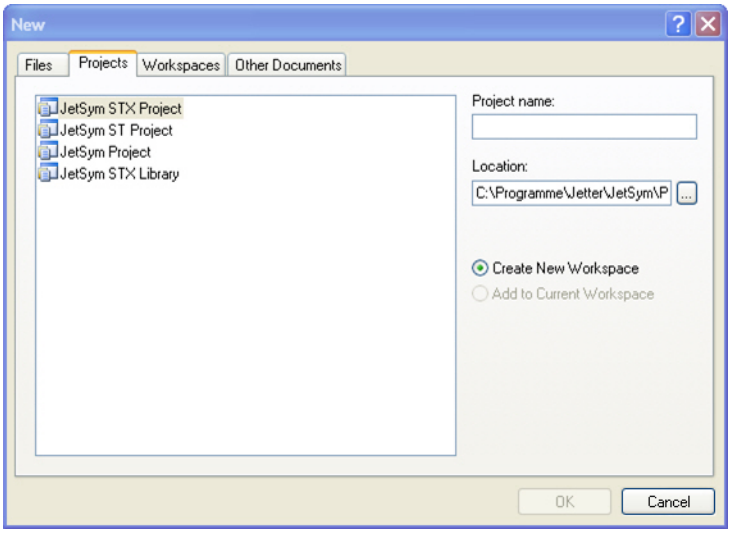
Prerequisites

The following requirements must be satisfied:

- JetSym has been installed on the PC used.
- JetSym has been licensed (see online help in JetSym).
- The controller has been connected to the same network as the PC.
- An active CAN connection between controller, HMI, and PC has been established.
- Initial commissioning in JetViewSoft has been completed.

Creating a project

To create a new programming project in JetSym, proceed as follows:

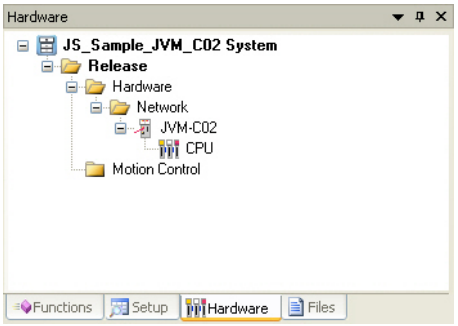
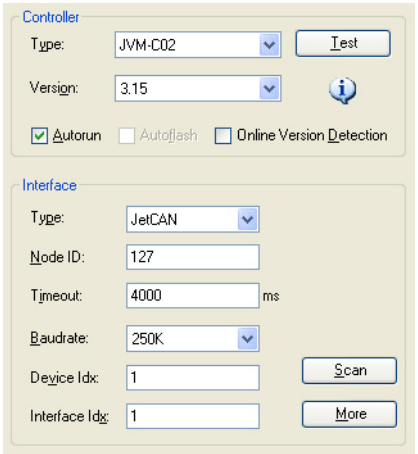
Step	Action
1	Launch JetSym.
2	Open the menu **File**. Select menu item **New**. **Result:** The dialog box **New** opens. 
3	Select **JetSym STX project** as the project type.
4	Enter the project name.

Step	Action
5	Select the path. It is recommended to store project files within a JetViewSoft project to the directory **STX**. **Example:** *C:\Programs\Jetter\JetViewSoft_Projects\VehicleType_1000\Visu\VehicleType_1000\VehicleType_1000_Dashboard\STX* **Advantage:** The JetSym project files are located in the same directory as the file **VisualInterface.stxp** created by JetViewSoft.
6	Confirm your settings by clicking **OK**.

Result: Creation of the project is completed.

Configuring the hardware

To establish a connection between JetSym and the HMI, you need to configure the hardware.

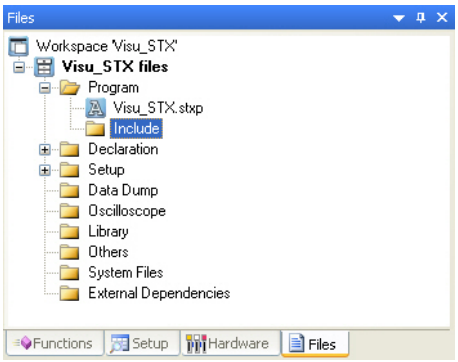
Step	Action
1	Navigate to the tab **Hardware** and click it. 
2	Fully expand the **Hardware tree**.
3	If you wish to set JVM-C02 as HMI or set interface parameters, double-click **CPU**. **Result:** The dialog box **Configuration** opens. 
4	From **Controller/Type**, select JVM-C02.
5	Under **Interface/Type** select JetCAN.

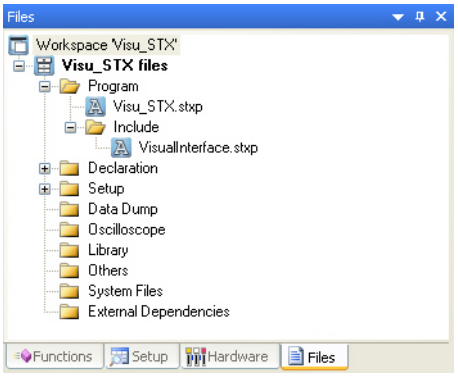
Step	Action
6	Enter the node ID of the HMI into the box **Interface/Node-ID.** If the node ID is unknown to you, it can be retrieved by the **Scan hardware** function.
7	Under **Interface/Baudrate** select 250K.
8	Test the connection with JVER running by pressing the button **Test**. If the test is unsuccessful, check the node ID, the baud rate and the CAN connection with the JVM-104.
9	Save your settings using the shortcut **[Ctrl] + [S]**.

Result: The hardware settings have been configured in JetSym.

VisualInterface.stxp - Include in the project

In order for the description of the objects and masks included in the visualization application to be available for programming, the file **Visualinterface.stxp** must be included as follows:

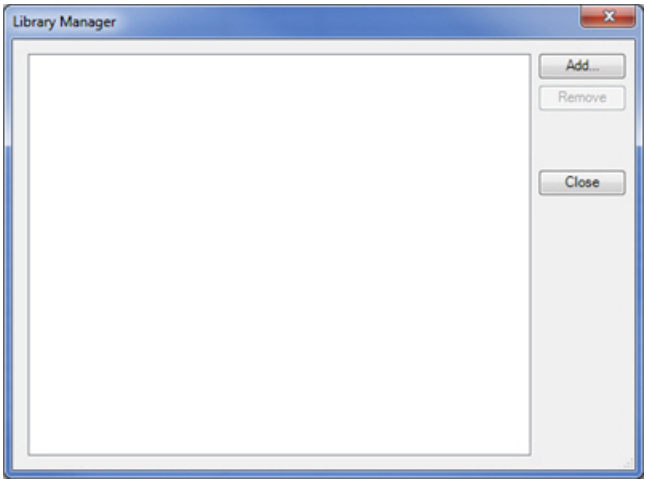
Step	Action
1	Switch to the view **Files**. 
2	Expand the folder **Program**.
3	Click on the folder **Include** and open the shortcut menu (by pressing the right mouse button).
4	Select the shortcut menu entry **Add Files to Directory**. **Result:** An Explorer window for selecting a file opens.
5	Navigate to the **STX** folder of the JetViewSoft project. The default location for this is at *[Project location]/ Name of the JetViewSoft project/STX*.
6	Select here the file **VisualInterface.stxp**.

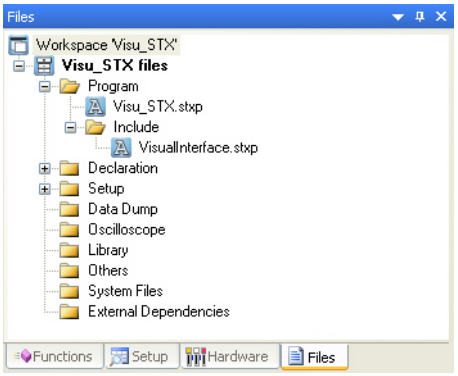
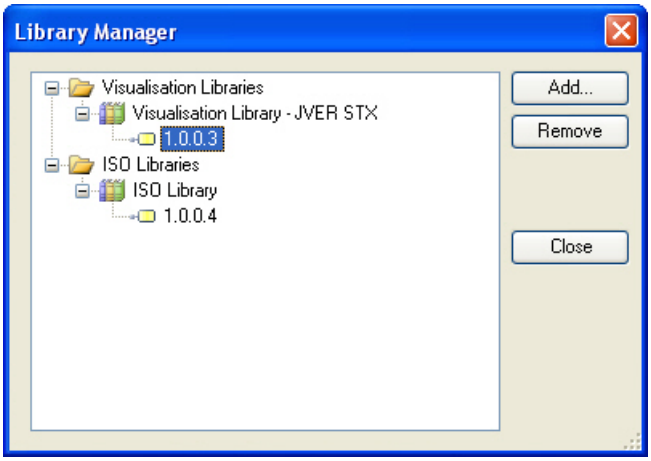
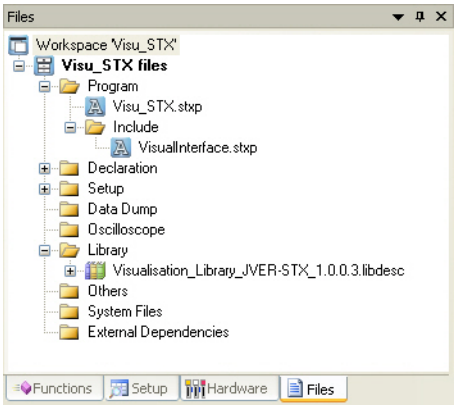
Step	Action
7	Click the button **Open**. 

Result: The file **VisualInterface.stxp** is now included into the project.

Including a library

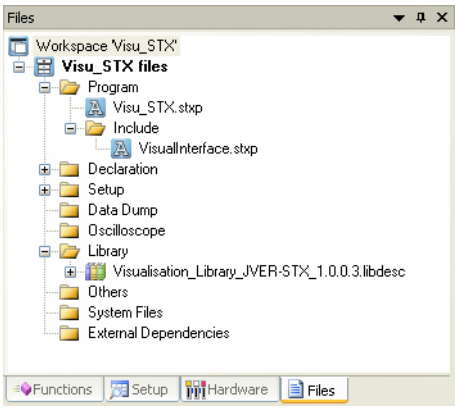
For the library with its visualization functions to be available in JetSym, you have to include it as follows:

Step	Action
1	Open the menu **Tools**. Select menu item **Library Manager**.
⇨	The dialog box of the same name opens. 
2	Click the button **Add**. **Result:** An Explorer window opens in the **Lib** folder of the JetSym installation.
3	Select the file **Visualisation_Library_1.0.0.3.libpackage** or an up-to-date version of this library.
4	Click the button **Open**. **Result:** The library file has now been integrated into the library manager. So, you can now include the library into your JetSym project.

Step	Action
5	Switch to the view **Files**. 
6	Select the folder **Library**. Open the shortcut menu by pressing the right mouse button.
7	Select menu item **Add Libraries...**. **Result:** JetSym opens the Library Manager.
8	Select the visualization library and click the button **Select**. 
⇨	

Creating a compilable program

To create and compile an executable program, proceed as follows:

Step	Action
1	Switch to the view **Files**. 
2	Double-click the program file. The program file has the same name as the project, plus the extension **stxp**. **Result:** The program file opens in the JetSym editor.

3	Enter the following program code.
	Mind this when giving the Include instruction.
	`#Include "VisualInterface.stxp";`
	
	`Function OnKeyDown(KeyCode:long, Flags:long)`
	`End_Function;`
	`Function OnKeyUp(KeyCode:long, Flags:long)`
	`End_Function;`
	
	`Task Main Autorun`
	`End_Task;`

4	Press the **[F7]** key to trigger the build process for this project. **Result:** The visualization functions and the VisualInterface header file are now available for programming.

Result:

You can expand the program now. In **IntelliSense** (**Ctrl + Space bar**), the visualization functions and the information from the VisualInterface header file are now available. You can **transfer** the program to the HMI by the shortcut **[Strg] + [F5]**. However, the program has no function as yet.

Functions `OnKeyDown` and `OnKeyUp`

The functions `OnKeyDown` and `OnKeyUp` let you trigger, for example, visualization commands when a key is pressed or released. The operating system returns a specific key code to the function depending on what key you press.

The following key codes, for example, are assigned to the keys **[UP]**, **[DOWN]**, **[OK]** and **[ESC]**:

Key	Key code	Constant
▲	0x26	KEY_UP
▼	0x28	KEY_DOWN
OK	0x0D	KEY_RETURN
ESC	0x1B	KEY_ESCAPE

The file **VISU_Defines.stxp** holds the key codes assigned to individual keys as constants. This lets you use constants in the application program.

A sample STX program is listed below:

```
#include "VisualInterface.stxp"

Function OnKeyDown(KeyCode:LONG, Flags:Long)
Case KeyCode Of
KEY_UP: VisuCmdAttribute(Ellipse_4000,
ELLIPSE_ATTR_FILLATTRIBUTE, FillAttribute_26000);
Break;
KEY_DOWN: VisuCmdAttribute(Ellipse_4000,
ELLIPSE_ATTR_FILLATTRIBUTE, FillAttribute_26001);
Break;
KEY_RETURN: VisuCmdAttribute(Ellipse_4000,
ELLIPSE_ATTR_FILLATTRIBUTE, FillAttribute_26002);
Break;
KEY_ESC: VisuCmdAttribute(Ellipse_4000,
ELLIPSE_ATTR_FILLATTRIBUTE, FillAttribute_26003);
Break;
End_Case;
End_Function;
```

Recommendations

It is advisable to use for **Ellipse_4000** and **FillAttribute_26000** object names that are more descriptive. This makes it easier to find these objects and to assign them properly. Instead of *FillAttribute_26000* you could name it, for example, *FillAttribute_White*.

Blanks or special characters (ä, ö, ü, ß, -, ...) are not allowed for object names.

JetViewSoft lets you enter object names in the properties pane of the corresponding object. JetViewSoft incorporates this object name and the object ID into the file **VisualInterface.stxp**. Then, you can use the object name and ID in the program.

Related topics

- **Initial commissioning in JetViewSoft** (see page 45)

5.3 ER-STX-CE platform - Programming

Introduction

This chapter consists of the following two parts:

- Entering data via digipot on the HMI
- Making changes to visualization objects through visualization commands (VisuCommands) from within the application program.

Prerequisites

This description applies to the platform JetView ER-STX-CE/PC.

Additional information

For more information refer to the JetSym and JetViewSoft online help.

Contents

Entering data via digipot

Introduction

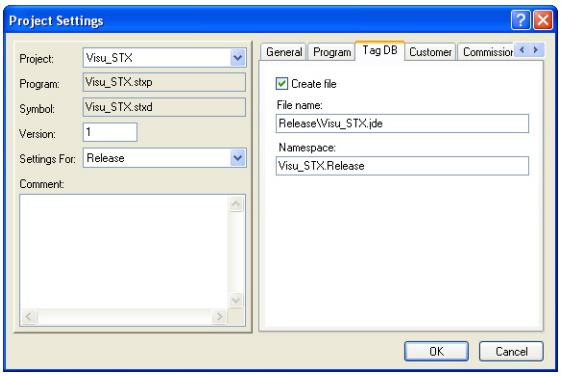
A few lines of program code enable you to enter data via digipot on the HMI. These data are then available in a JetSym STX application program.

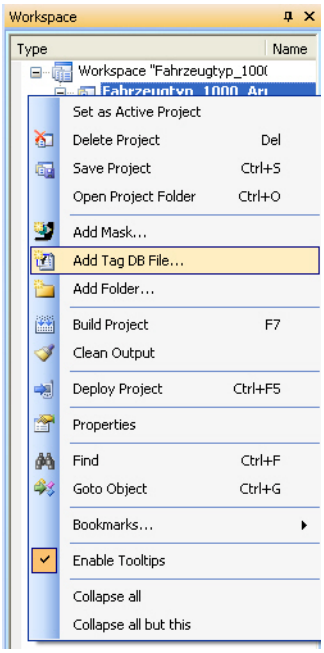
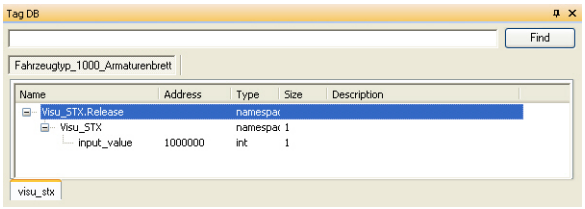
Entering data via digipot

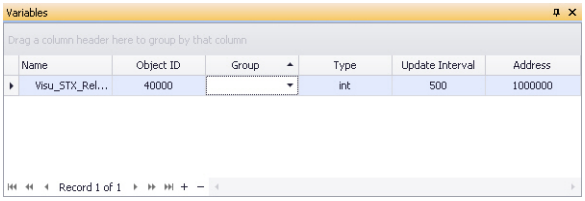
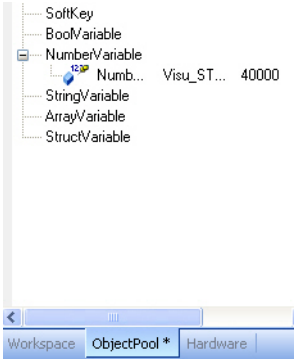
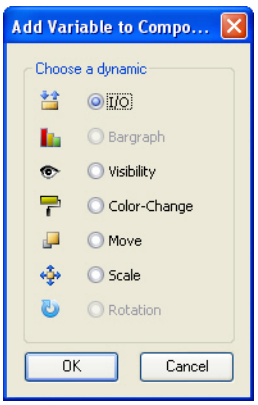
To enter data via digipot, proceed as follows:

Step	Action

1	Declare the variables in the JetSym STX program using [export jde].
	
	`Task Visu_STX Autorun`
	`Var`
	`input_value: Int At %VL 1000000 [export jde];`
	`End_Var;`
	`End_Task;`

2	In JetSym navigate to **Project Settings** and the tab **TagDB**. Here, tick the checkbox **Create file**. 
3	Open the **Build** menu. Select menu item **Build**. Another way is to press the key **[F7]**.
⇨	During the build process JetSym will then create a TagDB file with the extension *.jde. This file holds the declarations of all variables. **Storage location:** JetSym creates a folder named **Release**. JetSym stores the file to this folder.

Step	Action
4	Embed the TagDB file into your JetViewSoft visualization project. To this end, activate the shortcut menu in the workspace and click on **Add TagDB File...** . 
⇨	JetViewSoft displays the variables declared in the TagDB file in the TagDB window. 
5	Use Drag&Drop to drag a variable into the window **Variables**.

Step	Action
⇨	JetViewSoft creates a variable of the type *Number Variable* as an object. This is how the window **Variables** looks like:  This is what the tab **ObjectPool** looks like:  If the TagDB file contains controller information, JetViewSoft creates a controller and links it with the variables.
6	Use Drag&Drop to drag the variable from the window **Variables** or **ObjectPool** to the visualization object **Edit Numeric**. When the PLUS sign appears, release the mouse button. **Result:** A link between visualization object and variable has been created.
7	Select the corresponding dynamic feature, such as I/O.  **Result:** JetViewSoft adds the variable and dynamic feature to the object properties.

Step	Action
8	In JetSym and JetViewSoft, carry out the commands **Build** and **Deploy**. **Result:** The application is now available on the HMI.
9	Restart the HMI.

Result:

The HMI shows the start screen. Now, you can use the digipot to enter a value in the object **Edit Numeric**.

If you enter in the JetSym setup pane register number R 1000000, JetSym displays the set value.

Using visualization commands to manipulate visualization objects

Introduction

Visualization commands are functions included in the JetSym visualization library. These function can be invoked in the JetSym STX program. Thus, visualization commands let you manipulate visualization objects directly from within the JetSym STX program. The description below shows how to change, for example, the fill color of an ellipse using the corresponding visualization command.

Components of the visualization library

All available commands have been declared in the file **VISU_Functions.stxp**. Predefined data types, such as the color as RGB value, attributes and key codes have been declared in the file **VISU_Defines.stxp**. Both files form an integral part of the visualization library.

Prerequisites

For the compiler to compile the following program without errors, add the program code listed below to the sample programs:

```
#Include "VisualInterface.stxp";

Function OnKeyDown (KeyCode:long, Flags:long)
End_Function;
Function OnKeyUp (KeyCode:long, Flags:long)
End_Function;
```

Task 1

The application program is to control the color change of an ellipse through the fill color attribute. After 5 seconds the fill color is to change from red to blue and after another 5 seconds back from blue to red.

Task 1 - Solution

The application program invokes the function **VisuCmdAttribute()** at regular intervals (cycles).

Task 1 - JetSym STX program

```
Var
    Flag:    Bool At %MX 1;
End_Var;

Task Visu_STX Autorun
Flag := FALSE;
Loop
    If Flag = FALSE Then
        Flag := TRUE;
    ELSE
        Flag := FALSE;
    End_If;

Case Flag Of
    TRUE:  VisuCmdAttribute (Ellipse_4000,
ELLIPSE_ATTR_FILLATTRIBUTE, FillAttribute_Blue);
    Break;
    FALSE: VisuCmdAttribute (Ellipse_4000,
ELLIPSE_ATTR_FILLATTRIBUTE, FillAttribute_Red);
    Break;
```

```
End_Case;

Delay(T#5s);
End_Loop;
End_Task;
```

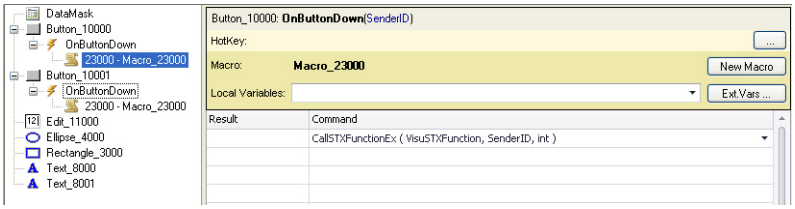
Task 2

When a certain button is activated (Button_10000), the fill color of a rectangle object is to change to red.

When a second button is activated (Button_10001), the fill color of the same rectangle object is to change to blue. The digipot is used to activate the buttons.

Task 2 - Solution

Assign in JetViewSoft the event **OnButtonDown** to both buttons. When this event occurs, the macro function **CallSTXFunctionEx()** is invoked.



In the application program, a function has been declared which in turn executes a visualization command. In the given case it is the function **VisuSTXFunction()**.

The SenderID holds the information which button has been pressed. A case instruction is used to retrieve this information.

Task 2 - JetSym-STX program

```
Function VisuSTXFunction (SenderID : long)

Case SenderID Of
    Button_10000: VisuCmdAttribute (Rectangle_3000;
RECTANGLE_ATTR_FILLATTRIBUTE, FillAttribute_Red);
    Break;
    Button_10001: VisuCmdAttribute (Rectangle_3000;
RECTANGLE_ATTR_FILLATTRIBUTE, FillAttribute_Blue);
    Break;
End_Case;
End_Function;

Task Visu_STX Autorun

End_Task;
```

Note

Once you have selected the macro function **CallSTXFunctionEX()** in a macro object, this function with its name is declared in the file **VisualInterface.stxp** as **forward**.

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, for instance, in the automobile sector.

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.

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.

Application

These STX functions are used in communication between the JVM-104 and other CANopen® nodes.

Note: Take into account the point of view!

In this chapter we use the point of view from the higher-level controller, whereas in the document CiA DS 301 the point of view from the devices is used.

This is why you need, for example, a PDO-RX macro to invoke the function `CanOpenAddPDORx()`.

Terms and abbreviations

In this chapter, the following terms and abbreviations are used:

Term	Description
Node ID	Node identification number of the device: This ID lets you address the device.
NMT	Network management - Netzwerkmanagement
ro	Read only access
rw	Read/write access

Table of contents

STX function: CanOpenInit()

Introduction

The function `CanOpenInit()` lets you initialize one of the CAN busses. The JVM-104 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 function `CanOpenInit()` has got the following parameters.

Parameter	Description	Value
CANNo	CAN bus number	0 ... CANMAX
NodeID	Node ID of the given device	1 ... 127
SWVersion	Reference to own software version This software version is entered into the index 0x100A in the object directory.	String up to 255 characters

Return value

This 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
-4	The JX2 system bus driver is activated

CANNo parameter

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

How to use this function

This function lets you initialize CAN bus 0. The JVM-104 has node ID 20 (0x14).

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

Operating principle

During initialization, the JVM-104 processes the following process steps:

Step	Description
1	First, the bootup message is sent as a heartbeat message.
2	As soon as the JVM-104 goes into **pre-operational** status, it sends the heartbeat message **pre-operational**.

Access to the object directory

If the JVM-104 is in **pre-operational** state, it lets you access the object directory using SDO.

NMT messages

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

Related topics

- **STX function CanOpenSetCommand** (see page 69)

STX function: CanOpenSetCommand()

Introduction

The function `CanOpenSetCommand()` lets you change the heartbeat status of the device itself and of all other devices (NMT slaves) on the CAN bus.

Function declaration

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

Function parameters

The function `CanOpenSetCommand()` has got the following parameters:

Parameter	Description	Value
CANNo	CAN bus number	0 ... CANMAX
iType	Command selection	See table below.

iType	Description: Value
CAN_CMD_HEARTBEAT	Only the own heartbeat status is changed. Selecting heartbeat states: CAN_HEARTBEAT_STOPPED (0x04) CAN_HEARTBEAT_OPERATIONAL (0x05) CAN_HEARTBEAT_PREOPERATIONAL (0x7F)
CAN_CMD_NMT	The heartbeat status is changed for all other devices or for a specific device on the CAN bus. Selecting heartbeat states (NMT master): CAN_NMT_OPERATIONAL (0x01) or CAN_NMT_START (0x01) CAN_NMT_STOP (0x02) CAN_NMT_PREOPERATIONAL (0x80) CAN_NMT_RESET (0x81) CAN_NMT_RESETCOMMUNICATION (0x82)
CAN_CMD_TIME_CONSUMER	This command lets you set the device to ready-to-receive state to allow time synchronization via CAN bus (CAN ID 0x100). Refer to document by CiA e.V. DS301 V402 *Selecting Synchronization*, page 59. CAN_TIME_CONSUMER_DISABLE = 0 CAN_TIME_CONSUMER_ENABLE = 1
CAN_CMD_TIME_PRODUCER	The time is published on the CAN bus. For more information on the structure refer to document DS301 by CiA e.V., CAN ID 0x100: CAN_TIME_PRODUCER_SEND = 1 (for sending TIME_OF_DAY once)

Note

The macro function `CAN_CMD_NMT_Value(NodeID, CAN_CMD_NMT)` is used to select the command 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 is to be sent to all devices on the CAN bus, use the parameter `CAN_CMD_NMT_ALLNODES(0)`.

CANNo parameter

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

Return value

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

Return value	
0	OK
-1	Error when checking parameters Command not known

How to use this function (example 1)

Task: Set the own heartbeat status to **operational**.

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

How to use this function (example 2)

Task: Set the own heartbeat status and the status of all other devices on the CAN bus to **operational**.

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

How to use this function (example 3)

Task: Set the heartbeat status of the device with the node ID 60 (0x3C) to **operational**.

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

How to use this function (example 4)

Task: Enable time synchronization via CAN bus (CAN ID 0x100).

```
Result := CanOpenSetCommand(0, CAN_CMD_TIME_CONSUMER,
CAN_TIME_CONSUMER_ENABLE);
```

How to use this function (example 5)

Task: Publish the time on the CAN bus.

```
Result := CanOpenSetCommand(0, CAN_CMD_TIME_PRODUCER,
CAN_TIME_PRODUCER_SEND);
```

STX function: CanOpenUploadSDO()

Introduction

The function `CanOpenUploadSDO()` lets you access a particular object in the object directory of the message recipient and read the value of the object.

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,              // Number of the bus line
    NodeID:Int,             // Device ID
    wIndex:Word,
    SubIndex:Byte,
    DataType:Int,           // Type of the data to be received
    // Data length for the global variable DataAddr
    DataLength:Int,
    // Global variable into which the received value is entered
    const ref DataAddr,
    ref Busy: Int,          // Status of the SDO transmission
) :Int;
```

Function parameters

The `CanOpenUploadSDO()` function has got the following parameters:

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

Return value

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

Return value	
0	OK
-1	Error in checking parameters
-2	Device in **Stop** status
-3	DataType is greater than DataLength
-4	Insufficient memory

CANNo parameter

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

DataType parameter

The following data types can be received.

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

Busy parameter

After successfully 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 function returns the number of bytes transmitted.

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 specified node ID is not responding.

If the specified node ID does not respond within 1 second, the timeout bit is set.

SDOACCESS_ILLCMD

The response to the request is invalid.

SDOACCESS_ABORT

Access to the device with the specified 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.

How to use this function

```
Result := CanOpenUploadSDO(
    0,                              // CANNo
    66,                             // NodeID
    0x100A,                         // wIndex
    0,                              // SubIndex
    CANOPEN_STRING,                 // DataType
    sizeof(var_Versionstring),      // DataLength
    var_Versionstring,              // DataAddr
    busy);                          // 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.

```
Const
    CANNo = 0;              // Number of the bus line
    NodeID_Node_0 = 10;     // Device ID of node 1
    NodeID_Node_0 = 66;     // Device ID of node 2
End_Const;

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


Task Example_UploadSDO autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

// Initializing CAN 0
CanOpenInit(CANNo,              // Number of the bus line
            NodeID_Node_0,      // Node ID
            SW_Version);        // Manufacturer's software version

// All nodes on the CAN bus are in PREOPERATIONAL state

// Request manufacturer's software version per SDO
Objectindex := 0x100A;
Subindex := 0;
```

```
Result:= CanOpenUploadSDO(CANNo,         // Number of the bus line
                 NodeID_Node_1,          // Node ID
                 Objectindex,            // wIndex
                 Subindex,               // SubIndex
                 CANOPEN_STRING,         // DataType
                 sizeof(Versionstring), // DataLength
                 Versionstring,          // DataAddr
                 busy);                  // Busy

// Checking the command for successful execution
If (Result == 0) Then

    // Waiting until communication is completed
    When SDOACCESS_FINISHED(busy) Continue;

    // Checking for errors
    If (SDOACCESS_ERROR(busy)) Then
        // Troubleshooting
    End_If;
End_If;

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

End_Task;
```

STX function: CanOpenDownloadSDO()

Introduction

The function `CanOpenDownloadSDO()` lets you access a particular object in the Object Directory of the message recipient and specify the value of the object. Data is exchanged in accordance with the SDO upload 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,              // Number of the bus line
    NodeID:Int,             // Device ID
    wIndex:Word,
    SubIndex:Byte,
    DataType:Int,           // Type of the data to be sent
    // Data length of the global variable DataAddr
    DataLength:Int,
    // Global variable holding the value to be sent
    const ref DataAddr,
    ref Busy: Int,          // Status of the SDO transmission
) :Int;
```

Function parameters

The `CanOpenDownloadSDO()` function has got the following parameters:

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

Return value

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

Return value	
0	OK
-1	Error when checking parameters
-2	Device in **Stop** status (own heartbeat status)
-3	DataType is greater than DataLength
-4	Insufficient memory

CANNo parameter

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

DataType parameter

The following data types can be received.

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

Busy parameter

After successfully 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 function returns the number of bytes transmitted.

"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 node ID does not respond within 1 second, the timeout bit is set.

SDOACCESS_ILLCMD

The response to the request is invalid.

SDOACCESS_ABORT

Access to the device with the specified 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.

How to use this function

```
Result := CanOpenDownloadSDO(
    0,                              // CANNo
    68,                             // NodeID
    0x1017,                         // wIndex
    0,                              // SubIndex
    CANOPEN_WORD,                   // DataType
    sizeof(var_Heartbeat_time),     // DataLength
    var_Heartbeat_time,             // DataAddr
    busy);                          // 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.

```
Const
    CANNo = 0;              // Number of the bus line
    NodeID_Node_0 = 10;     // Node ID of node 1
    NodeID_Node_0 = 68;     // Node ID of node 2
End_Const;

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


Task Example_DownloadSDO autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

// Initializing CAN 0
CanOpenInit(CANNo,            // Number of the bus line
            NodeID_Node_0,    // Device ID of the node
            SW_Version);      // Manufacturer's software version

// Setting the node with ID NodeID_Node_1 on the CAN bus to
// PREOPERATIONAL state
CanOpenSetCommand(CANNo, CAN_CMD_NMT_Value(NodeID_Node_1,
CAN_CMD_NMT), CAN_NMT_PREOPERATIONAL);

// Changing the heartbeat time of the addressed device via SDO
Objectindex := 0x1017;
Subindex := 0;
Result:= CanOpenDownloadSDO(CANNo,  // Number of the bus line
                  NodeID_Node_1,          // Node ID
                  Objectindex,            // wIndex
                  Subindex,               // SubIndex
                  CANOPEN_WORD,           // DataType
                  sizeof(Heartbeat_time), // DataLength
                  Heartbeat_time,         // DataAddr
                  busy);                  // Busy

// Checking the command for successful execution
If (Result == 0) Then
```

```
    // Waiting until communication is completed
    When SDOACCESS_FINISHED(busy) Continue;

    // Checking for errors
    If (SDOACCESS_ERROR(busy)) Then
    // Troubleshooting
    End_If;
End_If;

// Resetting all devices on the CAN bus to OPERATIONAL status
CanOpenSetCommand(CANNo, CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES,
CAN_CMD_NMT), CAN_NMT_OPERATIONAL);

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

End_Task;
```

STX function: CanOpenAddPDORx()

Introduction

The function `CanOpenAddPDORx()` lets you specify which process data, sent by other CANopen® devices, must be received.

Process data can be received only when a CANopen® device is sending them.

Notes

- Only if the CANopen® devices on the bus are in state **operational**, the PDO telegram is transmitted.
- 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,              // Number of the bus line
    CANID:Int,              // CAN identifier
    // Starting position of data to be received
    BytePos:Int,
    DataType:Int,           // Data type of the data to be received
    // Data length of the global variable VarAddr
    DataLength:Int,
    // Global variable into which the received value is entered
    const ref VarAddr,
    // Cycle time for receiving a telegram
    // Event time
    EventTime: Int,
    // Minimum interval between two received messages
    // Inhibit time
    InhibitTime: Int,
    Paramset: Int,          // Bit-coded parameter
) :Int;
```

Function parameters

The `CanOpenAddPDORx()` function has got the following parameters:

Parameter	Description	Value
CANNo	CAN bus number	0 ... CANMAX
CANID	CAN identifier 11-bit CAN identifier 29-bit	0 ... 0x7FF 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	Data length of the global variable VarAddr	
VarAddr	Global variable into which the received value is entered	
EventTime	Time lag between two telegrams (> InhibitTime)	

Parameter	Description	Value
InhibitTime	Minimum time lag between two telegrams received (< EventTime)	
Paramset	Bit-coded parameter	

Return value

This function sends 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

CANNo parameter

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

Note: Take into account the point of view!

In this chapter we use the point of view from the higher-level controller, whereas in the document CiA DS 301 the point of view from the devices is used.

This is why you need a PDO-RX macro to invoke the function `CanOpenAddPDORx()` from the controller.

CANID parameter

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)	Description
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			

DataType parameter

The following data types can be received.

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

Paramset parameter

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 to the sender (after each expired EventTime). If there is no response to RTR frames, the request time increases to five times the EventTime.

CANOPEN_ASYNCPDO

Receive asynchronous PDOs.

CANOPEN_PDOINVALID

PDO not received. Disk space is reserved.

CANOPEN_NORTR

PDO cannot be requested by RTR (Remote Request).

Only if CANOPEN_ASYNCPDORTROnly has been set, an RTR is sent.

CANOPEN_29BIT

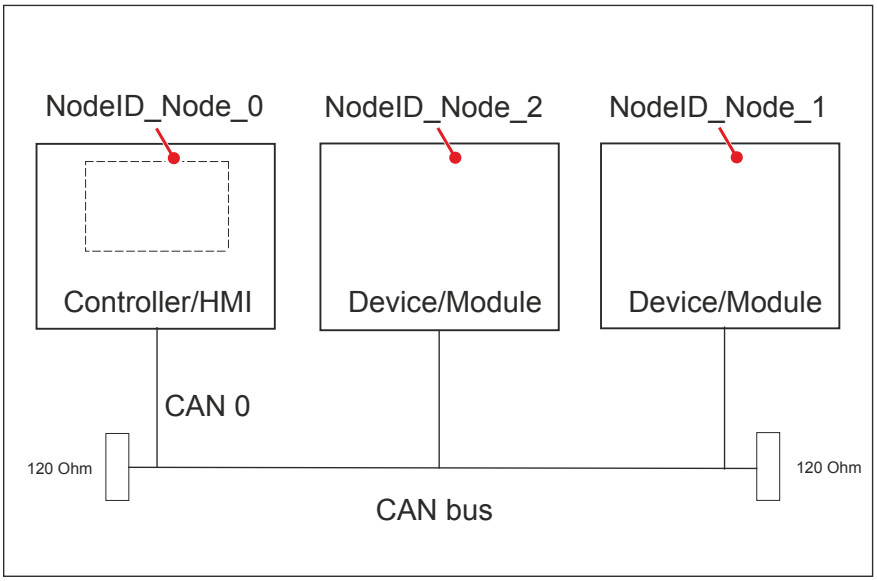
Use 29-bit identifier
Default: 11-bit identifier

How to use this function

```
Result := CanOpenAddPDORx(
    0,                                  // CANNo
    662,                                // CANID
    0,                                  // BytePos
    CANOPEN_DWORD,                      // DataType
    sizeof(var_Data_1_of_Node_1),       // DataLength
    var_Data_1_of_Node_1,               // VarAddr
    1000,                               // Event time
    10,                                 // Inhibit time
    CANOPEN_ASYNCPDO | CANOPEN_NORTR);  // Paramset
```

JetSym STX program

JVM-104 with node ID 10 wants to receive a PDO from two CANopen® devices with node ID 64 and 102. The function `CanOpenAddPDORx()` is invoked for this purpose. After running the program, the JVM-104 receives cyclic PDO telegrams.




```
Const
    CANNo = 0;              // Number of the bus line
    NodeID_Node_0 = 10;     // Device ID of node 1
    NodeID_Node_1 = 64;     // Device ID of node 2
    NodeID_Node_2 = 102;    // Device ID of node 3
    Event_Time = 1000;      // Event time in ms
    Inhibit_Time = 10;      // Inhibit time in ms
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';

// Initializing CAN 0
CanOpenInit(CANNo,            // Number of the bus line
            NodeID_Node_0,    // Node ID
            SW_Version);      // Manufacturer's software version

// Entering process data to be received
CanOpenAddPDORx(
    CANNo,                              // Number of the bus line
    CANOPEN_PDO2_RX(NodeID_Node_1),     // CANID
    0,                                  // BytePos
    CANOPEN_DWORD,                      // DataType
    sizeof(Data_1_of_Node_1),           // DataLength
    Data_1_of_Node_1,                   // VarAddr
    Event_Time,                         // Event time
    Inhibit_Time,                       // Inhibit time
    CANOPEN_ASYNCPDORTRONLY);           // Paramset

CanOpenAddPDORx(
    CANNo,                              // Number of the bus line
    CANOPEN_PDO2_RX(NodeID_Node_1),     // CANID
    4,                                  // BytePos
    CANOPEN_DWORD,                      // DataType
    sizeof(Data_2_of_Node_1),           // DataLength
    Data_2_of_Node_1,                   // VarAddr
    Event_Time,                         // Event time
    Inhibit_Time,                       // Inhibit time
    CANOPEN_ASYNCPDORTRONLY);           // Paramset
```

```
CanOpenAddPDORx(
    CANNo,                                  // Number of the bus line
    CANOPEN_PDO3_RX(NodeID_Node_2),         // CANID
    0,                                      // BytePos
    CANOPEN_BYTE,                           // DataType
    sizeof(Data_1_of_Node_2),               // DataLength
    Data_1_of_Node_2,                       // VarAddr
    Event_Time,                             // Event time
    Inhibit_Time,                           // Inhibit time
    CANOPEN_ASYNCPDO | CANOPEN_NORTR); // Paramset

// All nodes on the CAN bus are in PREOPERATIONAL state

// Setting all nodes on the CAN bus to OPERATIONAL state
CanOpenSetCommand(CANNo, CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES,
CAN_CMD_NMT), CAN_NMT_START);

// As of now, PDO telegrams are received/sent
// from the higher-level controller
//      ...
//      ...
//      ...

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

- Only if the CANopen® devices on the bus are in state **operational**, the PDO telegram is transmitted.
- 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,            // Number of the bus line
    CANID:Int,            // CAN identifier
    BytePos:Int,          // Starting position of the data to be sent
    DataType:Int,         // Data type of the data to be sent
    // Data length of the global variable VarAddr
    DataLength:Int,
    // Global variable holding the value to be sent
    const ref VarAddr,
    // Cycle time for sending a telegram
    // Event time
    EventTime: Int,
    // Minimum interval between two transmitted messages
    // Inhibit time
    InhibitTime: Int,
    Paramset: Int,        // Bit-coded parameter
) :Int;
```

Function parameters

The `CanOpenAddPDOTx()` function has got the following parameters:

Parameter	Description	Value
CANNo	CAN bus number	0 ... CANMAX
CANID	CAN identifier 11-bit CAN identifier 29-bit	0 ... 0x7FF 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	Data length of the global variable VarAddr	
VarAddr	Global variable into which the value to be sent is entered	

Parameter	Description	Value
EventTime	Time lag between two telegrams (> InhibitTime)	
InhibitTime	Minimum time lag between two telegrams to be sent (< EventTime)	
Paramset	Bit-coded parameter	

Return value

This function sends 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

CANNo parameter

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

CANID parameter

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)	Description
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			

DataType parameter

The following data types can be received.

Byte types	CANopen® format	Jetter format
1	CANOPEN_INTEGER8 CANOPEN_UNSIGNED8	Byte
2	CANOPEN_INTEGER16 CANOPEN_UNSIGNED16	Word
3	CANOPEN_INTEGER24 CANOPEN_UNSIGNED24	-
4	CANOPEN_INTEGER32 CANOPEN_UNSIGNED32 CANOPEN_REAL	Int
5	CANOPEN_INTEGER40 CANOPEN_UNSIGNED40	-
6	CANOPEN_INTEGER48 CANOPEN_UNSIGNED48 CANOPEN_TIME_OF_DAY CANOPEN_TIME_DIFFERENCE	-
7	CANOPEN_INTEGER56 CANOPEN_UNSIGNED46	-
8	CANOPEN_INTEGER64 CANOPEN_UNSIGNED64 CANOPEN_REAL64	-

Byte types	CANopen® format	Jetter format
n	CANOPEN_VISIBLE_STRING CANOPEN_OCTET_STRING CANOPEN_UNICODE_STRING CANOPEN_DOMAIN	String

Paramset parameter

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.
This feature is not yet supported at the moment.

CANOPEN_ASYNCPDO
Send asynchronous PDO.

CANOPEN_PDOINVALID
PDO not sent. The required disk space is reserved.

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

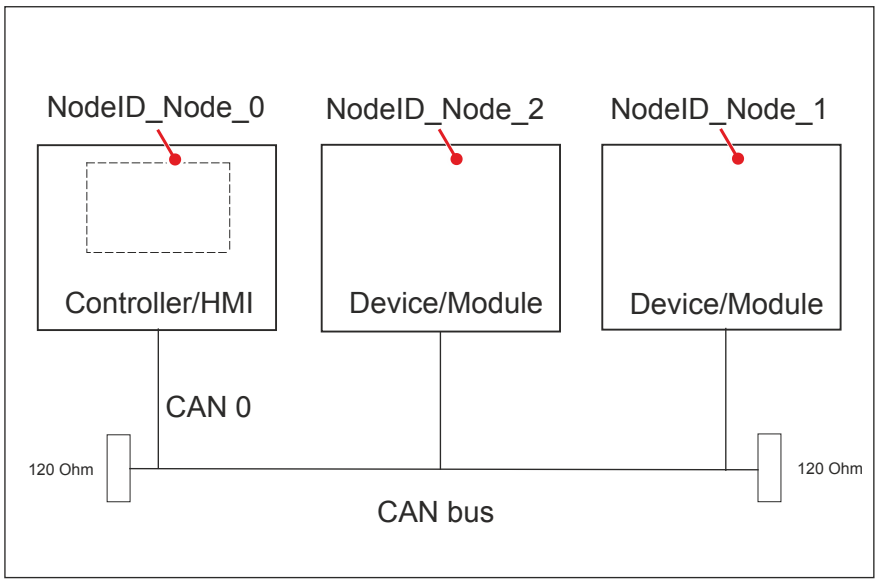
CANOPEN_29BIT
Use 29-bit identifier
Default: 11-bit identifier

How to use this function

```
Result := CanOpenAddPDOTx(
    0,                                  // CANNo
    842,                                // CANID
    0,                                  // BytePos
    CANOPEN_DWORD,                      // DataType
    sizeof(var_Data_1_of_Node_3),       // DataLength
    var_Data_1_of_Node_3,               // VarAddr
    1000,                               // Event time
    100,                                // Inhibit time
    CANOPEN_ASYNCPDO | CANOPEN_NORTR);  // Paramset
```

JetSym STX program

JVM-104 sends process data to two CANopen® devices with the node ID 74 and 112. After running the program and in the case of changes, the JVM-104 sends cyclic PDO telegrams every 3,000 ms (event time). As a maximum, the PDO telegram is sent every 10 ms (inhibit time).




```
Const
    CANNo = 0;              // Number of the bus line
    NodeID_Node_0 = 10;     // Device ID of node 1
    NodeID_Node_1 = 74;     // Device ID of node 2
    NodeID_Node_2 = 112;    // Device ID of node 3
    Event_Time = 3000;      // Event time in ms
    Inhibit_Time = 100;     // Inhibit time in ms
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';

// Initializing CAN 0
CanOpenInit(CANNo,              // Number of the bus line
            NodeID_Node_0,      // Node ID
            SW_Version);        // Manufacturer's software version
```

```
// Send data per PDO
CanOpenAddPDOTx(
    CANNo,                              // Number of the bus line
    CANOPEN_PDO2_TX(NodeID_Node_1),     // CANID
    0,                                  // BytePos
    CANOPEN_DWORD,                      // DataType
    sizeof(Data_1_of_Node_1),           // DataLength
    Data_1_of_Node_1,                   // VarAddr
    Event_Time,                         // Event time
    Inhibit_Time,                       // Inhibit time
    CANOPEN_ASYNCPDORTRONLY);           // Paramset

CanOpenAddPDOTx(
    CANNo,                              // Number of the bus line
    CANOPEN_PDO2_TX(NodeID_Node_1),     // CANID
    4,                                  // BytePos
    CANOPEN_DWORD,                      // DataType
    sizeof(Data_2_of_Node_1),           // DataLength
    Data_2_of_Node_1,                   // VarAddr
    Event_Time,                         // Event time
    Inhibit_Time,                       // Inhibit time
    CANOPEN_ASYNCPDORTRONLY);           // Paramset

CanOpenAddPDOTx(
    CANNo,                                  // Number of the bus line
    CANOPEN_PDO3_TX(NodeID_Node_2),         // CANID
    0,                                      // BytePos
    CANOPEN_BYTE,                           // DataType
    sizeof(Data_1_of_Node_2),               // DataLength
    Data_1_of_Node_2,                       // VarAddr
    Event_Time,                             // Event time
    Inhibit_Time,                           // Inhibit time
    CANOPEN_ASYNCPDO | CANOPEN_NORTR);      // Paramset

// All nodes on the CAN bus are in PREOPERATIONAL state

// Setting all nodes on the CAN bus to OPERATIONAL state
CanOpenSetCommand(CANNo, CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES,
CAN_CMD_NMT), CAN_NMT_START);

// As of now, PDO telegrams will be transmitted from the devices with
node ID 74
// and 112.
//      ...

End_Task;
```

Heartbeat monitoring

Introduction

The heartbeat protocol is for monitoring the activity of communication partners. If the inactivity exceeds the set interval (Heartbeat consumer time), the status is set to **offline**.

The application program lets you define heartbeat functions, such as

- Displaying information to the user
- Rebooting the device
- Ignoring process data

Prerequisites

Heartbeat monitoring is available only for specific devices and its availability depends on the OS version.

Device	OS version
FMC-01	1.18.1.00 or higher
JVM-C02	4.00.0.00 or higher
JCM-350	1.09.0.215 or higher
JCM-620	JVER bersion 3.2.2.645 and JetVM version 3.04.0.00 or higher

Registers for heartbeat monitoring

Heartbeat monitoring uses the following registers:

Register	Description	Data type	Attributes
40x001	Own heartbeat status of the device; Value range: 0 = Bootup 4 = Stopped 5 = Operational 127 = Preoperational 255 = Offline (default value)	Int	ro (read only)
40x100	The heartbeat status of all monitored node IDs has changed. Value range: 0 = False 1 = True	Bool	rw (read and write)
40x101 ... 40x227	Heartbeat status of nodes with ID 1 ... 127; value range: 0 = Bootup 4 = Stopped 5 = Operational 127 = Preoperational 255 = Offline (default value)	Byte	ro
40x229 ... 40x355	Heartbeat timeout of nodes with ID 1 ... 127; value range: 0 ... 65535 [ms]	Word	rw

In the register number, the letter **x** represents the number of the CAN bus line used: x = 0 ... CANMAX.

Launching heartbeat monitoring

To launch heartbeat monitoring, proceed as follows:

Step	Action
1	Enable heartbeat monitoring: Enter the timeout value into the corresponding register. This value must range between 1 and 65535 [ms]. Example: For CAN 0 and node ID 1: Enter a timeout value of 3000 [ms] into register 400229.
2	Define in your application program how the device is to respond to individual values in the heartbeat status register. When the state in register 40x101 ... 40x227 changes, the value in register 40x100 changes to 1 (true).
3	Reset the value in register 40x100 to zero (false). This step ensures that subsequent changes in register 40x101 ... 40x227 can be displayed.

Heartbeat monitoring starts on receipt of the first heartbeat (including bootup message). The DLC (Data Length Code) of the heartbeat message must be 1.

Terminating heartbeat monitoring

To terminate heartbeat monitoring, proceed as follows:

Step	Action
1	Disable heartbeat monitoring: Enter a timeout value of 0 [ms] into the timeout register.

Emergency message

When a heartbeat timeout is detected, an emergency message is sent automatically.

On receipt of the next heartbeat message, the emergency message is reset.

Example:

The following emergency message is tripped:

Reference	Value
Error code	0x8130
Error Register	0x81
Manufacturer error	0x00,NodeID,0x00,0x00,0x00

The message on the CAN bus looks as shown below:

- Own NodeID 5
- Monitored NodeID 1
- ID: 0x85 DLC = 8 Data: 0x30 0x81 0x81 0x00 0x01 0x00 0x00 0x00

Emergency message Rx

The declaration of the emergency message Rx consists of the following elements:

```
CanOpenAddEmergencyRx(
    CANNo:Int,          // Number of the bus line
    NodeID:Int,         // Node ID
    // Status, number of valid messages
    ref stCanOpenEmergencyStat:CanOpenEmergencyStat,
    // Array holding the emergency messages
    ref CanOpenEmergencyMSG:CanOpenEmergencyArray,
    ):int
```

Example:

The above program lines must be included into the corresponding tasks of your application program. The example below shows an emergency message from a device with node ID 21.

```
...
// Initializing the CAN bus once.

...

// Defining global variables
Var
    stCanOpenEmergencyMsg : ARRAY[5] of CanOpenEmergencyMsg;
    stCanOpenEmergencyStat : CanOpenEmergencyStat;
End_Var;

stCanOpenEmergencyStat.lBuffer := sizeof(stCanOpenEmergencyMsg);
iRet:= CanOpenAddEmergencyRx(0,                         // CANNo.
                             21,                        // NodeID
                             stCanOpenEmergencyStat,    // Status
                             stCanOpenEmergencyMsg);    // Array

...
```

The above program lines produce the following result:

When the device with node ID 21 receives an emergency message, the value in register 400100 switches from 0 to 1 (true).

Reset this value always to 0 (false). In doing so, you make sure that new emergency messages are displayed.

Emergency message Tx

The declaration of the emergency message Tx consists of the following elements:

```
CanOpenAddEmergencyTx(
    // Number of the bus line
    CANNo:int,
    // For error code see CiA DS 301 V4.02 page 60
    // or CiA DS 4xx (device profile)
    ErrorCode:word,
    // Error register (object 0x1001)
    ErrorRegister:byte,
    // 5 bytes can be used at the user's discretion
    ManufacturerArray:ByteArray5,
    // True = An error has occurred
    // False = Error has been cleared (acknowledged)
    bSet:bool
    ):Int;
```

CANopen® object dictionary for JVM-104

Supported objects

The operating system of JVM-104 supports the following objects:

Index (hex)	Object (code)	Object name	Type	Attributes
1000	VAR	Device type	Unsigned32	ro (read only)
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 (read & write)
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	Subindex	Default	Description
0x1000	0	0x0000012D	Device type (read-only)

Error Register object (index 0x1001)

The function `CanOpenAddEmergencyTx()` lets you set the bits in this register.

The structure of the **Error Register object** is shown in the following table.

Index	Subindex	Default	Description
0x1001	0	0	Error register (read-only)

This object implements the CANopen® error register functionality.

The following error messages may appear:

- Bit 0 = Generic error
- Bit 1 = Current error
- Bit 2 = Voltage error
- Bit 3 = Temperature error
- Bit 4 = Communication error (overrun, error state)
- Bit 5 = Specific device profile error
- Bit 6 = Reserved (always 0)
- Bit 7 = Manufacturer-specific error

Pre-defined Error Field object (index 0x1003)

The structure of the **Pre-defined Error Field object** is shown in the following table.

Index	Subindex	Default	Description
0x1003	0	0	Number of errors entered in the array's standard error field
	1	0	Most recent error 0 indicates no error
	2 ... 254	-	Earlier errors

This object shows a history list of errors that have been detected by the JVM-104. The maximum length of the list is 254 errors. The list content is deleted on restart.

Composition of 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	Subindex	Default	Description
0x1008	0	JVM-104	Hardware name

Manufacturer Hardware Version object (index 0x1009)

The structure of the **Manufacturer Hardware Version object** is shown in the following table.

Index	Subindex	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	Subindex	Default	Description
0x100A	0		Software version of the application program that runs on the JVM-104

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	Subindex	Default	Description
0x100B	0		Node ID of the given device

Producer Heartbeat Time object (index 0x1017)

The structure of the **Producer Heartbeat Time object** is shown in the following table.

Index	Subindex	Default	Description
0x1017	0	1,000 [ms]	Heartbeat time

CANopen® registers - JVM-104

The table below lists the JVM-104 registers associated with the CANopen® Object Dictionary.

The letter x in the register number represents the CAN bus number ranging from 0 ... CANMAX.

Register number	Description	Value range	Attributes	Data type
40x000	Own node ID	1 ... 127	rw (read & write)	Int
40x001	Own heartbeat status	0 = Bootup 4 = Stopped 5 = Operational 127 = Preoperational 255 = Offline	ro (read only)	Int
40x002		Refer to object 0x1001	ro	Int
40x019			ro	Int (IP format)

Register number	Description	Value range	Attributes	Data type
40x020			rw	Int
40x021			rw	Int
40x022			rw	Int
40x023			rw	Int
40x030			rw	Int
40x100			rw	bool
40x400			rw	bool
40x101 ... 40x227	Node ID 1 ... 127 Status	0 = Bootup 4 = Stopped 5 = Operational 127 = Preoperational 255 = Offline (default)	ro	byte
40x229 ... 40x355	Node ID 1 ... 127 timeout	0 ... 65535 ms	rw	word

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-104 and other ECUs in the vehicle. As a rule, engine data, such as RPM, speed or coolant temperature are read and displayed.

Documentation

The key SAE J1939 specifications are:

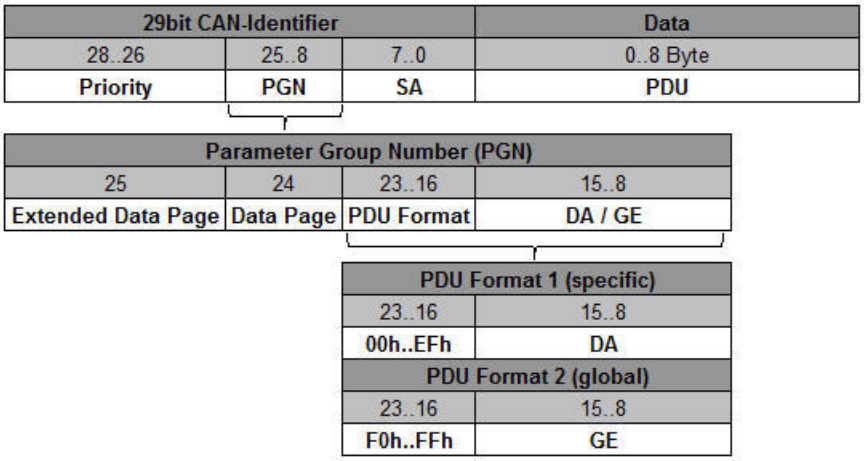
- J1939-11 - Information on the physical layer
- J1939-21 - Information on the data link layer
- J1939-71 - Information on the application layer vehicles
- J1939-73 - Information on the application layer range analysis
- J1939-81 - Network management

Contents

Content of a J1939 message

Content of a J1939 message

The following diagram shows the structure 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 - ET1 **Engine Temperature 1**

Part of the PGN	Value	Comment
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®) for use with the J1939 protocol. From then on, the JVM-104 has got the SA (Source Address) assigned by the function parameter mySA. Thus, it has got its own device address on the bus.

Function declaration

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

Function parameters

The `SAEJ1939Init()` function comprises 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

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

How to use this function

Initializing CAN bus 1. The JVM-104 has got Node SA 20 (0x14). The JVM-104 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

The function `SAEJ1939SetSA()` lets you change the own SA (Source Address) during runtime.

Function declaration

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

Function parameters

The function SAEJ1939SetSA() comprises the following parameters:

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
mySA	New SA	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

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

How to use this function

Changing the SA during runtime.

Result := SAEJ1939SetSA(1, 20);

Important note!

Messages are immediately sent/received using the new SA.

STX function SAEJ1939GetSA()

Introduction

The function `SAEJ1939GetSA()` lets you determine your own SA (Source Address).

Function declaration

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

Function parameters

The function `SAEJ1939GetSA()` comprises the following parameters:

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
mySA	SA currently set	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

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

How to use 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-104 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()` comprises the following parameters:

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
IPGN	PGN 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 (> InhibitTime)	Default value: 1,000 ms
InhibitTime	Minimum time lag between two telegrams received (< EventTime)	Default value: 100 ms

Return value

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

Return value	
0	OK
-1	Error when checking parameters

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

DataType parameter

All allowed data types are listed below:

Byte types	Bit types	
1	-	SAEJ1939_UNSIGNED8 SAEJ1939_BYTE
2	-	SAEJ1939_UNSIGNED16 SAEJ1939_WORD
4	-	SAEJ1939_UNSIGNED32 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;
```

How to use 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-104 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 for capturing the fuel temperature has got SA 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);

// Receiving the fuel temperature value
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 JVM-104 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()` comprises the following parameters:

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
IPGN	PGN 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 (> InhibitTime)	Default value: 1,000 ms
InhibitTime	Minimum time lag between two telegrams received (< EventTime)	Default value: 100 ms

Return value

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

Return value	
0	OK
-1	Error when checking parameters

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

DataType parameter

All allowed data types are listed below:

Byte types	Bit types	
1	-	SAEJ1939_UNSIGNED8 SAEJ1939_BYTE
2	-	SAEJ1939_UNSIGNED16 SAEJ1939_WORD
4	-	SAEJ1939_UNSIGNED32 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;
```

How to use 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
// Setting 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 terminated only if 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 constantly be recalled in cycles.

Function declaration

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

Function parameters

The function `SAEJ1939RequestPGN()` comprises the following parameters:

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

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

DataType parameter

All allowed data types are listed below:

Byte types	Bit types	
1	-	SAEJ1939_UNSIGNED8 SAEJ1939_BYTE
2	-	SAEJ1939_UNSIGNED16 SAEJ1939_WORD
4	-	SAEJ1939_UNSIGNED32 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

How to use this function

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

JetSym STX program

JVM-104 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;

// Requesting 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 constantly be recalled in cycles.

Function declaration

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

Function parameters

The function `SAEJ1939GetDM1()` comprises the following parameters:

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
bySA	Source Address of message sender	0 ... 253 The own SA cannot be used
stJ1939DM1stat	lStatus lMsgCnt lBuffer	Lamp status Number of received messages Size of variable stJ1939DM1msg
stJ1939DM1msg	lSPN byOC byFMI	Error code Error counter Error type

Return value

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

Return value	
0	OK
-1	Error when checking parameters

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

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:
ISPN = 0
byOC = 0
byFMI = 0
For older controllers (grandfathered setting):
ISPN = 524287 (0x7FFFF)
byOC = 31 (0x1F)
byFMI = 127 (0x7F)

How to use this function

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

JetSym STX program

By calling up the function `SAEJ1939GetDM1()`, the JVM-104 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;

// Requesting 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 ones (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()` comprises the following parameters:

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
bySA	Source Address of message sender	0 ... 253 The own SA cannot be used
stJ1939DM2stat	lStatus lMsgCnt lBuffer	Lamp status Number of received messages Size of variable stJ1939DM2msg
stJ1939DM2msg	lSPN byOC byFMI	Error code Error counter Error type

Return value

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

Return value	
0	OK
-1	Error when checking parameters

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

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:
ISPN = 0
byOC = 0
byFMI = 0
For older controllers (grandfathered setting):
ISPN = 524287 (0x7FFFF)
byOC = 31 (0x1F)
byFMI = 127 (0x7F)

How to use this function

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

JetSym STX program

By calling up the function `SAEJ1939GetDM2()`, the JVM-104 requests the current diagnostics error codes (PGN 65227).

```
#Include "SAEJ1939.stxp"

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

Task main autorun
// 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
     Int;
End_Var;

// Requesting 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, this function lets you specify the conversion method.

Function declaration

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

Function parameters

The function `SAEJ1939SetSPNConversion()` comprises 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 4: Automatic detection 2: Default

Return value

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

Return value	
0	OK
-1	Error when checking parameters

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

How to use this function

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

STX Function SAEJ1939GetSPNConversion()

Introduction

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

Function declaration

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

Function parameters

The function `SAEJ1939GetSPNConversion()` comprises 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 4: Automatic detection 2: Default

Return value

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

Return value	
0	OK
-1	Error when checking parameters

CANNo parameter

This parameter specifies the number of the SAEJ1939 interface. CANNo = 1 is assigned to the first interface. The number of SAEJ1939 interfaces depends on the device. For information on the maximum number of SAEJ1939 interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

How to use this function

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

8 File system

Introduction

This chapter describes the file system of the JVM-104. The file system lets you access files located on the internal flash disk.

When problems occur, a good understanding of the file system is very helpful.

Note

Exercise extreme caution when dealing with the file system, at least with system files. Failure to do so may render your device inoperative. It may even refuse to boot.

Some files may be protected against read/write access or deletion. This is normal behavior. Some of these files are virtual files, such as firmware images, or protected files, such as EDS files.

File categories

The files of the file system are categorized as follows:

- System directories or system files used by the operating system
- Files which are at the user's disposal

Contents

8.1 Directories

System directories

The system directories cannot be deleted. System directories even survive formatting.

Directory	Description
\System	■ System configuration ■ System information ■ Splash screen (boot image) ■ Screenshot
\App	■ Directory for applications
\Data	■ Directory for data
\Windows	■ Windows CE system directory
\	■ RAM disk drive

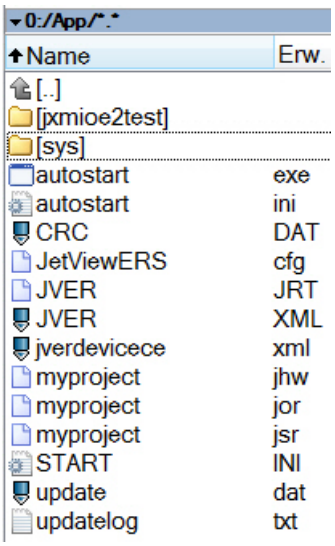
Contents

Directories

Directory *\App*

\App

This directory holds application and visualization data. In this example, the STX application is stored in the folder **jxmioe2test.**



The data is based on the new CE platform. This platform does not use *.iop files. JetViewSoft creates several visualization files instead.

Note:
Copy all application and visualization files to the folder **App** and not to the folder **Data**. Failure to do so will slow down the boot process, see directory **\Data.**

start.ini

This text file defines which application will be started.

\App\sys

This directory holds the interpreter of the STX programming language and of the visualization software. **Do not make any changes here!**

autostart.xxx

This application lets you update the operating system. **Do not make any changes here!**

updatelog.txt

This is a log file which is created during an OS update.

Directory *Data*

Data

This directory holds the HMI's bulk data. The HMI lets you store parameter or configuration files to this directory.

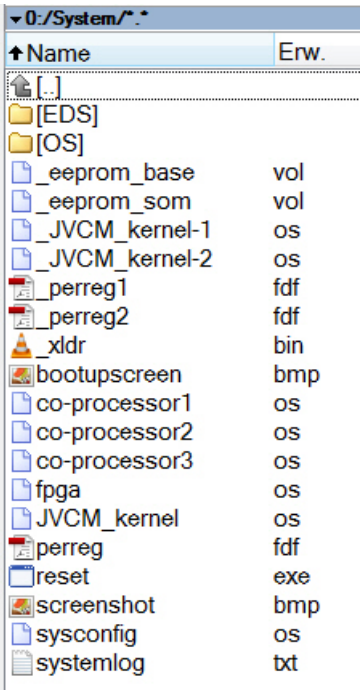
Important Note!

Larger amounts of data can be stored to this data partition. To speed up system launch, this partition will be mounted a short instance, if needed, after launching the STX application. Therefore, the STX application must not be stored to this partition.

Directory *System*

System

This directory holds system-relevant files, such as the kernel, co-processor firmware, configuration data, EDS, etc.



	bootupscreen.bmp
	This file is a 16-bit bmp file (r5, g6, b5) which is displayed while the device is booting. You may create an image of your own and replace this file.
	co-processor1
	This virtual file holds the firmware of a hidden co-processor controlling most of the interactions with the user (buttons, buzzer, background lighting, etc.).
	reset.exe
	If you delete this file, the HMI reboots immediately. You can use this function in batch files, for example, which, after complete processing, require automatic rebooting.
Directory *Windows*	***\Windows***
	This subdirectory holds the Windows CE files. **Do not make any changes here!**

8.2 Properties

Introduction

This chapter describes the properties of the file system on the internal flash disk.

General properties

The following conventions apply to the internal flash disk:

- 8 files max. to be opened simultaneously
- When the HMI creates a file, it assigns its date and time.
- Date, time, and file size are not available for all system files.

Contents

Flash disk - Properties

Capacity

The following disk space is available to the user:

Parameter	Value
Flash disk capacity	512 MBytes
Size of folder **App** (of the a.m. capacity)	64 MBytes
Size of folder **Data** (of the a.m. capacity)	368 MBytes

Properties

The internal flash disk drive has got the following further properties:

- Up to 7 directory levels and 1 file level are allowed.
- Upper- and lower-case are not distinguished.
- Directory and file names are permitted to have a total length of 63 characters.
- All characters except "/" and ".." are permitted for directory and file names
- The location of the folders **App** and **Data** is on the flash disk drive.

9 Programming

Purpose of this chapter

This chapter is for supporting you in programming the HMI JVM-104 in the following fields of activity:

- Programming additional functions

Prerequisites

To be able to program the HMI JVM-104 the following prerequisites must be fulfilled:

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

Contents

Abbreviations, module register properties and formats

Abbreviations

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

Abbreviation	Description
R 100	Register 100
MR 150	Module register 150

Module register properties

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

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

Numerical formats

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

Notation	Numerical format
100	Decimal
0x100	Hexadecimal
0b100	Binary

JetSym sample programs

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

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

9.1 Memories - Overview

Introduction

The JVM-104 features several types of program and data memories. There is, for example, volatile memory. Volatile memory loses its content at switching off. Non-volatile memory keeps its content even when the power supply is off.

This chapter gives an overview of the available memory.

Contents

Operating system memory

Introduction	The OS is stored to a non-volatile flash memory in the CPU. Therefore, the OS can be executed immediately after the device is powered up.
Properties	▪ Internal flash memory for storing OS data ▪ Internal volatile RAM for storing OS data
Memory access	▪ The user is not allowed to directly access the OS memory. ▪ To modify the OS, it must be updated.
Related topics	▪ **OS update** (see page 162)

File system memory

Introduction	The file system memory is for storing data and program files.
Properties	▪ Non-volatile ▪ Internal flash disk size: 368 MBytes
Memory access	▪ By operating system ▪ By JetSym ▪ By means of file commands from within the application program

Application program memory

Introduction	By default, the application program (STX script) is uploaded from JetSym to the HMI and is stored to it.
Properties	▪ Stored as file within the file system ▪ Default directory "\app\program name" ▪ Files may also be stored to other directories
Memory access	▪ By operating system ▪ By JetSym ▪ By means of file commands from within the application program
Related topics	▪ **Application program** (see page 165)

Memory for volatile application program variables

Introduction

Volatile variables are used to store data which need not be maintained when the JVM-104 is de-energized.

Properties

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

Memory access

- By JetSym
- From the application program

JetSym STX program

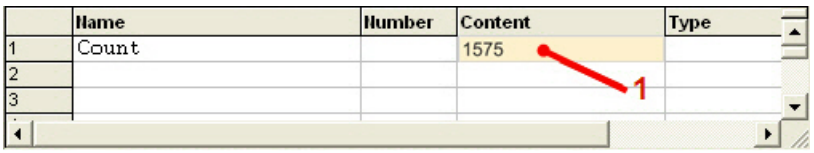
The following program increments the content of a global variable by one every 2 s.

```
Var
    Count:    Int;
End_Var;

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

Setup pane

The JetSym setup pane displays the content of the variable.



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

Memory for non-volatile application program registers

Introduction

Non-volatile registers are used to store data which must be maintained when the JVM-104 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: 30,000
- Register numbers: 1000000 ... 1029999

Memory access

- By JetSym
- From HMIs
- From the application program
- From other controllers

JetSym STX program

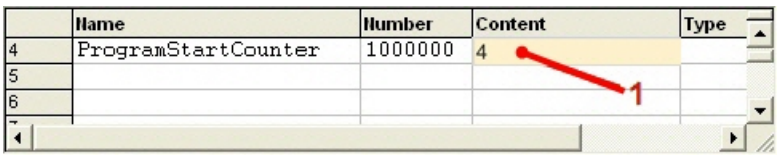
The following program increments the content of a register variable every time the application program is started. This way, the number of program starts is counted.

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

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

Setup pane

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



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

Memory for non-volatile application program variables

Introduction

Non-volatile variables are used to store data which must be maintained when the JVM-104 is de-energized.

Properties

- Global variables which are assigned to permanent registers (%RL)
- Variables are stored in a compact way.
- Size: 120,000 bytes
- Register numbers: 1000000 ... 1029999

Memory access

- By JetSym
- From HMIs
- From the application program

JetSym STX program

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

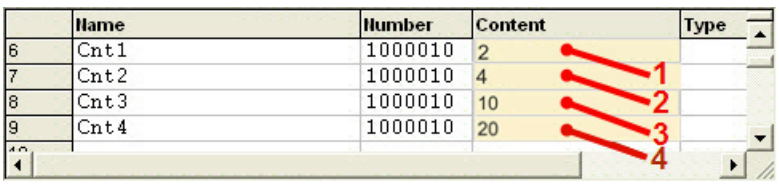
The working range of the counters is between 0 and 255 (variable type: byte). For these four variables the four bytes of register 1000010 are used.

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

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

Setup pane

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



Number	Description	Function
1	Current content of the variable Cnt1	The content of the variable is incremented by one every second.
2	Current content of the variable Cnt2	The content of the variable is incremented by two every second.
3	Current content of the variable Cnt3	The content of the variable is incremented by five every second.
4	Current content of the variable Cnt4	The content of the variable is incremented by ten every second.

Special registers

Introduction

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

Properties

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

Memory access

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

JetSym STX program

The following program uses the special register to store the digipot value. In this program, the background lighting for the JVM-104 is dimmed by using the digipot. An upper and lower limit for the digipot is specified for this purpose. If you press the pushbutton, full background lighting is activated.

```
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;
```

Flags

Introduction

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

Properties of user flags

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

Properties of overlaid user flags

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

Properties of special flags

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

Memory access

- By JetSym
- From HMIs
- From the application program

JetSym STX program

In the program listed below, a flag is set when the user presses KEY_UP. If KEY_DOWN is pressed, the flag is reset. As long as this flag is set, special register 364000 (background lighting) is incremented. Incrementing of the special register continues until the flag is reset.

```
Var
    Flag1:              Bool At %MX 1;
    Key_Up:               Bit  At %XL 361000.3;
    Key_Down:             Bit  At %XL 361000.2;
    Background_Light:     Int  At %VL 364000;
End_Var;

Task Main Autorun
    Flag1:= False;
    Loop
        If Key_Up Then
            Flag1 := True;
        ElseIf Key_Down Then
            Flag1 := False;
        End_IF;
```

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

9.2 Controls and ignition

Introduction

This chapter covers the programming of controls, ignition and switching off delay for the JVM-104.

Contents

Input keys

Introduction

The HMI JVM-104 has got four input keys: **[UP]**, **[DOWN]**, **[OK]** and **[ESC]**. These input keys are user-programmable.

Special registers

In register 361000 of the JVM-104, there is a bit-coded map of the input keys which can be used for programming them.

The following registers are available for programming these input keys:

Register	Description
361000	Bit-coded map of the input keys
361000.0	**Input key [OK]** Bit 0 = 1: Key **[OK]** is pressed.
361000.1	**Input key [ESC]** Bit 1 = 1: Key **[ESC]** is pressed.
361000.2	**Input key [DOWN]** Bit 2 = 1: Key **[DOWN]** is pressed.
361000.3	**Input key [UP]** Bit 3 = 1: Key **[UP]** is pressed.

Software versions

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

- JetSym version 5.2
- HMI JVM-104, OS version 4.01

For more information on programming by STX, please turn to the online help in JetSym.

JetSym STX program

Description:

In the following sample program, the input keys are continuously retrieved in one task. Pressing one or more keys changes the background lighting of the display or the night lighting of the keys.

```
Var
    btnKey_Ok:    Bit At %XL 361000.0;
    btnKey_Esc:   Bit At %XL 361000.1;
    btnKey_Dwn:   Bit At %XL 361000.2;
    btnKey_Up:    Bit At %XL 361000.3;

    dispBackLed :        Int At %VL 364000;
    dispButtonBackLed :  Int At %VL 364001;
End_Var;
```

```
Task Main Autorun
    Loop
        If btnKey_Up Then
            // Half brightness of background lighting
            dispBackLed := 127;
        End_If;
        If btnKey_Dwn Then
            // Full brightness of background lighting
            dispBackLed := 255;
        End_If;
        If btnKey_Esc Then
            // Full brightness of night key lighting
            dispButtonBackLed := 255;
        End_If;
        If btnKey_Ok Then
            // Turn off night lighting of the keys
            dispButtonBackLed := 0;
        End_If;
        Delay(T#100ms);
    End_Loop;
End_Task;
```

Digipot

Introduction

The JVM-104 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:

Register	Description
363000	This register holds the current count value. If you turn the digipot, the count value increments or decrements. The following rule applies: ■ **Turning the digipot clockwise** = incrementing the register ■ **Turning the digipot counter-clockwise** = decrementing the register
363001	Bit 0: 0 = Pusbutton not pressed Bit 0: 1 = Pusbutton pressed
363002	This register lets you specify the lower limit for the digipot reading. If you continue turning the digipot counter-clockwise, register 363000 remains at this minimum value.
363003	This register lets you specify the upper limit for the digipot reading. If you continue turning the digipot clockwise, register 363000 remains at this maximum value.

Software versions

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

- JetSym version 5.2
- HMI JVM-104, OS version 4.01

For more information on programming by STX, please turn to the online help in JetSym.

JetSym STX program

In the following sample program, the background lighting for the JVM-104 is dimmed using the digipot. An upper and lower limit for the digipot is specified for this purpose. If you press the pushbutton, full background lighting is activated.

```
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;
```

Ignition and shutdown delay

Introduction

This chapter covers the ignition and the function `Shutdown()`.

Special registers

The special register 361100 of the JVM-104 is responsible for prompting ignition. Here, the following applies:

If ...	... then ...
Bit 0 = 0:	... ignition is ON and voltage is applied to terminal 15 IGNITION (+).
Bit 0 = 1:	... ignition is OFF. Ignition is switched off and no voltage is applied to terminal 15 IGNITION (+).

Default ignition function

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

If ...	... and ...	... then ...
... voltage is applied to the HMI, ...	... the ignition is off, ...	... the HMI does not boot up.
... voltage is applied to the HMI, ...	... the ignition is on, ...	... the HMI boots up.
... the HMI is powered on, ...	... the ignition is switched off (not the power supply), ...	... the HMI remains switched on.

Shutdown() function - Options

Notwithstanding the default ignition function, the `Shutdown()` function provides the following options:

- The HMI can be explicitly 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: System shutdown:	True False

Return value

This function transmits 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, the HMI can be restarted. Such a restart is carried out irrespective of the fact that the ignition is on.

Software versions

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

- JetSym version 5.2
- HMI JVM-104, OS version 4.01

For more information on programming by STX, please turn to the online help in JetSym.

JetSym STX program

If you switch off the vehicle's ignition, in the sample program the function `Shutdown()` is carried out after a delay of 3 seconds. The **Reboot** parameter for the `Shutdown()` function has the value **false**. This means that the device will switch off.

```
Var
    Ignition: Int At %VL 361100;
End_Var;

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

9.3 Runtime registers

Introduction

The JVM-104 provides several registers which are incremented by the operating system at regular intervals.

Application

These registers can be used to easily carry out time measurements in the application program.

Contents

Description of the runtime registers

Register overview

The device is equipped with the following runtime registers:

Register	Description
R 201000	Application time base in milliseconds
R 201001	Application time base in seconds
R 201002	Application time base in R 201003 * 10 ms
R 201003	Application time base units for R 201002
R 201004	System time base in milliseconds
R 201005	System time base in microseconds

R 201000

Application time base in milliseconds

Every millisecond this register is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)

R 201001

Application time base in seconds

Every second this register is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)

R 201002

Application time base in application time base units

Every [R 201003] * 10 ms this register value is incremented by one. Using the reset value 10 in register 201003, this register is incremented every 100 ms.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)

R 201003 **Application time base units for R 201002**

This register contains the multiplier for runtime register R 201002.

Register properties	
Values	1 ... 2,147,483,647 (* 10 ms)
Value after reset	10 (--> 100 ms)
Enabling conditions	After at least 10 ms

R 201004 **System time base in milliseconds**

Every millisecond this register value is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)
Type of access	Read

R 201005 **System time base in microseconds**

Every microsecond this register value is incremented by one.

Register properties	
Values	-2,147,483,648 ... 2,147,483,647 (overflowing)
Type of access	Read

Sample program - Runtime registers

Task

Measure how much time it takes to store variable values to a file.

Solution

Before storing the values, set register 201000 to 0.

Once the values have been stored, you can see from this register how much time it took to store the values [in milliseconds].

Software versions

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

- JetSym version 5.2
- HMI JVM-104, OS version 4.01

For more information on programming by STX, please turn to the online help in JetSym.

JetSym STX program

```
Var
    DataArray:    Array[2000] Of Int;
    File1:        File;
    WriteTime:    Int;
    WriteIt:      Bool;

    MilliSec:     Int At %VL 201000;
End_Var;

Task WriteToFile Autorun
    Loop
        // Resetting the start flag
        WriteIt := False;
        // Waiting for user to set start flag
        When WriteIt Continue;

        // Opening the file in write mode
        // If there is no file available, a new file
        // is created
        If FileOpen(File1, 'Test.dat', fWrite) Then

            // Setting the application time base register to zero
            MilliSec := 0;

            // Writing the data array into the file
            FileWrite(File1, DataArray, SizeOf(DataArray));

            // Registering the run time
            WriteTime := MilliSec;
            FileClose(File1);

            // Displaying the run time
            Trace(StrFormat('Time : %d [ms]$n', WriteTime));
```

```
            Else
                // Displaying the error message
                Trace('Unable to open file!$n');
            End_If;
        End_Loop;
    End_Task;
```

10 Operating system update

Introduction

Jetter AG are continuously striving to enhance the operating systems for their HMIs. Enhancing means adding new features, upgrading existing functions and fixing bugs.

This chapter describes how to carry out operating system updates.

Downloading an operating system

You can download operating systems from the Jetter AG **homepage http://www.jetter.de**. You get to the OS files for download at *Mobile Automation - Support - Downloads* or by clicking on the quick link *Operating System Download* on the website of the corresponding HMI.

Contents

10.1 Updating the operating system of an HMI

Introduction

This chapter describes how to update the OS of the JVM-104. There are several options to transfer the OS file to the device:

- From within the programming tool JetSym
- From the directory \App

Contents

OS update by means of JetSym

Introduction

The programming tool JetSym offers an easy way to transfer an OS file to the JVM-104.

Prerequisites

- An OS file for the JVM-104 is available.
- The device is connected to the PC via CAN.
- The following parameters have been set in JetSym:
  - Type of device
  - Type of interface
  - Node ID
  - CAN baud rate
- The controller must not be de-energized during the OS update process.

Updating the OS

To update the OS, proceed as follows:

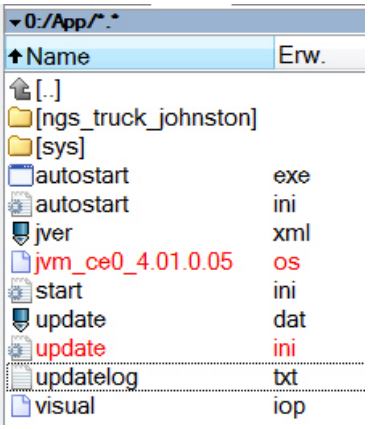
Step	Action
1	Select in the JetSym menu **Build** the menu item **Update OS.** **Result:** The file selection dialog opens.
2	Select the new OS file here. **Result:** In JetSym, a confirmation dialog opens.
3	Launch the OS upload by clicking the button **Yes**.
4	Wait until the update process is completed.
5	To activate the newly installed OS, re-boot the device.

Operating system update via \App

Procedure

Copying update files into the directory *\App* lets you easily update the operating system. To update the OS, proceed as follows:

Step	Action
1	Enter the name of the file collection into the file **update.ini**. Otherwise the update will not work.
2	Copy the file collection or OS and the file **update.ini** into the directory *\App*.
3	Restart the device.
⇨	**Autostart.exe** detects the update during the boot process, installs the files and restarts the device. **Important!** Do not interrupt this process.
4	Open the file **updatelog.txt** to make sure that the update has completed without errors.



11 Application program

Introduction

This chapter describes how to store the application program in JVM-104. The user determines the program that is to be executed.

Required programmer's skills

This chapter requires knowledge on how to create application programs in JetSym and how to transmit them via the file system of the JVM-104.

Contents

Application program - Default path

Introduction

When uploading the application program from JetSym to the controller, this program is stored as a file to the internal flash disk. The device enters the path and file name into the file **\App\start.ini**.

Path and file name

In the directory *\App*, JetSym, by default, creates a subdirectory and assigns the project name to it. Then, JetSym stores the application program to this subdirectory assigning the extension **.es3** to it. Path and file names are always converted into lower case letters.

\App\start.ini - Structure

This file is a text file with one section holding two entries:

Element	Description
[Startup]	Section name
Project	Path to the application program file, relative to *\App*
Program	Name of the application program file

Example:

```
[Startup]
Project = test_program
Program = test_program.es3
```

Result: The application program is loaded from the file **\App\test_program\test_program.es3**.

Loading an application program

Introduction

At reboot of the application program via JetSym or booting the JVM-104, the application program is loaded and executed via the file system.

Loading process

The application program is loaded by the JVM-104's OS as follows:

Step	Description
1	The OS reads the file **/App/start.ini** from the internal flash disk.
2	The OS evaluates the **Project** entry. It contains the path leading to the application program file.
3	The OS evaluates the **Program** entry. It contains the program name.
4	The OS loads the application program from the file **<Project>\<Program>**.

12 Quick reference JVM-104

Corresponding OS version

This quick reference summarizes in brief the registers and flags of the HMI JVM-104 running OS version 4.01.

Default address on the CANopen® bus

Default address of the JVM-104:
Node ID: 127 (0x7F)

Maximum number of CANopen® interfaces

Maximum number of CAN interfaces: 1
CANMAX: 0

Maximum number of SAE J1939 interfaces

Maximum number of CAN interfaces: 0
CANMAX: 0

Registers - General overview

100000 ... 100999	Electronic Data Sheet (EDS)
101000 ... 101999	Configuration
104000 ... 104999	Ethernet
106000 ... 106999	CAN
108000 ... 108999	CPU/backplane
200000 ... 209999	General system registers
210000 ... 219999	Application program
230000 ... 239999	Networking via JetIP
260000 ... 269999	RemoteScan
270000 ... 279999	Modbus/TCP
290000 ... 299999	E-mail
310000 ... 319999	File system/data files
350000 ... 359999	User-programmable IP interface
360000 ... 369999	Display
1000000 ... 1029999	Application registers (remanent)

I/Os - General overview

Input keys

361000 ... 361007	Bit-coded map of input keys

Flags - General overview

0 ... 255	Application flags (remanent)
256 ... 2047	Overlaid by registers R 1000000 through 1000055
2048 ... 2303	Special flags

Electronic Data Sheet (EDS)

100500	Interface (0 = CPU, 4 = Base board)
100600	Internal version number
100601	Module ID
100602 ...	Module name (register string)
100612	
100613	PCB revision
100614	PCB options

Production

100700	Internal version number
100701 ...	Serial number (register string)
100707	
100708	Day
100709	Month
100710	Year
100711	TestNum.
100712	TestRev.

Features

100800	Internal version number
100801	MAC Address (Jetter)
100802	MAC address (device)

Electronic name plate (device as a whole)

Production

100900	Internal version number
100901 ...	Serial number (register string)
100907	
100708	Day
100709	Month
100710	Year
100950	Internal version number
100951	Module ID
100952 ...	Module name (register string)
100962	
100965	Config ID
100966	Vendor ID
100967	Variant ID
100968	Type ID
100992	Navision ID
100993	FBG version

Configuration

From system configuration

101100	IP address (rw - remanent)
101101	Subnet mask (rw - remanent)
101102	Default gateway (rw - remanent)

Used by the system

101200	IP address
101201	Subnet mask
101202	Default gateway

Ethernet

IP

104531	Current IP address (rw - temporary)
104532	Current subnet mask (rw - temporary)
104533	Current default gateway (rw - temporary)

CAN

106000	Baud rate CAN 1
106001	Node ID CAN 1

Flash memory

107501	30: Read present flash disk statistics 99: Clear flash disk statistics
107510	Available sectors
107511	Used sectors
107512	Blocked sectors
107513	Free sectors
107520	Size of the flash disk in bytes

107521	Used memory in bytes
107522	Blocked memory in bytes
107523	Free memory in bytes

CPU hardware

108015	Voltage of backup battery (e.g. for clock) 0 = Data not valid 1 = Supply voltage is OK Once the power supply has been restored, enter 1 into this register.

System information

108500 ... 108509	JetVM-DII version string
108510 ... 108519	Version string of the host application
108520 ... 108529	File name of the host application
108530 ... 108539	OS version (string)
108570	CPU type
108571	Number of CPUs
108573	Physical RAM
108574	Free physical RAM
108575	Memory utilization (in %)
108581	Screen width (in pixels)
108582	Screen height (in pixels)
108590	HID version

USB flash drive

109000	Bit 0 = 1: Data medium is available Bit 1 = 1: Data medium is ready
109001	1 = Data medium is write-protected (only valid if R 109000 = 3)
109002	Size in MBytes

General system registers

200000	OS version (major * 100 + minor)
200001	Application program is running (bit 0 = 1)
200008	Error register (identical with 210004) Bit 8: Illegal jump Bit 9: Illegal call Bit 10: Illegal index Bit 11: Illegal opcode Bit 12: Division by 0 Bit 13: Stack overflow Bit 14: Stack underflow Bit 15: Illegal stack Bit 16: Error when loading the application program Bit 24: Timeout - Cycle time Bit 25: Timeout - Task lock Bit 31: Unknown error
200168	Bootloader version (IP format)
200169	OS version (IP format)
201000	Runtime register in milliseconds (rw)
201001	Runtime register in seconds (rw)
201002	Runtime register in R 201003 Units (rw)
201003	* 10 ms units for R 201002 (rw)
201004	Runtime register in milliseconds (ro)
201005	Runtime registers in microseconds (ro)
202930	Web status (bit-coded) Bit 0 = 1: FTP server available Bit 1 = 1: HTTP server available Bit 2 = 1: E-mail available Bit 3 = 1: Data file function available Bit 4 = 1: Modbus/TCP has been licensed Bit 5 = 1: Modbus/TCP available Bit 6 = 1: Ethernet/IP available
202960	Password for system command register (0x424f6f74)
202961	System command register
202980	Error history: Number of entries
202981	Error history: Index
202982	Error history: Entry
203100 ... 203107	32-bit overlaying - Flag 0 ... 255
203108 ... 203123	16-bit overlaying - Flag 0 ... 255
203124 ... 203131	32-bit overlaying - Flag 2048 ... 2303
203132 ... 203147	16-bit overlaying - Flag 2048 ... 2303
209700	System logger: Global enable
209701 ... 209739	Enabling system components

Application program

210000	Application program is running (bit 0 = 1)
210001	JetVM version
210004	Error register (bit-coded) Bit 8: Illegal jump Bit 9: Illegal call Bit 10: Illegal index Bit 11: Illegal opcode Bit 12: Division by 0 Bit 13: Stack overflow Bit 14: Stack underflow Bit 15: Illegal stack Bit 16: Error when loading the application program Bit 24: Timeout - cycle time Bit 25: Timeout - task lock Bit 31: Unknown error
210006	Highest task number
210007	Minimum program cycle time
210008	Maximum program cycle time
210009	Current program cycle time
210011	Current task number
210050	Current program position within an execution unit
210051	ID of the execution unit being processed
210056	Desired total cycle time in µs
210057	Calculated total cycle time in µs
210058	Maximum time slice per task in µs
210060	Task ID (for R210061)
210061	Priority for task [R210060]
210063	Length of scheduler table
210064	Index in scheduler table
210065	Task ID in scheduler table
210070	Task ID (for R210071)

210071	Timer number (0 ... 31)
210072	Manual triggering of a timer event (bit-coded)
210073	End of cyclic task (task ID)
210074	Command for cyclic tasks
210075	Number of timers
210076	Timer number (for R210077)
210077	Timer value in milliseconds
210100 ... 210199	Task state
210400 ... 210499	Task - Program address
210600	Task ID of a cyclical task (for R210601)
210601	Processing time of a cyclical task in per mil figure
210609	Task lock timeout in ms -1: Monitoring disabled
210610	Timeout (bit-coded) Bit 0 -> Timer 0, etc.

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	Amount of network errors
232711	Error code of last access 0 = No error 1 = Timeout 3 = Error message of the remote station 5 = Invalid network address 6 = Invalid amount of registers 7 = Invalid interface number
232717	Max. number of retries
232718	Number of retries

Modbus/TCP

272702	Register offset
272704	Input offset
272705	Output offset
278000 ... 278999	16-bit I/O registers overlaid by virtual I/Os 20001 ... 36000

E-mail

292932	IP address of the SMTP server
292933	IP address of the POP3 server
292934	Port number of the SMTP server
292935	Port number of POP3 server
292937	Status of e-mail processing
292938	Task ID - E-mail

File system/data file function

312977	Status of file operation
312978	Task ID

User-programmable IP interface

Reading out the connection list

350000	Last result (-1 = no connection selected)
350001	1 = Client; 2 = Server
350002	1 = UDP; 2 = TCP
350003	IP address
350004	Port number
350005	Connection state
350006	Number of sent bytes
350007	Number of received bytes

Application registers

1000000 ... 1005999	32-bit integer (remanent)

CAN-PRIM registers

200010500	CAN-PRIM status
200010501	CAN-PRIM command register
200010502	Message box number
200010503	FIFO buffer occupancy
200010504	FIFO data
200010506	Global receiving mask
200010507	Global receive ID
200010510	Box status register
200010511	Box configuration register
200010512	CAN ID
200010513	Number of data bytes
200010514	Data bytes
...	
200010521	

Display

Input keys

361000 ... 361007	Bit-coded map of input keys e.g. bit 0: 1 = Key 1 is pressed
361000.0	KEY_OK
361000.1	KEY_ESC
361000.2	KEY_DOWN
361000.3	KEY_UP

Ignition (IGN)

361100	Bit 0: 0 = Ignition ON 1 = Ignition OFF

Digipot

363000	Present count value
363001	Digipot key
363002	Minimum count value
363003	Maximum count value

Display

364000	Background lighting
364001	Night-lighting of keys
364003	Brightness sensor

Visualization

365100	Language selection according to ID

Special flags for networks

2075	Error in networking via JetIP

Special flags - Interface monitoring

2088	OS flag - JetIP
2089	User flag - JetIP
2098	OS flag - Debug server
2099	User flag - Debug server

32 combined flags

203100	0 ... 31
203101	32 ... 63
203102	64 ... 95
203103	96 ... 127
203104	128 ... 159
203105	160 ... 191
203106	192 ... 223
203107	224 ... 255

16 combined flags

203108	0 ... 15
203109	16 ... 31
203110	32 ... 47
203111	48 ... 63
203112	64 ... 79
203113	80 ... 95
203114	96 ... 111
203115	112 ... 127
203116	128 ... 143
203117	144 ... 159
203118	160 ... 175
203119	176 ... 191
203120	192 ... 207
203121	208 ... 223
203122	224 ... 239
203123	240 ... 255

32 combined special flags

203124	2048 ... 2079
203125	2080 ... 2111
203126	2112 ... 2143
203127	2144 ... 2175
203128	2176 ... 2207
203129	2208 ... 2239
203130	2240 ... 2271
203131	2272 ... 2303

16 combined special flags

203132	2048 ... 2063
203133	2064 ... 2079
203134	2080 ... 2095
203135	2096 ... 2111
203136	2112 ... 2127
203137	2128 ... 2143
203138	2144 ... 2159
203139	2160 ... 2175
203140	2176 ... 2191
203141	2192 ... 2207
203142	2208 ... 2223
203143	2224 ... 2239
203144	2240 ... 2255
203145	2256 ... 2271
203146	2272 ... 2287
203147	2288 ... 2303

Overlaid application registers/flags

1000000	256 ... 287
1000001	288 ... 319
1000002	320 ... 351
1000003	352 ... 383
1000004	384 ... 415
1000005	416 ... 447
1000006	448 ... 479
1000007	480 ... 511
1000008	512 ... 543
1000009	544 ... 575
1000010	576 ... 607
1000011	608 ... 639
1000012	640 ... 671
1000013	672 ... 703
1000014	704 ... 735
1000015	736 ... 767
1000016	768 ... 799
1000017	800 ... 831
1000018	832 ... 863
1000019	864 ... 895
1000020	896 ... 927
1000021	928 ... 959
1000022	960 ... 991
1000023	992 ... 1023
1000024	1024 ... 1055
1000025	1056 ... 1087
1000026	1088 ... 1119
1000027	1120 ... 1151
1000028	1152 ... 1183
1000029	1184 ... 1215
1000030	1216 ... 1247
1000031	1248 ... 1279
1000032	1280 ... 1311
1000033	1312 ... 1343
1000034	1344 ... 1375
1000035	1376 ... 1407
1000036	1408 ... 1439
1000037	1440 ... 1471
1000038	1472 ... 1503
1000039	1504 ... 1535
1000040	1536 ... 1567
1000041	1568 ... 1599
1000042	1600 ... 1631
1000043	1632 ... 1663
1000044	1664 ... 1695
1000045	1696 ... 1727
1000046	1728 ... 1759
1000047	1760 ... 1791
1000048	1792 ... 1823
1000049	1824 ... 1855
1000050	1856 ... 1887
1000051	1888 ... 1919
1000052	1920 ... 1951
1000053	1952 ... 1983
1000054	1984 ... 2015
1000055	2016 ... 2047

System function

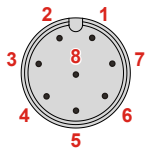
For reasons of compatibility, the system functions are listed below. In JetSym STX, use the corresponding JetSym STX functions instead of system functions.

4	Conversion from BCD to HEX
5	Conversion from HEX to BCD
20	Square root
21	Sine
22	Cosine
23	Tangent
24	Arc sine
25	Arc cosine
26	Arc tangent
27	Exponential function
28	Natural logarithm
29	Absolute value
30	Separation of digits before and after the decimal point
50	Sorting register values
60	CRC generation for Modbus RTU
61	CRC check for Modbus RTU
65/67	Reading register block via Modbus/TCP
66/68	Writing register block via Modbus/TCP
80/85	Initializing RemoteScan
81	Starting RemoteScan
82	Stopping RemoteScan
90	Writing a data file
91	Appending a data file
92	Reading a data file
96	Deleting a data file
110	Sending an e-mail
150	Configuring NetCopyList
151	Deleting NetCopyList
152	Sending NetCopyList

JetSym STX functions

System function	Corresponding JetSym STX function
4	Function Bcd2Hex(Bcd: Int): Int;
5	Function Hex2Bcd(Hex: Int): Int;
50	Function QSort(DataPtr: Int, ElementCnt: Int, ElementSize: Int, SortOffset: Int, SortType: STXBASETYPE, SortMode: QSORTMODE): Int;
60	Function ModbusCRCgen(FramePtr: Int, Length: Int): Int;
61	Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int;
65/67	Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int;
66/68	Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int;
80/85	Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int;
81	Function RemoteScanStart(Protocol: Int): Int;
82	Function RemoteScanStop(Protocol: Int): Int;
90/91	Function FileDAWrite(Const Ref FileName: String, Const Ref Mode: String, VarType: DAWRITE_TYPE, First: Int, Last: Int): Int;
92	Function FileDARead(Const Ref FileName: String): Int;
110	Function EmailSend(Const Ref FileName: String): Int;
150	Function NetCopyListConfig(IPAddr: Int, IPPort: Int, Const Ref List: TNetCopyLinstL): Int;
151	Function NetCopyListSend(Handle: Int): Int;
152	Function NetCopyListDelete(Handle: Int): Int;

Assignment: 8-pin M12 connector



Pin	Description
1	Power supply UB for logic circuits Voltage: DC 12 V or DC 24 V Maximum current: 2 A
2	Unassigned
3	Ignition (+)
4	Unassigned
5	CAN_L
6	Reference potential GND
7	CAN_H
8	Shield

Appendix

Introduction

This appendix contains electrical and mechanical data, as well as operating data.

Contents

A: Interfaces

Introduction

The HMI JVM-104 is equipped with the following interface:

- M12 male connector

M12 male connector

The M12 connector has the following function:

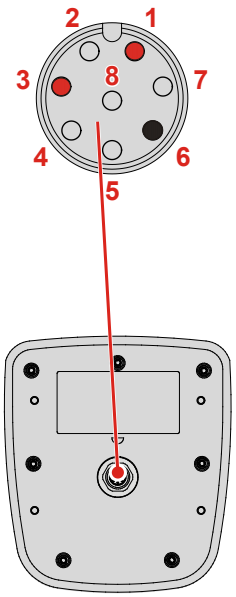
- Power supply of the JVM-104
- CANopen® bus interface: CAN 1
- Recognition of the ignition

Contents

Pinout - Overview

Power supply

This chapter describes the pinout of the connector for the power supply.
The diagram shows the pinout of the power supply and ignition connector (viewing the cable side):



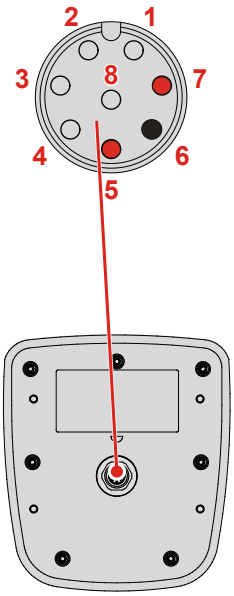
The pinout is as follows:

Pin	Description	Terminal number in vehicles
1	Power supply UB for logic circuits Voltage: DC 12 V or DC 24 V Current consumption: 2 A max.	Terminal # 30
3	Ignition (+)	Terminal # 15
6	Reference potential (GND)	Terminal # 31

CAN interface

This chapter describes the pinout of the connector for the CAN interface.

The diagram shows the pinout of the connector for the CANopen® bus (viewing the cable side). Pin 6 for the reference potential is also color-coded.



The pinout is as follows:

Pin	Description
5	CAN_L
6	Reference potential (GND)
7	CAN_H

B: Technical data

Introduction

This chapter contains information on electrical and mechanical data, as well as on operating data of the JVM-104.

Contents

Technical specifications

Technical specifications - Power supply UB

Parameter	Description
Rated voltage	DC 12 V or DC 24 V
Permissible voltage range UB	DC 8 V ... DC 32 V, to ISO 7637
Permissible voltage range - Ignition	DC 5 V ... DC 32 V
Maximum current	2 A
Load dump protection	DC 70 V max.
Typical current consumption logic circuit (UB)	170 mA at DC 12 V 90 mA at DC 24 V
Power consumption	Approx. 2 W
Integrated protective functions	Protection against polarity reversal, overloading, voltage surges

Technical specifications - Display

Parameter	Description
Display	3.5" TFT LCD flat screen monitor
Brightness	LED backlight (white), typ. 350 cd/m^2
Display resolution	320 x 240 pixels

Technical specifications - CAN interface

Parameter	Description
Baud rate	250 kBaud ... 1 MBaud
Bus terminating resistor	None
External bus termination	120 Ω
Connector specifications	Twisted pair conductors, unshielded

Max. number of CANopen® ports

Parameter	Description
Max. number of CAN ports	1
CANMAX	0

Max. number of SAEJ1939 ports

Parameter	Description
Max. number of CAN ports	0
CANMAX	0

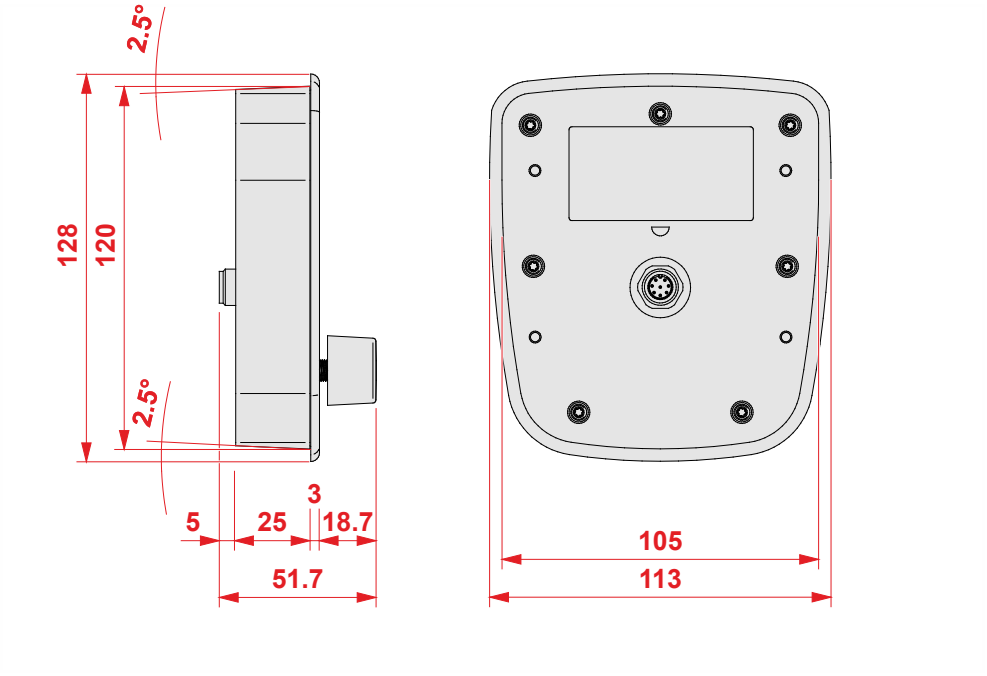
Memory configurations

Parameter	Description
Number of remanent registers	30,000
Remanent memory for variables	120,000 bytes
Flash disk: Total memory Folder **App** Folder **Data**	 512 MBytes 64 MBytes 368 MBytes

Physical dimensions

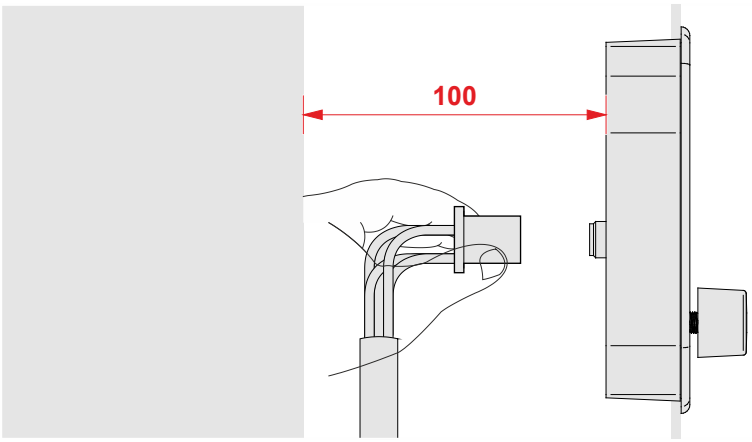
Physical dimensions

The illustration below shows the physical dimensions of the JVM-104 in millimeters.



Space required for installation and service

The illustration shows the space required for the HMI JVM-104. It is stated in millimeters.



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

- It should be possible to disconnect the connector at any time.

Operating parameters - Environment and mechanics

Environment

Parameter	Value	Standard
Operating temperature range	-20 ... +60 °C	
Storage temperature range	-30 ... +70 °C	ISO 16750-4 DIN EN 60068-2-1 DIN EN 60068-2-2
Air humidity	10 ... 95 % Do not use a steam jet or other such devices to clean the JVM-104.	DIN EN 61131-2
Climate test	Humid heat	ISO 16750-4
Pollution degree	2	DIN EN 61131-2
Installation location	The JVM-104 must be installed in the driver's cab.	

Mechanical parameters

Parameter	Value	Standard
Vibration	Broadband noise, 10 Hz/0.005 $(m/s^2)^2$/Hz 200 Hz/0.02 $(m/s^2)^2$/Hz 300 Hz/0.01 $(m/s^2)^2$/Hz 350 Hz/0.002 $(m/s^2)^2$/Hz Duration: 3x24 h	To DIN EN 60068-2-64
Shock resistance	Sinusoidal half wave, 30 g (300 m/s^2), 18 ms, 3 shocks in all 6 orientations	To DIN EN 60068-2-27
Degree of protection	On the front: IP65 Rear: IP65	DIN EN 60529

Operating parameters - EMC

Voltage testing at UB and UB_PA

The voltage testing results comply with DIN EN 16750-2.

EMC - Emitted interference

Parameter	Value	Standard
Emitted interference to e1	Frequency band 400 ... 1,000 MHz, limit 63 dB (µV/m), constant	DIN EN 55025
Emitted interference to CE	0.15 ... 0.5 MHz, 66 ... 56 dB (µV) QP DC supply 0.5 ... 5 MHz, 56 dB (µV) QP 5 ... 30 MHz, 60 dB (µV) QP	DIN EN 55011-DC
	30 ... 230 MHz, 30/40 dB (µV/m) enclosure 230 ... 1,000 MHz, 37/47 dB (µV/m)	DIN EN 55011-HF

EMC - Immunity to interference

Parameter	Value	Standard
Immunity to interference to CE	10 V/m over 80 % of the frequency band	DIN EN 61000-4-3
	2/1 kV data 4/2 kV power	DIN EN 61000-4-4
	± 1 kV line/ground ± 0.5 kV line/line	DIN EN 61000-4-5
	10 V, 0.15 ... 80 MHz, 80 % AM sine 1 kHz	DIN EN 61000-4-6
ESD	Discharge through air: Test peak voltage 8 kV Contact discharge: Test peak voltage 4 kV	DIN EN 61000-4-2

C: Index

Jetter automation

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg | Germany

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

We automate your success.