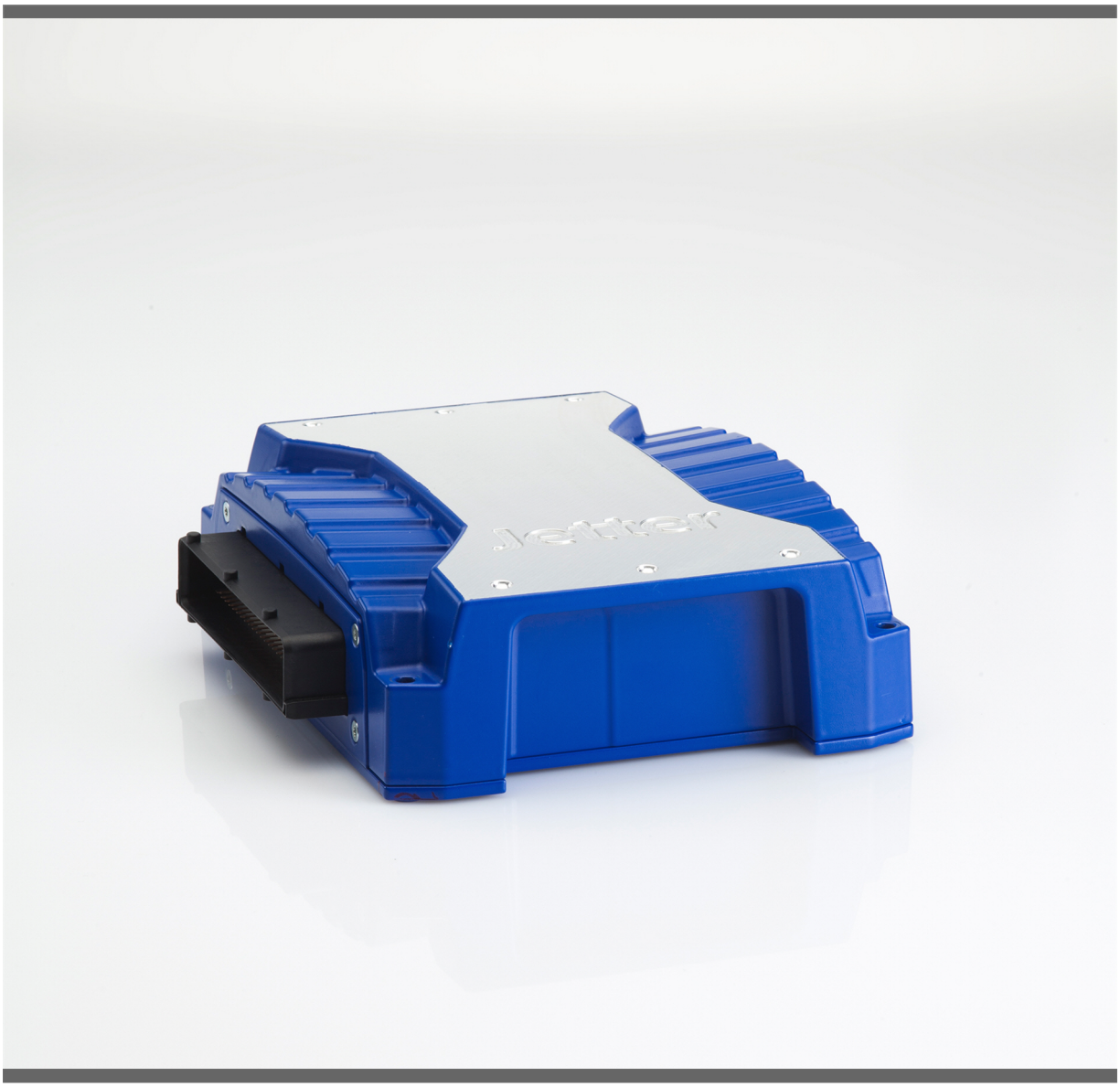
JCM-350-E03

Controller on the CAN Bus



User Manual

Jetter

Variant: Jetter
Item # 60877279
Revision 1.09.2
November 2011 / Printed in Germany

Address

How To Contact us:

Jetter AG
Graeterstrasse 2
D-71642 Ludwigsburg
Germany

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

Assignment to Product

This user manual is an integral part of JCM-350-E03:

Type: _____

Serial #: _____

Year of construction: _____

Order #: _____

CE

To be entered by the customer:

Inventory #: _____

Place of operation: _____

Significance

Significance of this user manual

The user manual is an integral part of JCM-350-E03:

- It must be kept in a way that it is always at hand, until the JCM-350-E03 will be disposed of.
- If the JCM-350-E03 is sold or loaned/leased out, the user manual has to be passed on.

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

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

This user manual contains important information on how to transport, erect, install, operate, maintain and repair the JCM-350-E03.

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

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

Hazard Levels

Introduction

This topic describes the safety labels and hazard levels used in this manual.

Safety Labels

Signs using this symbol are to warn you of inuries or even death. It is imperative to follow the instructions to prevent hazards.

Hazard Levels

Safety information is classified into the following hazard levels:

Hazard Level	Consequences	Probability
DANGER	Death/severe injury (irreversible)	The hazard is imminent
WARNING	Death/severe injury (irreversible)	Potential occurrence
CAUTION	Slight injury (reversible)	Potential occurrence
CAUTION	Material damage	Potential occurrence

Table of Contents

1 Safety Instructions

Introduction

This chapter contains the general safety instructions and warns of possible residual dangers.

Contents

This chapter contains the following topics:

General Safety Instructions

Introduction

This device complies with the valid safety regulations and standards. Special emphasis was given to the safety of the users.

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

- relevant accident prevention regulations;
- accepted safety rules;
- EC guidelines and other country-specific regulations

Intended Conditions of Use

Usage according to the intended conditions of use implies operation in accordance with this user manual.

The controller JCM-350-E03 has been developed and designed to control certain applications for commercial vehicles and mobile machines, such as sweepers, fire-fighting vehicles, harvesting and construction machinery.

The controller JCM-350-E03 meets the requirements of the European Automotive EMC Directive for electric/electronic subassemblies. The controller JCM-350-E03 is intended for installation in a mobile machine.

The controller JCM-350-E03 must be operated within the limits and conditions established in the technical specifications. The operating voltage of the controller JCM-350-E03 is classified as SELV (Safety Extra Low Voltage). Therefore, the JCM-350-E03 controller is not subject to the EU Low Voltage Directive.

Usage Other Than Intended

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

The JCM-350-E03 is no safety-related part as per Machinery Directive 2006/42/EC. This device is not qualified for safety-relevant applications and must, therefore, NOT be used to protect persons.

If the device is to be run under ambient conditions which differ from the allowed operating conditions, Jetter AG is to be contacted beforehand.

Personnel Qualification

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

Product Life Cycle	Minimum Qualification
Transport / Storage:	Trained and instructed personnel with knowledge in handling electrostatic sensitive components.
Mounting / Installation:	Specialized personnel with training in electrical/automotive engineering, such as automotive mechatronics fitters.
Commissioning / Programming:	Trained and instructed experts with profound knowledge of, and experience with, automotive / automation technology, such as automotive engineers for mobile machinery.
Operation:	Trained, instructed and assigned personnel with knowledge in operating electronic devices for mobile machinery.

Product Life Cycle	Minimum Qualification
Decommissioning:	Specialized personnel with training in electrical/automotive engineering, such as automotive mechatronics fitters.

Modifications and Alterations to the Device

For safety reasons, no modifications and changes to the device and its functions are permitted.

Any modifications to the device not expressly authorized by Jetter AG will result in a loss of any liability claims to Jetter AG.

The original parts are specifically designed for the device. Parts and equipment from other manufacturers are not tested on our part, and are, therefore, not released by Jetter AG.

The installation of such parts may impair the safety and the proper functioning of the device.

Any liability on the part of Jetter AG for any damages resulting from the use of non-original parts and equipment is excluded.

Transport

The JCM-350-E03 contains electrostatic sensitive components which can be damaged if not handled properly.

To exclude damages to the JCM-350-E03 during transport it should only be shipped in its original packaging or in packaging protecting against electrostatic discharge. This is particularly true for transport via mail.

- Use an appropriate outer packaging to protect the JCM-350-E03 against impact or shock.
- In case of damaged packaging inspect the device for any visible damage. Inform your freight forwarder and the manufacturer, if applicable.

Storing

When storing the JCM-350-E03 observe the environmental conditions given in the technical specification.

Repair and Maintenance

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

The device must be sent to Jetter AG for repair.

Disposal

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

Residual Dangers and Protective Measures

Residual Dangers

Consider the residual dangers mentioned in this chapter when assessing the risks associated with your machine.

DANGER

Hazard in explosive gas atmosphere!

This device can become a source of ignition in potentially explosive atmospheres.

➢ **Do not use this device in potentially explosive atmospheres.**

WARNING

Hot surface hazard!

The JCM-350-E03 can heat up during operation. During operation the surface temperature of this device will become hot enough (> 60 °C) to cause burns.

➢ **Take protective measures to prevent inadvertent contact with the device, e.g. install protective covers.**

➢ Allow the device to cool down for some time before you start working on it, e.g. to carry out maintenance jobs.

CAUTION

Possible occurrence of malfunctions!

CAN wires which have not been twisted may increase susceptibility to noise. This may disturb communications with the device which, in turn, may cause malfunctions.

➢ Make sure that twisted pair cables are used for connecting the CAN interfaces.

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 - JCM-350-E03

Controller JCM-350-E03

The controller JCM-350 has especially been designed for use in the harsh environment of commercial vehicles and mobile machines.

JCM-350-E03 - Configuration

The JCM-350-E03 consists of the controller JCM-350 and the I/O module JXM-IO-E02 which are internally connected via CAN bus. The CAN bus is brought out to allow communication with other CANopen® nodes. The default node ID of the JXM-IO-E02 is 16, the default node ID of the JCM-350 is 127. This way, both components within the JCM-350-E03 can be addresses separately.

Product Features

The features of this product are listed below:

- CANopen® node with 1 or 2 interfaces to CAN-2.0B
- 16 digital active-high inputs
- 10 digital active-high outputs supplying up to 2.5 A
- 6 digital active-high outputs supplying up to 5 A
- 5 digital inputs which can be configured as active-high or active-low inputs
- 1 analog output (resolution: 8 bits)
- 4 analog inputs (voltage, current, resolution: 10 bits)
- 2 frequency inputs (5 Hz ... 20 kHz, resolution: 10 Hz)
- 3 PWM outputs, 2.5 A max.
- 1 H-bridge, 2.5 A max.
- 2 tri-state inputs for setting the node ID
- Powerful programming language JetSym STX
- Non-volatile registers: 6.000
- RAM memory: 16 MBytes
- Flash memory: 16 MBytes
- Realtime clock (without buffer)

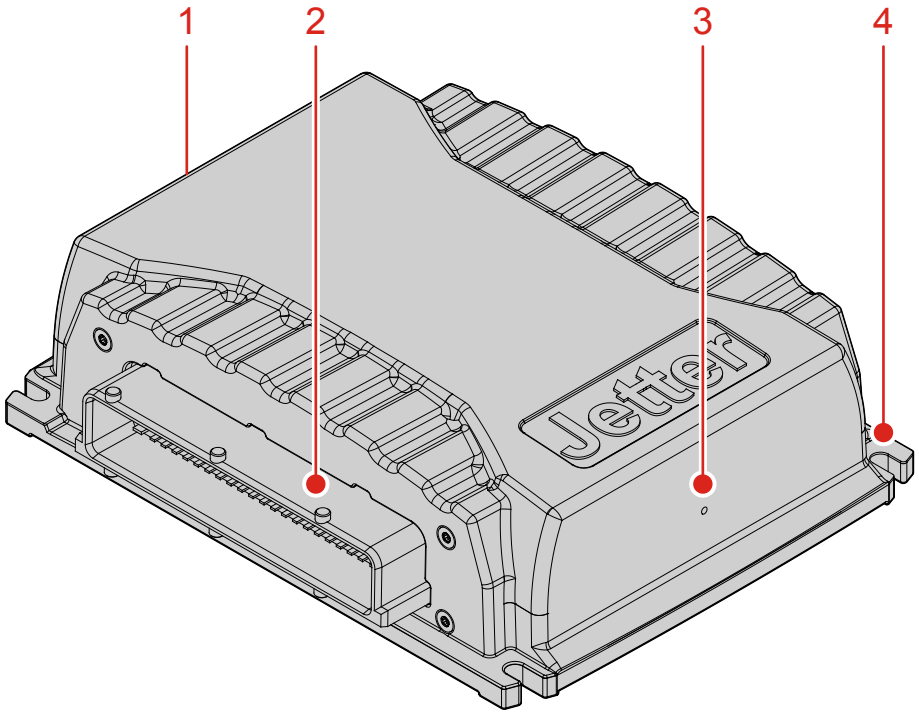
Parts and Interfaces

Introduction

This chapter describes the parts and interfaces of the JCM-350-E03.

Parts and Interfaces

The JCM-350-E03 features the following parts and interfaces:



Number	Content	Description
1	Nameplate	For identifying the JCM-350-E03
2	Connector	For connecting external components and the controller
3	Pressure compensation membrane	Compensation of inside and outside air pressure
4	Fastening lugs	For screwing down the JCM-350-E03

Order Reference / Options

Order Reference

The JCM-350-E03 is available in the following configurations. To order a specific module from Jetter AG please specify the corresponding part number.

Part Number	Order Reference	Name
10000753	JCM-350-E03-G06-K00	Controller

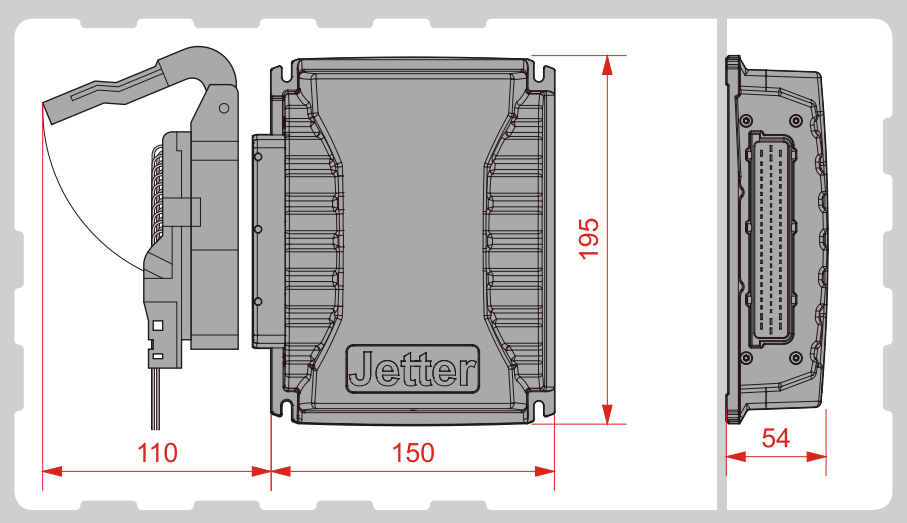
Physical Dimensions

Introduction

This chapter details the physical dimensions of the JCM-350-E03 and the conditions for installation.

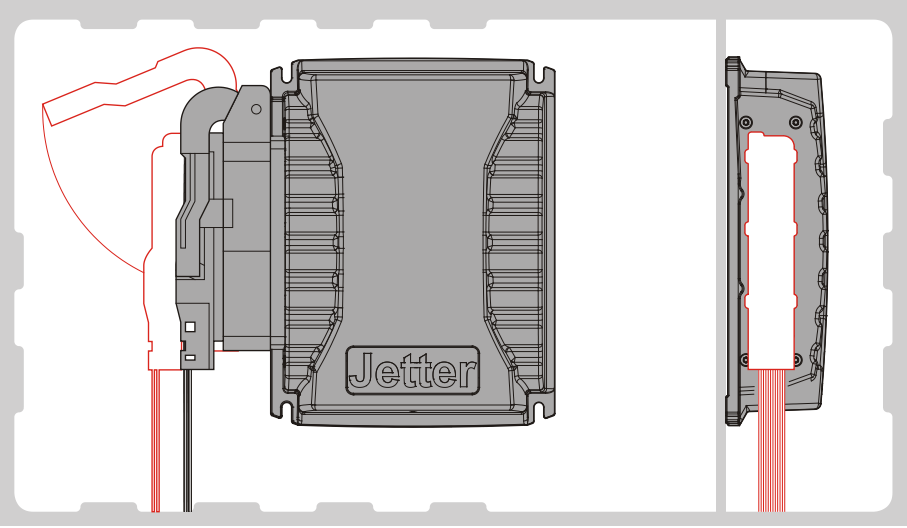
Physical Dimensions

The diagram shows the dimensions of the JCM-350-E03.



Space Required for Installation and Service

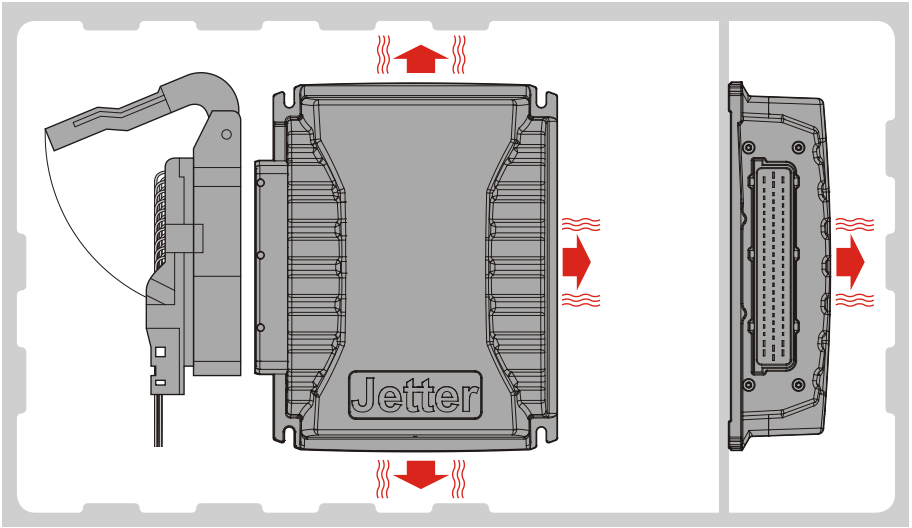
The diagram shows the space required for the JCM-350-E03.



Ensure there is enough space around the connector for servicing requirements. It should be possible to disconnect the connector at any time.

Space Required to Protect Against Overheating

The diagram indicates the safety distances to protect against overheating.



Please note:

- The JCM-350-E03 increases the temperature of the environment as a result of heat emission under load.
- The JCM-350-E03 operates without interruption at an ambient temperature of up to +85 °C.

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

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

3 Identifying the Controller

Purpose of this Chapter

This chapter is for supporting you in identifying the following information with regard to JCM-350-E03:

- Hardware revision.
- Electronic data sheet (EDS). Numerous production-relevant data are permanently stored in the EDS.
- Identifying the OS Release of the Controller and Software Components.

Prerequisites

To be able to identify the JCM-350-E03 controller the following prerequisites have to be fulfilled:

- The controller is connected to a PC.
- The programming tool JetSym 4.3 or higher is installed on the PC.

Information for Hotline Requests

If you have to contact the hotline of Jetter AG in case of a problem, please have the following information on the JCM-350-E03 controller ready:

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

Contents

3.1 Identification by Means of the Nameplate

Introduction

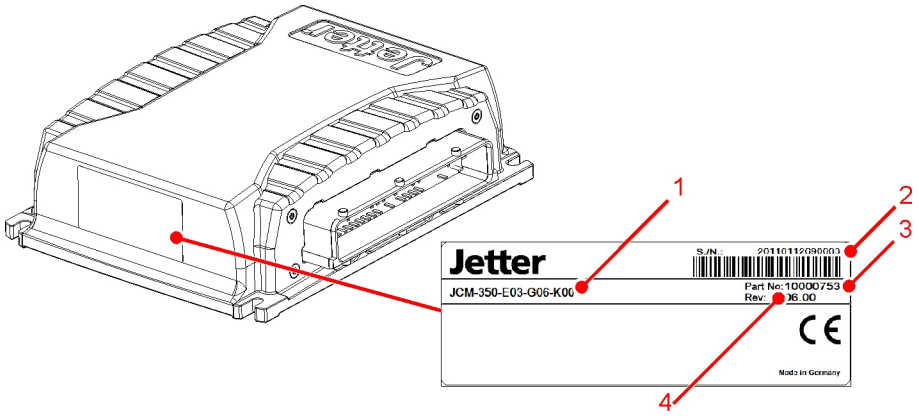
The nameplate is attached to the housing of the JCM-350-E03 and contains details, such as hardware revision number and serial number. You will need this information when contacting the Jetter AG hotline in case of a problem.

Contents

Nameplate

Nameplate

The nameplate of a JCM-350-E03 contains the following information:



Number	Description
1	Controller type
2	Serial number
3	Part number
4	Hardware revision

3.2 Electronic Data Sheet EDS

Introduction

The controller JCM-350-E03 features an electronic data sheet (EDS). Numerous production-relevant data are permanently stored in the EDS. The EDS data can be read out via files in the file system of the controller or via special registers.

Contents

EDS File "eds.ini"

Introduction

EDS data can be read out from the file "eds.ini".

Properties

- The file can be accessed via file system of the controller.
- For an FTP connection, the user must have administrator rights (e.g. user "admin") or system rights (e.g. user "system").
- The EDS file of the controller is located in the subdirectory "/System".
- This file is read-only.
- Formatting the flash disk has no impact on this file.

Path to EDS Files

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

File Structure

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

Example - Controller

This is an example of an EDS file belonging to a JCM-350:

```
;Jetter AG JetControl Electronic Data Sheet

[IDENTIFICATION]
Version = 0
Code = 928
Name = JCM-350
PcbRev = 01
PcbOpt = 00

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

```
[FEATURES]
Version = 1
MAC-Addr = 00:50:CB:00:05:F0
STX = 1
NVRegs = 6000
```

Section [IDENTIFICATION]

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

Name	Example	Description
Version	0	Version of this section
Code	928	Module code for JCM-350
Name	JCM-350	Corresponds to the information on the nameplate
PcbRev	01	Hardware revision
PcbOpt	00	Hardware option

Section [PRODUCTION]

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

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

Section [FEATURES]

In the section [FEATURES] special properties of the controller are specified. Properties, which have no entries in the file, are regarded as as non-existing by the controller.

Name	Example	Description
Version	1	Version of this section
MAC Addr	00:50:CB:00:05:F0	Ethernet MAC address
STX	1	Runtime environment for application program is available
NVRegs	6000	Number of remanent registers

Related Topics

- **EDS Registers** on page 27

EDS Registers

Introduction

Entries in the Electronic Data Sheet (EDS) can be read by the controller via EDS registers.

Register Numbers

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

Controller	Basic Register Number	Register Numbers
JCM-350	100000	100500 ... 100817

EDS Registers of a Controller

The following table lists the EDS registers of a controller, as well as their connection to the entries in the EDS file "/System/eds.ini". As there is only one register set, the required module has to be selected via module registers 500 and 501. The contents of the selected EDS are then displayed in the following registers.

Registers	Section in eds.ini	Name in eds.ini	Description
MR 500	-	-	Functional group: 0 Controller 1 JXM modules
MR 501	-	-	Module number (if MR 500 > 0)
MR 600	IDENTIFICATION	Version	Version of this section
MR 601		Code	Module code
MR 602 to MR 612		Name	Module name or controller name
MR 613		PcbRev	Hardware revision
MR 614		PcbOpt	Hardware revision
MR 700	PRODUCTION	Version	Version of this section
MR 701 to MR 707		SerNum	Serial number
MR 708		Day	Production date: Day
MR 709		Month	Production date: Month
MR 710		Year	Production date: Year
MR 711		TestNum	Internal usage
MR 712		TestRev	Internal usage
MR 800	FEATURES	Version	Version of this section
MR 801		MAC Addr	MAC address (manufacturer section)
MR 802		MAC Addr	MAC address (device section)
MR 805		STX	Runtime environment for application program

Registers	Section in eds.ini	Name in eds.ini	Description
MR 806		NVRegs	Number of remanent registers
MR 810		MotionControl	MC software

Related Topics

- **EDS File "eds.ini"** on page 25

3.3 Version Registers

Introduction

The operating system of the JCM-350-E03 provides several registers which can be used to read out the version numbers of the OS and its components. You will need this information when contacting the hotline of Jetter AG in case of a problem.

Contents

Hardware Revisions

Introduction

The controller JCM-350 features special registers which can be used to identify the hardware.

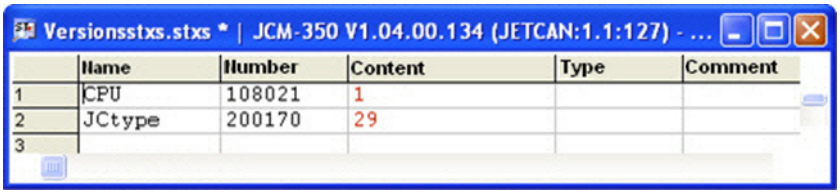
Overview of Registers

The following registers can be read to obtain the hardware revision:

Registers	Description
108021	Hardware revision - CPU board
200170	Controller type

Version Numbers in JetSym Setup

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



Related Topics

- **Software Versions** on page 31

Software Versions

Introduction

The controller JCM-350 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 JCM-350-E03 is a four-figure value.

1	.	2	.	3	.	4

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

Released Version

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

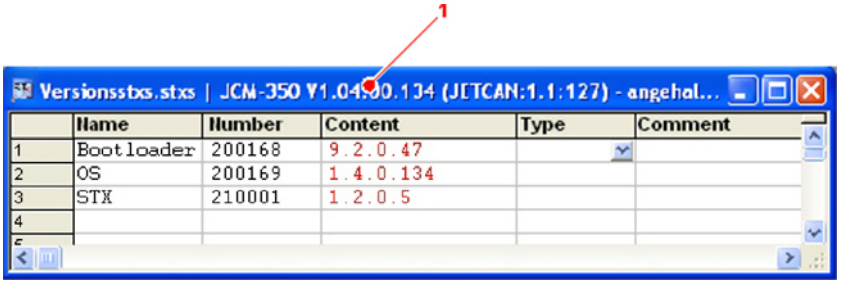
Overview of Registers

The following registers are used for reading out software versions:

Registers	Description
200168	Boot loader version
200169	Operating system version
210001	Version of the execution unit for the STX application program

Version Numbers in JetSym Setup

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



Number	Content	Description
1	V 1.04.00.134	OS version of the controller JetSym displays this information in the title bar of each setup window.

Related Topics

- **Hardware Revisions** on page 30

3.4 Identifying a JXM-IO-E02 via CAN Bus

Introduction

The module JXM-IO-E02 features an Electronic Data Sheet (EDS). Numerous production-relevant data are permanently stored in the EDS. EDS data can be read via CAN bus.

Inhalt

Electronic Data Sheet (EDS) and Software Version

Communication with JXM-IO-E02

Communication with the JXM-IO-E02 module takes place via CAN bus. As protocol the CANopen® standard is used. CANopen® is an open standard for networking and communication 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.

Electronic Data Sheet (EDS)

The Electronic Data Sheet (EDS) provides information clearly identifying the module. Data contained in the EDS are production-specific and are relevant for support purposes. If required, the data can be read using the object "Electronic Data Sheet" (0x4555).

JXM-IO-E02 - Software Version

Use the object "Detailed Software Version" (0x4559) to read out the version of the software running in the JXM-IO-E02. This read-only object supplies the same software version as object 0x100A, but in a 32-bit unsigned integer format which is compatible with the standard IP-type version numbers used at Jetter AG.

Example:

The 32-bit word 0x01070001 translates to a software version of 1.07.0.01.

Useful Documents

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.

Related Topics

- **Electronic Datasheet Object** on page 124
- **Detailed Software Version Object** on page 133

4 Mounting and Installation

Purpose of this Chapter

This chapter is for supporting you in mounting and installing the JCM-350-E03 as regards the following points:

- Planning the wiring of a JCM-350-E03
- Connecting sensors and actuators to the JCM-350-E03
- Installation
- CAN Bus - Project Work

Contents

4.1 Wiring

Purpose of this Chapter

This chapter covers wiring of the JCM-350-E03 and contains the following topics:

- Wiring principle
- Pin Assignment
- Example of Wiring
- Technical Specifications

Contents

Wiring Principle

Introduction

This chapter covers the wiring principle of the JCM-350-E03.

Wiring Principle

The JCM-350-E03 is connected through a wiring harness with external components, such as:

- Power Supply
- Controller
- Peripheral Module
- Sensors
- Actuators
- Indicator Lights

The wiring harness ends in a connector which is not included in the scope of delivery of the device. This connector is available as accessory.

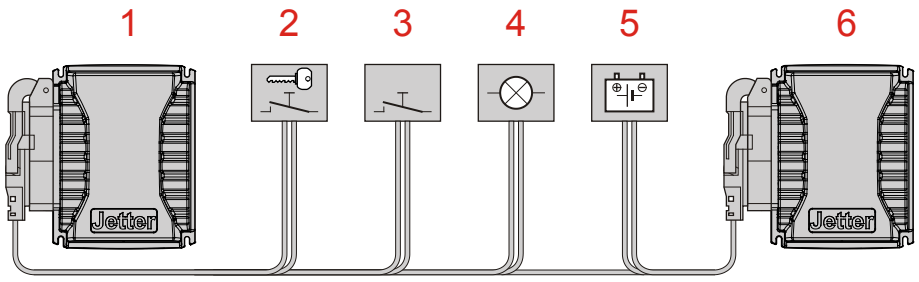
Connector Specification

The connector specification is listed below:

Connector Specification	
Manufacturer/Model	Tyco AMP
Article #	963484
Design	70-pin
Coding	A 1

Example

The diagram shows an example of a layout using a wiring harness.



Number	Description
1	Module JXM-IO-E02
2	Ignition lock
3	Door contact switch
4	Indicator light
5	Battery
6	Controller JCM-350-E03

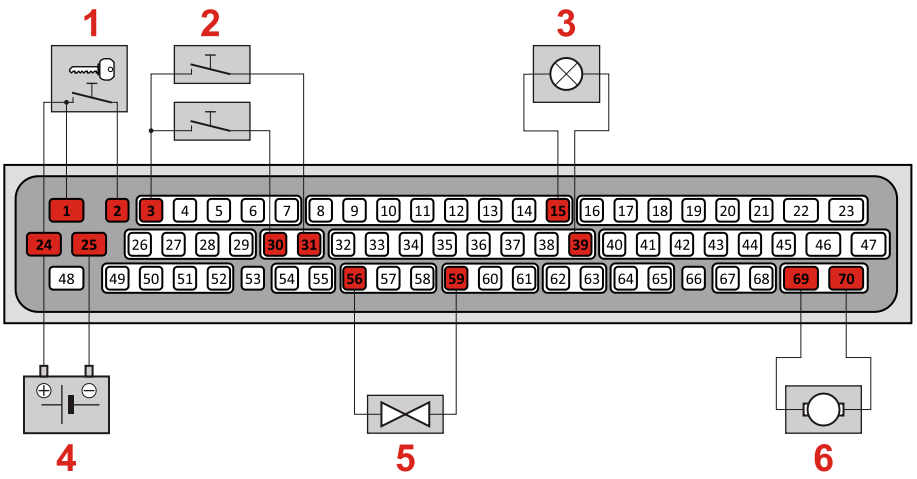
Example of Wiring Layout

Introduction

This chapter uses an example to show how the JCM-350-E03 is connected.

Example

The diagram shows an example of a wiring layout.



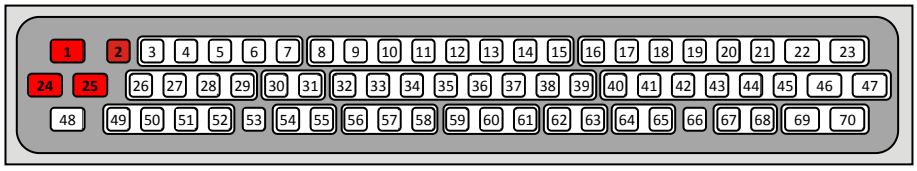
Number	Function
1	Ignition lock
2	Contacts (e.g. reed contacts or limit switch)
3	Indicator light
4	Power supply (battery)
5	Actuator (e.g. proportional valve)
6	Actuator (e.g. electric motor)

Connecting the Power Supply and the 5 V Output

Introduction

The following diagrams show the pin assignment of the connector (view from the front):

Power Supply



Pin	Function	Terminal number in vehicles
1	SAFETY FEED (+12 VDC or +24 VDC)	Terminal # 30
2	Ignition (+) (IGNITION FEED)	Terminal # 15
24	STANDARD FEED (+12 VDC or +24 VDC)	Terminal # 30
25	Ground	Terminal # 31

Ignition (IGNITION FEED)

- IGNITION FEED sources the digital electronics that control the inputs and outputs.
- The ignition must be active for the JCM-350-E03 to be active.
- The JCM-350-E03 will continue to run on a minimum input voltage of 5.9 V (on IGNITION FEED) in order to survive engine cranking (ISO 7637-2 Test Pulse 5 compliant). The JCM-350-E03 is designed to work with an input power voltage range of 8 V up to 32 V.
- The maximum current draw on this line is 2 A.
- Internal protection circuits protect against brief voltage drops on this line to ensure continued operation of the JCM-350-E03.

STANDARD FEED

- STANDARD FEED provides power for some of the outputs of the JCM-350-E03.
- The maximum current draw on this line is 52 A.
- However, internal current measurement will cut outputs if the current exceeds the 30 A limit. The current on STANDARD FEED is monitored by software.

SAFETY FEED

- SAFETY FEED provides power for some of the outputs of the JCM-350-E03.
- The maximum current draw on this line is 40 A.
- However, internal current measurement will cut outputs if the current exceeds the 30 A limit. SAFETY FEED is protected by solid state switches which also implements a hardware current limit of 30 A.

Note on Ignition

To start the JCM-350-E03, pin 2 (IGNITION FEED) must be connected with pin 24 (STANDARD FEED). The ignition control signal is issued when the key is in position "Ignition ON".

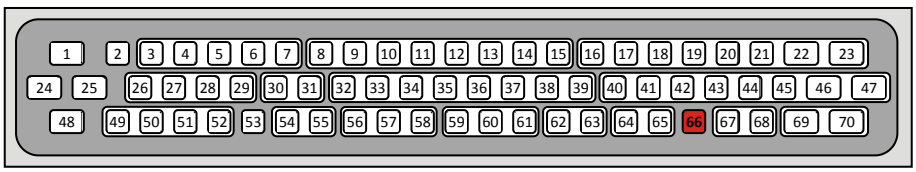
Power Supply - Technical Data

Parameter	Description
Rated voltage	DC 12 V or DC 24 V
Permissible voltage range	DC 8 ... 32 V
Current consumption at 12 V	tbd
Current consumption at 24 V	tbd

Note on Current Consumption

The base current consumption is measured shortly after switching on the JCM-350-E03 while there are no active output signals and input signals are not connected. Active outputs and also certain connected input signals will affect the current consumption.

Regulated 5 V Output



Pin	Function
66	Regulated 5 V output

Technical Data - Regulated Output

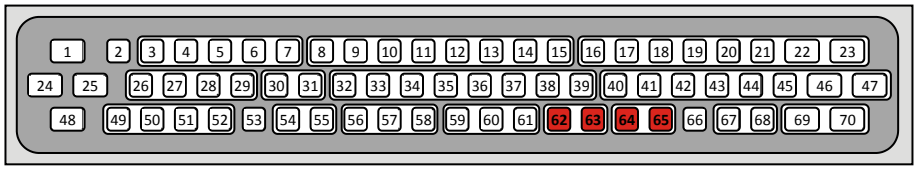
Parameter	Description
Regulated voltage	DC 5 V
Load current	max. 250 mA
Overcurrent detection	Yes

CAN Interface and Node ID

Introduction

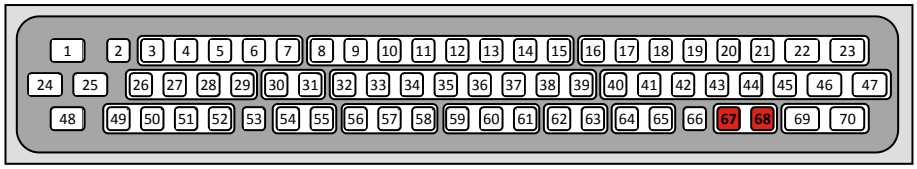
The following diagrams show the pin assignment of the connector (view from the front):

CANopen®



Pin	Function
62	OUT_CAN_L
63	OUT_CAN_H
64	IN_CAN_L
65	IN_CAN_H

Node ID



Pin	Function
67	Node ID (tri-state input # 1)
68	Node ID (tri-state input # 2)

Technical Data - Tri-State Inputs

Parameter	Description
Application	■ for device coding ■ as digital inputs
Type of inputs	Pull-up resistor to IGNITION FEED and pull-down resistor to ground
Tri-state detection	Tri-state operation is detected by a pull-down resistor to ground.
Rated voltage	IGNITION FEED
Threshold level OFF	$\leq$ 1.0 V
Threshold level ON	$\geq$ 4.0 V

Note

Note that because these inputs are tri-state enabled, they will always have bias voltage on the pin capable of sourcing current.

Calculating the Node ID Based on Tri-State Input State

The following table shows the effective node ID given that the default base ID of 0x10 is used:

State of pin 67	State of pin 68	CANopen® Node ID
Not Connected	Not Connected	0x10
Not Connected	OFF	0x11
Not Connected	ON	0x12
OFF	Not Connected	0x13
OFF	OFF	0x14
OFF	ON	0x15
ON	Not Connected	0x16
ON	OFF	0x17
ON	ON	0x18

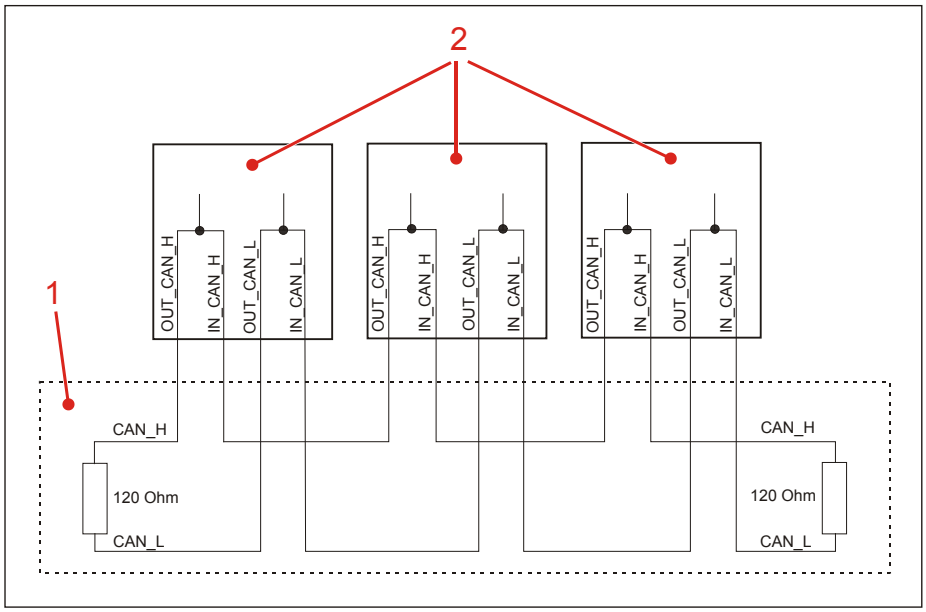
Related Topics

- **Specification - CAN Bus Cable** on page 43

Specification - CANopen® Bus Cable

Layout of CAN Bus Wiring

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



Number	Description
1	CAN bus
2	Jetter AG CANopen® devices

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

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

The CAN_L and CAN_H cables must be twisted together.

CAN Bus Cable Specification

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

Cable Lengths

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

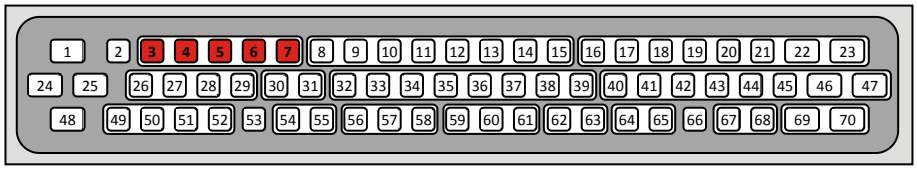
Baud Rate	Cable length	Stub length	Overall stub length
1000 kBaud	max. 25 m	max. 0.3 m	3 m
500 kBaud	max. 100 m	max. 1.0 m	39 m
250 kBaud	max. 200 m	max. 3.0 m	78 m
125 kBaud	max. 200 m	-	-

Connecting Digital Inputs and Outputs

Introduction

The following diagrams show the pin assignment of the connector (view from the front):

Digital Inputs

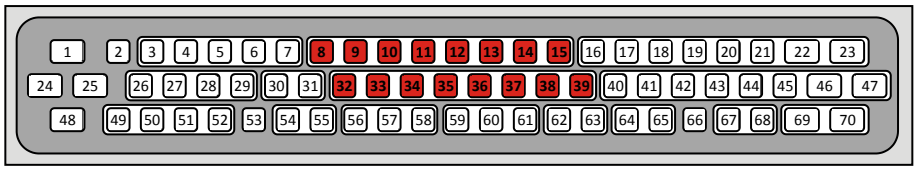


Pin	Description
3	Digital input 1
4	Digital input 2
5	Digital input 3
6	Digital input 4
7	Digital input 5

Technical Data - Digital Inputs IN 1 through IN 5

Parameter	Description
Type of inputs	Software selectable with either 2 kΩ pull-up to STANDARD FEED or 2 kΩ pull-down to ground.
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Threshold level OFF	≤ 1.0 V
Threshold level ON	≥ 3.5 V

Digital Universal I/Os (STANDARD)



Pin	Description
8	Universal I/O: IN 6 / OUT 1
9	Universal I/O: IN 7 / OUT 2
10	Universal I/O: IN 8 / OUT 3
11	Universal I/O: IN 9 / OUT 4
12	Universal I/O: IN 10 / OUT 5
13	Universal I/O: IN 11 / OUT 6
14	Universal I/O: IN 12 / OUT 7

Pin	Description
15	Universal I/O: IN 13 / OUT 8
32	Ground Return: IN 6 / OUT 1
33	Ground Return: IN 7 / OUT 2
34	Ground Return: IN 8 / OUT 3
35	Ground Return: IN 9 / OUT 4
36	Ground Return: IN 10 / OUT 5
37	Ground Return: IN 11 / OUT 6
38	Ground Return: IN 12 / OUT 7
39	Ground Return: IN 13 / OUT 8

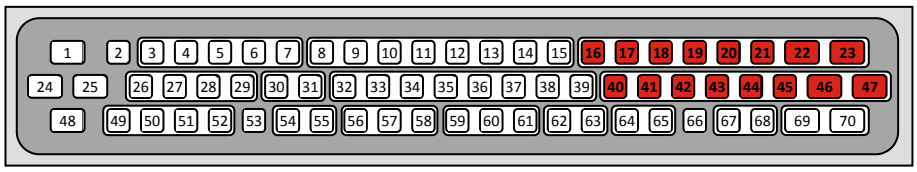
Technical Data - Digital Inputs IN 6 through IN 13

Parameter	Description
Type of inputs	Can be configured as active-high inputs
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Threshold level OFF	51 % of IGNITION FEED
Threshold level ON	51 % of IGNITION FEED
Input impedance	100 kΩ

Technical Data - Digital Outputs (STANDARD FEED)

Parameter	Description
Type of outputs	Active-high output
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Signal voltage OFF	< 1.0 V
Signal voltage ON	$U_{STANDARD}$ - 0.5 V
Load current of OUT 1 through OUT 8	max. 2.5 A
Maximum inrush current	tbd
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Digital Universal I/Os (SAFETY)



Pin	Description
16	Universal I/O: IN 14 / OUT 9
17	Universal I/O: IN 15 / OUT 10
18	Universal I/O: IN 16 / OUT 11
19	Universal I/O: IN 17 / OUT 12
20	Universal I/O: IN 18 / OUT 13
21	Universal I/O: IN 19 / OUT 14
22	Universal I/O: IN 20 / OUT 15
23	Universal I/O: IN 21 / OUT 16
40	Ground Return: IN 14 / OUT 9
41	Ground Return: IN 15 / OUT 10
42	Ground Return: IN 16 / OUT 11
43	Ground Return: IN 17 / OUT 12
44	Ground Return: IN 18 / OUT 13
45	Ground Return: IN 19 / OUT 14
46	Ground Return: IN 20 / OUT 15
47	Ground Return: IN 21 / OUT 16

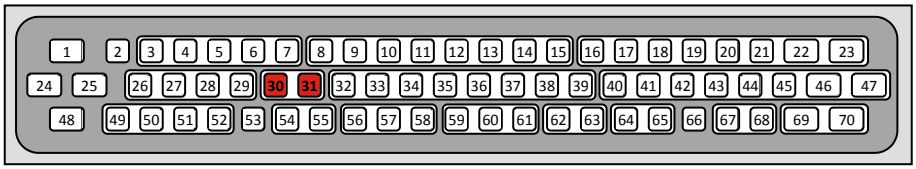
Technical Data - Digital Inputs IN 14 through IN 21

Parameter	Description
Type of inputs	Can be configured as active-high inputs
Rated voltage	SAFETY FEED
Permissible voltage range	DC 8 ... 32 V
Threshold level OFF	< 51 % of IGNITION FEED
Threshold level ON	> 51 % of IGNITION FEED
Input impedance	100 kΩ

Technical Data - Digital Outputs (SAFETY FEED)

Parameter	Description
Type of outputs	Active-high output
Rated voltage	SAFETY FEED
Permissible voltage range	DC 8 ... 32 V
Signal voltage OFF	< 1.0 V
Signal voltage ON	U_{SAFETY} - 0.5 V
Load current of OUT 9 through OUT 10	max. 2.5 A
Load current of OUT 11 through OUT 16	max. 5.0 A
Maximum inrush current	tbd
Can be switched off by electronic safety switch	Yes
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Switch Feed Outputs

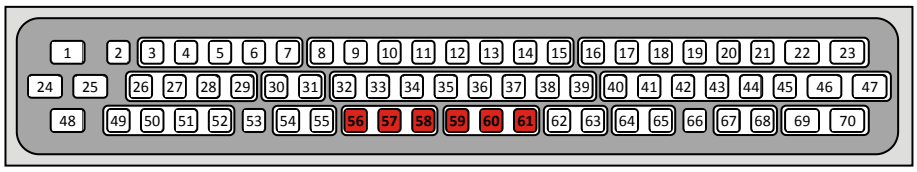


Pin	Description
30	Switch feed output 1
31	Switch feed output 2

Technical Data - Switch Outputs

Parameter	Description
Type of switch outputs	Active-high output
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Signal voltage OFF	< 1.0 V
Signal voltage ON	$U_{STANDARD}$ - 0.5 V
Load current	each 2.5 A max.
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

PWM Outputs



Pin	Description
56	PWM output 1
57	PWM output 2
58	PWM output 3
59	Ground Return: PWM output 1
60	Ground Return: PWM output 2
61	Ground Return: PWM output 3

Technical Data - PWM Outputs

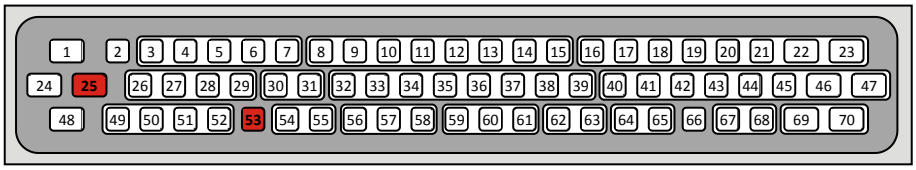
Parameter	Description
Operating Modes	■ Current-controlled output ■ PWM output with static duty cycle
Dither function	Yes, at PWM freq: 2 kHz
Resolution	8 bits
Load current	0 ... 2.5 A
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Connecting Analog Inputs and Outputs

Introduction

The following diagrams show the pin assignment of the connector (view from the front):

Analog Output

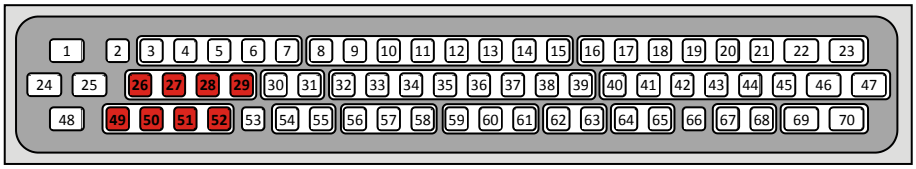


Pin	Description
25	Ground: Analog output
53	Analog Output

Technical Data - Analog Output

Parameter	Description
Voltage range at 50 mA	0 ... STANDARD FEED
Current range	0 ... 100 mA
Resolution	10 bits
Electrical isolation	none
Short circuit detection	Yes

Analog Inputs

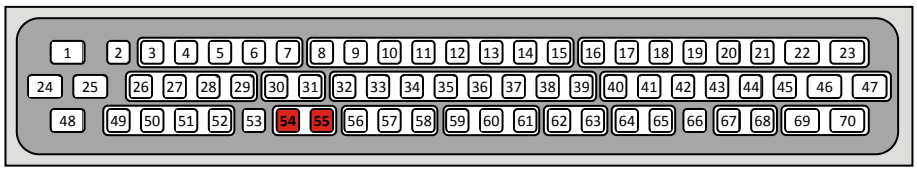


Pin	Description
26	Ground: Analog input 1
27	Ground: Analog input 2
28	Ground: Analog input 3
29	Ground: Analog input 4
49	Analog input 1
50	Analog input 2
51	Analog input 3
52	Analog input 4

Technical Data - Analog Inputs

Parameter	Description
Voltage range	■ 0 ... 5 V ■ 0 ... IGNITION FEED
Current range	■ 0 ... 20 mA ■ 4 ... 20 mA
Input impedance at 0 ... 5 V	100 kΩ
Input impedance at 0 ... IGNITION FEED	50 kΩ
Input impedance at 0 ... 20 mA	240 Ω
Resolution	10 bits
Electrical isolation	none

Frequency Inputs

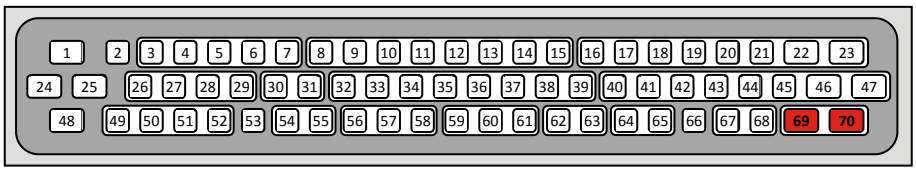


Pin	Description
54	Frequency input 1
55	Frequency input 2

Technical Data - Frequency Inputs

Parameter	Description
Application	■ as frequency counter ■ as two digital inputs
Type of inputs	Software selectable with either 2 kΩ pull-up to STANDARD FEED or 2 kΩ pull-down to ground.
Frequency measurement range	5 Hz ... 20 kHz
Measurement method	time-based
Result of measurement	Period of the signal in nanoseconds
Resolution	62.5 ns

H-Bridge Outputs



Pin	Description
69	H-bridge outputs
70	

Technical Data - H-Bridge

Parameter	Description
Application	■ used as H-Bridge ■ as two independent digital inputs
Rated output current	max. 2.5 A
Accuracy of current measurement (H-bridge)	< 100 mA
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

4.2 Installing the JCM-350-E03

Introduction

This chapter describes how to install the JCM-350-E03.

Contents

Installing the JCM-350-E03

Selecting a Place for Installation

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

- The installation surface must be made from one of the following materials:
  - aluminum plate
  - galvanized steel plate
  - lacquered steel plate
- The installation surface must be vertical.
- The installation surface must be level.
- The installation location must allow adequate air circulation.
- The installation location must be accessible for servicing.
- The installation location must be of sufficient size.

See also: **Physical Dimensions** on page 19

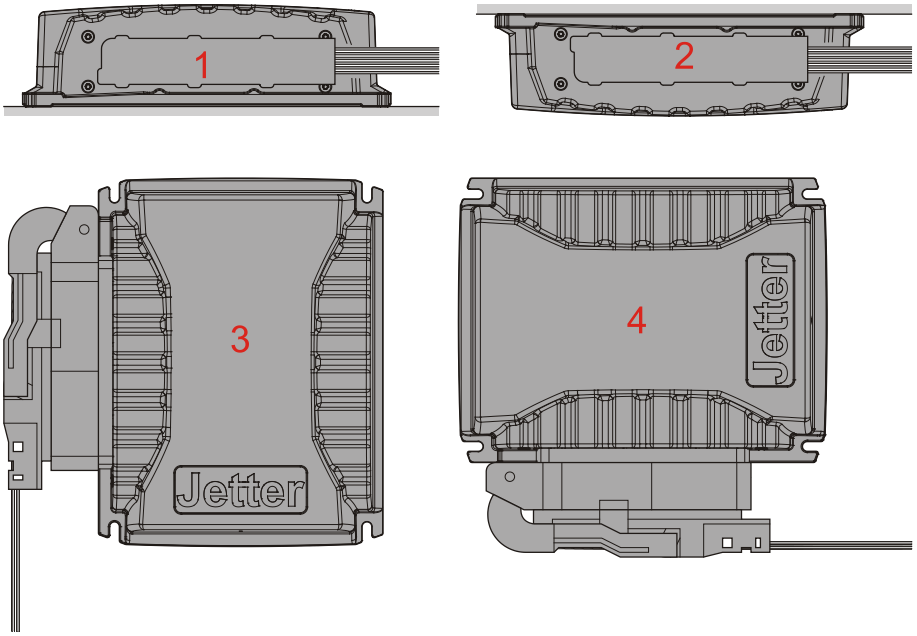
Avoiding Unsuitable Installation Locations

Do not install the device in inappropriate locations.
The following installation locations are not suited for mounting the JCM-350-E03:

Unsuitable installation location	Reason
Unventilated installation location	The device could overheat as heat builds up.
Stainless steel surfaces	Galvanic corrosion may occur between device and mounting surface
Installation location close to heat-sensitive materials	The materials could become warped or misshapen as a result of heat produced by the device.
Installation surfaces are uneven	The installation surface could become misshapen when fitting the device. Installation is unstable and precarious.

Permissible Installation Positions

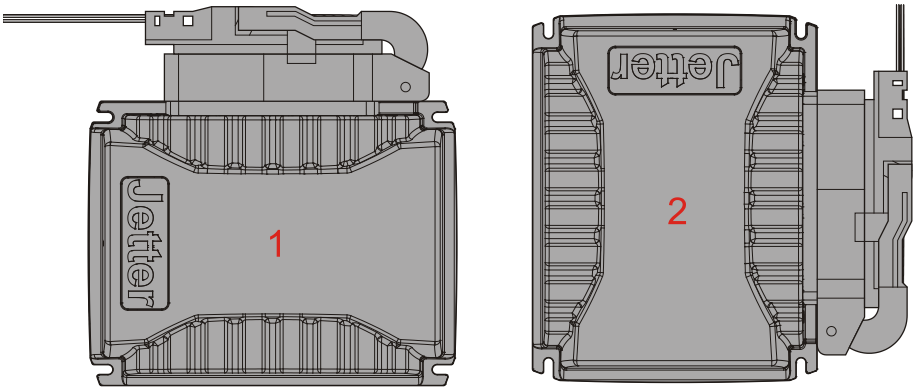
The diagram shows the positions permitted for installation.



Number	Permissible Installation Positions
1	horizontally, lying
2	horizontally, hanging
3	vertically, connector left
4	vertically, connector downwards

Prohibited Installation Positions

The diagram shows the positions prohibited for installation.



Number	Prohibited Installation Positions
1	vertically, connector upwards
2	vertically, pressure equalizing membrane upwards

Why are these installation positions prohibited?

- Vertically, connector upwards: The accumulation of moisture and water droplets in the connector can lead to current leakages and corrosion.
- Vertically, pressure equalizing membrane upwards: The accumulation of moisture and water droplets can block the hole which may impede pressure compensation.

Selecting Installation Material

Use the following installation material:

Part	Design
Screws/bolts	Size: M 5 x 15 Surface: galvanized Strenght class: 8.8
Washers	Size: 5.3 x 10 Surface: galvanized
Screw nuts	Size: M 5 Surface: galvanized Strenght class: 8.8

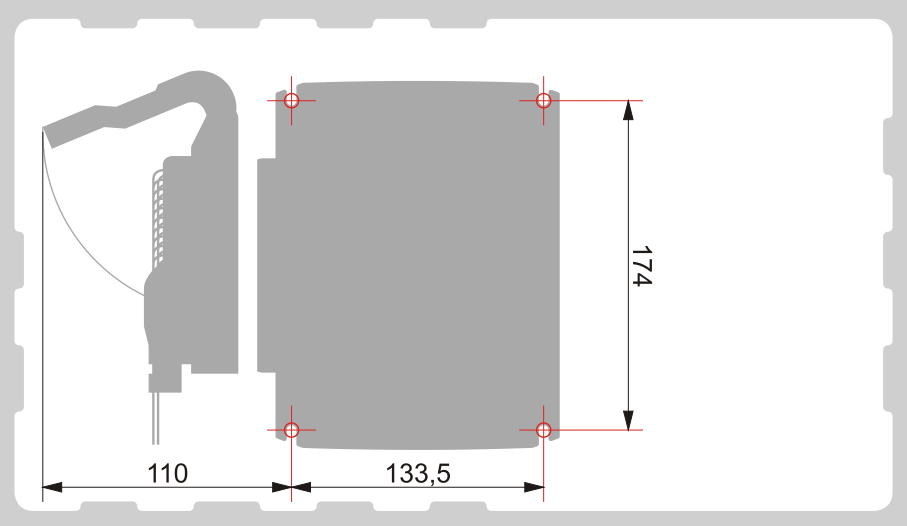
Avoid Improper Installation Material

Avoid installation material made from stainless steel. In connection with the housing material of the JCM-350-E03 galvanic corrosion may occur.

Preparing for Installation

Mark off the positions of the 4 mounting holes.

Center-punch the 4 holes.



If ...	... Then ...
the thickness of the mounting surface is ≥ 6 mm (steel) or ≥ 8 mm (aluminum)	drill the following holes: ■ Pre-drill Ø 4.2 mm. ■ Tap a thread M 5.

If ...	... Then ...
the thickness of the mounting surface is < 6 mm (steel) or < 8 mm (aluminum)	drill the following holes: ■ Drill the holes Ø 6 mm. ■ Deburr the holes.

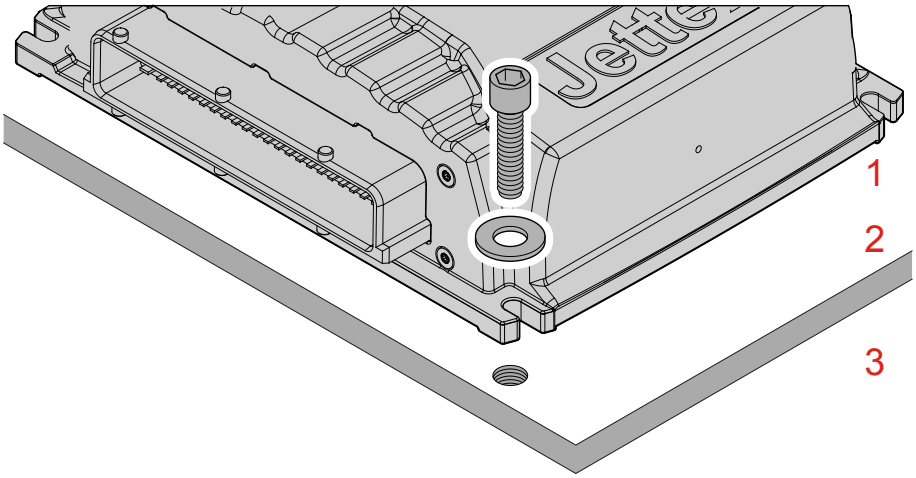
Notes on Installation

Direct contact between housing and mounting surface improves heat dissipation. Therefore:

- Install the device directly on the mounting surface.
- Do not use insulating material.
- Do not use spacers.

Installing the JCM-350-E03 (Tapped Holes)

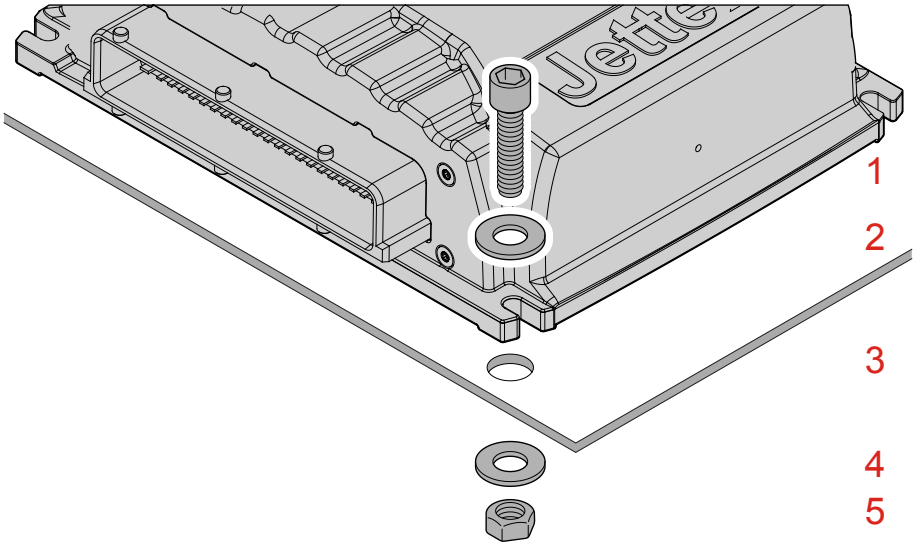
Screw the device down to the mounting surface.



Number	Description
1	Screw
2	Washer
3	Tapped hole

Installing the JCM-350-E03 (Through Holes)

Screw the device down to the mounting surface.



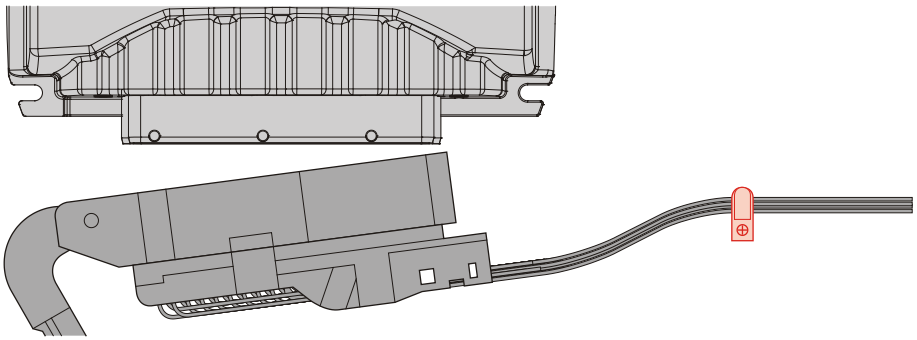
Number	Description
1	Screw
2	Washer
3	Through hole
4	Washer
5	Screw nut

Installing the Strain Relief

Install a strain reliever for the connection cable.

Take care to leave enough space for the connector.

The connectors should not be obstructed, so that it can be removed in the event of a service requirement.



5 Initial Commissioning

Introduction

This chapter covers the initial commissioning of the JCM-350-E03 with the aid of the following steps:

- Connecting the power supply and interfaces
- Installing and connecting a USB CAN adaptor
- Initial Commissioning Using the Programming Tool JetSym

JCM-350-E03 - Configuration

The JCM-350-E03 consists of the controller JCM-350 and the I/O module JXM-IO-E02 which are internally connected via CAN bus. The CAN bus is brought out to allow communication with other CANopen® nodes. The default node ID of the JXM-IO-E02 is 16, the default node ID of the JCM-350 is 127. This way, both components within the JCM-350-E03 can be addresses separately.

Contents

Preparatory Work for Initial Commissioning

Introduction

To be able to commission and program the JCM-350-E03 the following preparations are necessary:

- Wire the power supply, ignition and CAN interfaces
- Connect a USB CAN adaptor between controller and PC. Install the driver software for the given adaptor.

Default Values for JXM-IO-E02

The default values of the JXM-IO-E02 module are listed below:

- Baud rate: 250 kBaud
- CAN terminating resistor: 0x01 (resistor at the end of the CAN bus is enabled)
- Node ID: 0x10

Wiring the Controller

For more information please refer to **Wiring** on page 36. To wire the controller JCM-350-E03, proceed as follows:

Step	Action
1	Connect the following terminals with the power supply DC 8 - 32 V: ■ SAFETY FEED: **Pin 1** (terminal 30 in the vehicle) ■ Ignition **Pin 2** (terminal 15 in the vehicle) ■ STANDARD FEED: **Pin 24** (terminal 30 in the vehicle) ■ Ground: **Pin 25** (terminal 31 in the vehicle)
2	Connect a Sub-D connector (female) to IN_CAN (**pin 64** and **pin 65**) allowing to connect the USB CAN adaptor.
3	Make sure that there is a terminating resistor of 120 Ω at both ends of the CAN bus.
4	Energize the power supply. Make sure that the ignition is on. Otherwise the controller will not work.

Result: Now the controller is operational. To allow programming it can be connected with the USB CAN adaptor.

Supported USB CAN Adaptors

The following USB CAN adaptors are supported by the programming tool JetSym:

- **IXXAT Automation GmbH** (**http://www.ixxat.de http://www.ixxat.de**): The list of currently supported hardware can be found on the website of IXXAT Automation GmbH.
  We support the following driver versions: VCI version 3.3 and VCI version 2.18
- **PEAK-System Technik GmbH** (**http://www.peak-system.com http://www.peak-system.com**): The list of currently supported hardware can be found on the website of PEAK-System Technik GmbH.
  We support the following driver versions: Version 3.5.4.9547 or higher

Installing the USB CAN Adaptor

Prerequisites:

Before installing the USB CAN adaptor, **JetSym 4.3** or higher must be installed on the PC to be used.

To install the adaptor proceed as follows:

Step	Action
1	Insert the USB CAN adaptor into a USB port of your PC.
2	If the Hardware Wizard opens, close it.
3	Install the driver for the USB CAN adaptor.
4	Install the corresponding JetSym driver depending on the USB CAN adaptor used. **If ...** / **... Then ...** you use an adaptor by PEAK-Systems → proceed with step 5. you do not use an adaptor by PEAK-Systems → proceed with step 7.
5	Navigate in Windows Explorer to the directory **PcanDrv** located in the JetSym installation. Default location: **C:\Programme\Jetter\JetSym\Tools\PcanDrv**
6	Execute the file **PcanDrv.exe** and follow the instructions.
7	Plug the Sub-D connector of the adaptor into the IN_CAN port of the JCM-350-E03 (female Sub-D connector).

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

Related Topics:

- **Initial Commissioning in JetSym** on page 62

Initial Commissioning in JetSym

Introduction

JetSym is used to configure and program the controller JCM-350-E03. The following is detailed in this topic:

- Creating a project in JetSym
- Configuring the hardware/controller
- Initializing the JCM-350-E03

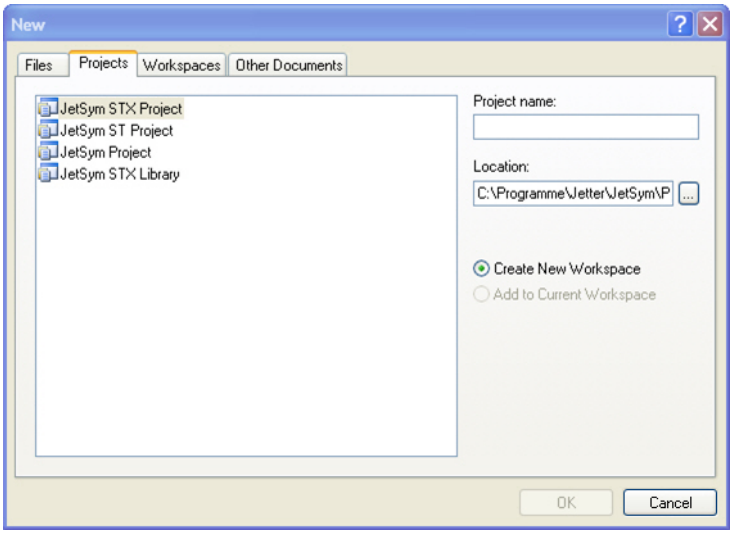
Prerequisites

The following requirements must be satisfied:

- JetSym is installed on the PC used.
- JetSym has been licensed (see online help in JetSym).
- Preparatory work for initial commissioning is completed and an active CANopen® connection between controller and PC has been established.

Creating a Project

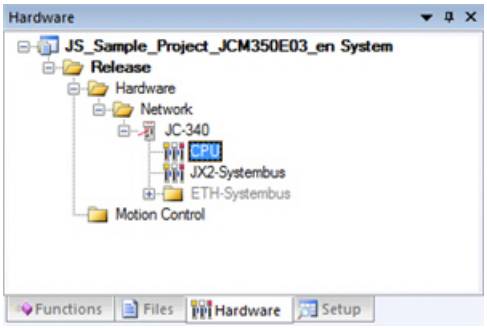
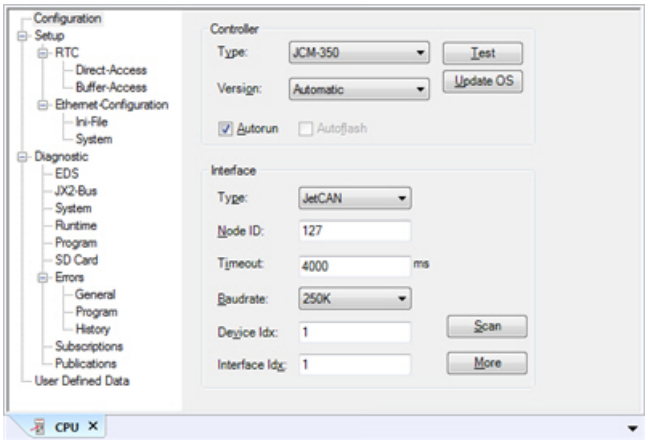
A new project for the programming is created in JetSym as follows:

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

Result: A project has now been created.

Configuring the Hardware

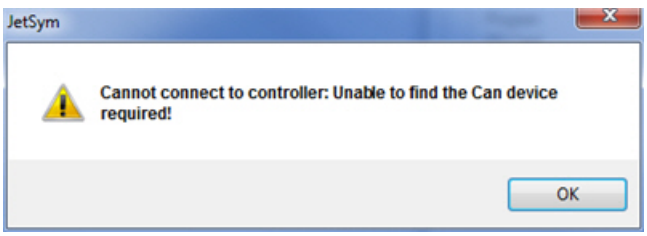
To establish a connection between JetSym and the controller, you need to configure the hardware as follows:

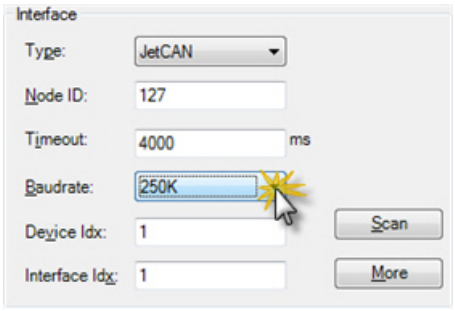
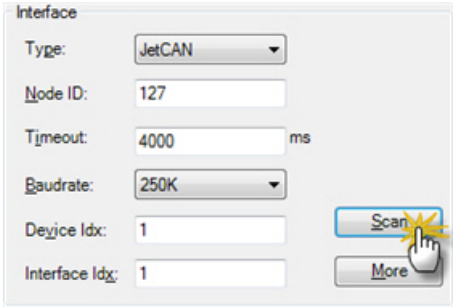
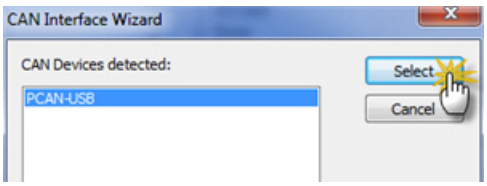
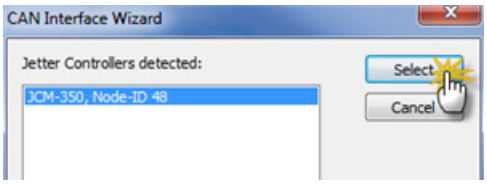
Step	Action
1	Switch to the **Hardware** view by clicking on the tab with the same name. 
2	Fully expand the **Hardware tree**.
3	Double-click on **CPU**. **Result:** The dialog box **Configuration** opens. 
4	Under **Controller/Type** select **JCM-350**.
5	Under **Interface/Type select JetCAN.**
6	Test the connection by clicking on the **Test** button. If the test fails, check the mechanical CAN connection between PC and JCM-350-E03 (also refer to the next topic "Possible Error Messages").
7	Save your settings using the shortcut **Ctrl + S**.

Result: The hardware settings are now configured in JetSym.

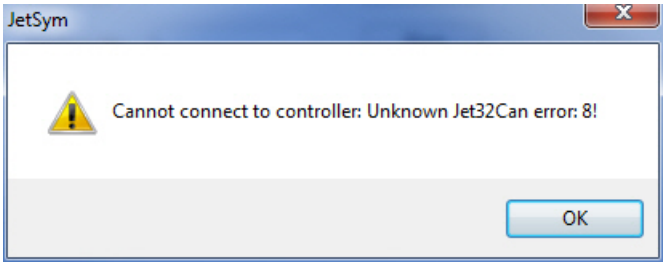
Possible Error Messages

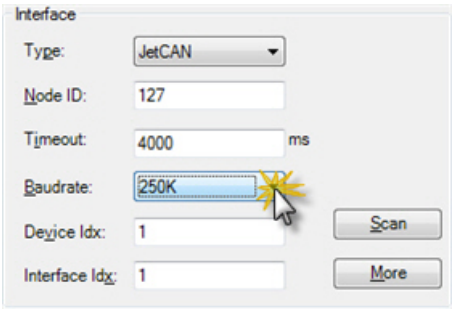
Error message # 1:



Possible Cause	Fix for this Problem
Selection of wrong controller type	Check whether your selection matches the controller type. If it does not match the type, open the configuration dialog and re-select the controller.
Wrong Baud rate	Check whether a Baud rate of 250 kBaud is set. If not, set it to 250 kBaud. 
Wrong node ID	If you don't know the node ID of your controller, scan the CAN bus for nodes:  Select the hardware that has been found as a result of the scan:  Then, select the controller that has been found as a result of the scan:  **Result:** The node ID of the controller is automatically entered into the corresponding box of the configuration mask.

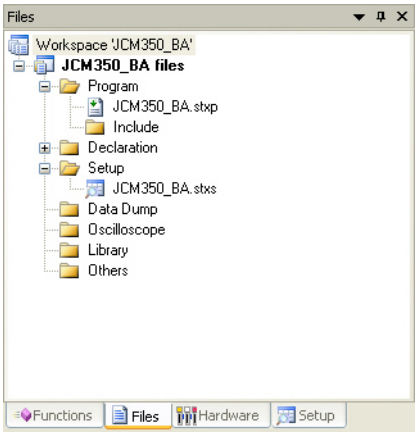
Error message # 2:



Possible Cause	Fix for this Problem
Wrong Baud rate	Check whether a Baud rate of 250 kBaud is set. Where necessary, restart JetSym. Enter the correct Baud rate and check the connection.  **ATTENTION:** If this error message is displayed after the restart of JetSym, re-boot your PC.

Initializing the JCM-350-E03

Proceed as follows to create a simple and executable program for initializing the JCM-350-E03:

Step	Action
1	Switch to **Files** view. 
2	Double-click on the program file (in our example **JCM350_BA.stxp)**. 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:
	`Var`
	`  Result: Int at %VL 1000000;`
	`End_Var;`
	
	`Task Main Autorun`
	`  Result := CanOpenInit(0, 127, 'Version: 01.00.0.00');`
	`End_Task;`

4	Press the **F7** key to trigger a project build. **Result:** A program which will run on the controller.
5	Press the shortcut **CTRL+F5**. **Result:** The program will be uploaded to the controller.

Result:

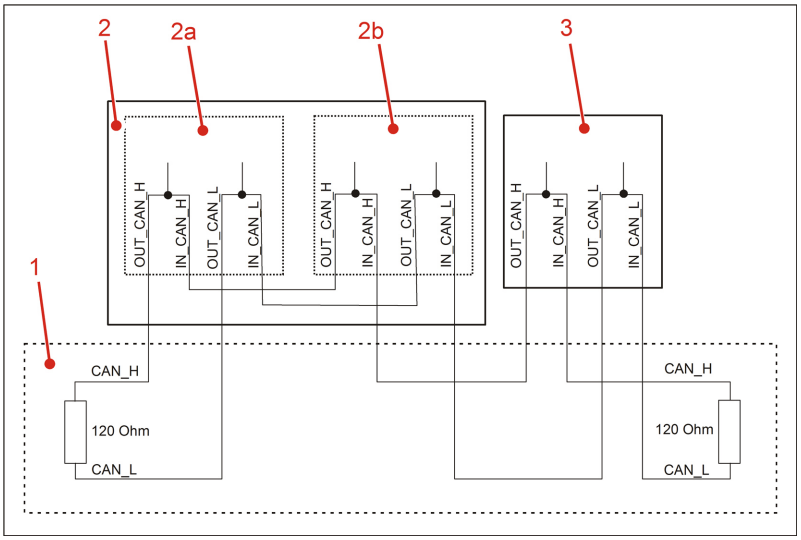
The program can now be enhanced. In **IntelliSense** (**Ctrl + Space Bar**), the CANopen® functions are now available.

Information on Communication with a JXM-IO-E02

Example - Wiring Diagram

The illustration below shows an wiring example of the following CANopen® devices by Jetter AG:

- Controller JCM-350-E03
- Peripheral module JXM-IO-E02



Number	Description	Node ID
1	CAN bus	
2	Controller JCM-350-E03 by Jetter	
2a	Controller JCM-350	0x7F (127 decimal)
2b	I/O module JXM-IO-E02	0x10 (16 decimal)
3	Separate I/O module JXM-IO-E02	0x11 (17 decimal) with user-configured tri-state inputs

CANopen® Interface - Restrictions

During initial commissioning the following restrictions/limitations of the CANopen® interface on the JXM-IO-E02 must be taken into account:

- PDOs are not user configurable.
- PDOs are transmitted only asynchronous on request.

Communication with Peripheral Modules

The following information supports you in commissioning peripheral modules, such as JXM-IO-E02:

- Initialize the controller as described in the manual of JCM-350-E03.
- Send an RTR frame to the peripheral module. This parameter is needed once in order to prompt the peripheral module to send the required data to the controller.

JetSym STX Sample Program

The following program fragment shows how the states of the digital inputs on the JXM-IO-E02 can be read by a Jetter controller, such as JCM-350.

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;

  Event_Time = 100;
  Inhibit_Time = 20;
End_Const;

Var
//State of the digital inputs
    Data_Inputs: Word;
    SW_Version: String;
End_Var;


Task Main Autorun

// Software version of the controller
SW_Version := 'v4.3.0';

// Initializing CAN 0

CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// Entering process data to be sent
CanOpenAddPDORx(CAN_CONTROLLER_0,
CANOPEN_PDO1_RX(NodeID_Node_1), 2, CANOPEN_WORD,
sizeof(Data_Inputs), Data_Inputs, Event_Time, Inhibit_Time,
CANOPEN_ASYNCPDORTRONLY);

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

End_Task;
```

Related Topics:

- **CANopen® Objects** on page 95

6 CANopen® STX API

Introduction

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

The CANopen® Standard

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

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

Application

These STX functions are used in communication between the controller JCM-350-E03 and e.g. the peripheral modules JXM-IO-E02, JXM-IO-E09, JXM-IO-E10, JXM-IO-E11 and JXM-MUX.

Documentation

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

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

Contents

STX Function CanOpenInit

Introduction

Calling up the CanOpenInit () function initializes one of the CAN busses. The JCM-350-E03 then automatically sends the heartbeat message every second with the following communication object identifier (COB-ID): Node ID + 0x700

Function Declaration

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

Function Parameters

The CanOpenInit () function has the following parameters.

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

Return Value

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

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

Parameter CANNo

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

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

Using this Function

Initializing the CAN bus 0. The JCM-350-E03 has node ID 20 (0x14).

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

How it Works

During initialization, the JCM-350-E03 processes the following process steps:

Step	Description
1	First, the bootup message is sent as a heartbeat message.
2	As soon as the JCM-350-E03 goes into Pre-operational status, it sends the Pre-operational heartbeat message.

Access to the Object Directory

The Object Directory can only be accessed via SDO, if the JCM-350-E03 is in "Pre-operational" status.

NMT Messages

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

Related Topics:

- **STX Function CanOpenSetCommand** on page 72

STX Function CanOpenSetCommand

Introduction

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

Function Declaration

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

Function Parameters

The CanOpenSetCommand () function has the following parameters.

Parameter	Description	Value
CANNo	CAN channel number	0 ... CANMAX
iType	Command selection	CAN_CMD_HEARTBEAT: Only the own heartbeat status is changed. CAN_CMD_NMT: The heartbeat status is changed for all other devices or for a specific device on the CAN bus.
Value	Selection of the heartbeat status for command CAN_CMD_HEARTBEAT: CAN_HEARTBEAT_STOPPED (0x04) CAN_HEARTBEAT_OPERATIONAL (0x05) CAN_HEARTBEAT_PREOPERATIONAL (0x7F) Selection of the heartbeat status for command CAN_CMD_NMT (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)	

Note

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

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

Parameter CANNo

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

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters Command not known

Using the Function (Example 1)

The own heartbeat status should be set to Operational.

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

Using the Function (Example 2)

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

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

Using the Function (Example 3)

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

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

STX Function CanOpenUploadSDO

Introduction

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

Function Declaration

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

Function Parameters

The CanOpenUploadSDO () function has the following parameters.

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

Return Value

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

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

Parameter CANNo

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

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

Parameter DataType

The following data types can be received.

Byte types	CANopen® format	Jetter format
1	CANOPEN_INTEGER8 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

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

"Busy" Error Codes

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

SDOACCESS_STILLUSED
Another task is communicating with the same node ID.

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

SDOACCESS_ILLCMD
The response to the request is invalid.

SDOACCESS_ABORT
The device with the node ID was aborted.

SDOACCESS_SYSERROR
General internal error

Macro Definitions

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

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

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

Using this Function

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

JetSym STX Program

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

```
#Include "CanOpen.stxp"

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

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

Task main autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

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

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

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

When SDOACCESS_FINISHED(busy) Continue;

If (SDOACCESS_ERROR(busy)) Then
// Troubleshooting

End_If;
//      ...
//      ...
End_Task;
```

STX Function CanOpenDownloadSDO

Introduction

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

Function Declaration

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

Function Parameters

The CanOpenDownloadSDO () function has the following parameters.

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

Return Value

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

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

Parameter CANNo

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

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

Parameter DataType

The following data types can be received.

Byte types	CANopen® format	Jetter format
1	CANOPEN_INTEGER8 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

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

"Busy" Error Codes

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

SDOACCESS_STILLUSED
Another task is communicating with the same node ID.

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

SDOACCESS_ILLCMD
The response to the request is invalid.

SDOACCESS_ABORT
The device with the node ID was aborted.

SDOACCESS_BLKSIZEINV
Communication error with Block Download

SDOACCESS_SYSERROR
General internal error

Macro Definitions

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

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

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

Using this Function

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

JetSym STX Program

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

```
#Include "CanOpen.stxp"

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

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


Task main autorun

Var
    SW_Version: String;
End_Var;

SW_Version := 'v4.3.0.2004';

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

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

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

When SDOACCESS_FINISHED(busy) Continue;

If (SDOACCESS_ERROR(busy)) Then
// Troubleshooting

End_If;
```

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

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

End_Task;
```

STX Function CanOpenAddPDORx

Introduction

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

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

Notes

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

Function Declaration

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

Function Parameters

The CanOpenAddPDORx () function has the following parameters.

Parameter	Description	Value
CANNo	CAN channel number	0 ... CANMAX
CANID	CAN identifier 11-bit 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	Volume of data for the global variable VarAddr	
VarAddr	Global variable into which the received value is entered	
EventTime	Time lag between two telegrams (> Inhibit Time)	
InhibitTime	Minimum time lag between two telegrams received (< EventTime)	
Paramset	Parameter bit-coded	

Return Value

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

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

Parameter CANNo

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

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

Parameter CANID

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

Macro definitions:

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

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

Example for calling up the macro:

CANOPEN_PDO2_RX (64)

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

Default CAN Identifier Distribution

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

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

Parameter DataType

The following data types can be received.

Byte types	CANopen® format	Jetter format
1	CANOPEN_INTEGER8 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	-

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

Parameter Paramset

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

CANOPEN_ASYNCPDORTRONLY

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

CANOPEN_ASYNCPDO

Receive asynchronous PDOs.

CANOPEN_PDOINVALID

PDO not received. Disk space is reserved.

CANOPEN_NORTR

PDO cannot be requested by RTR (Remote Request).

CANOPEN_29BIT

Use 29-bit identifier

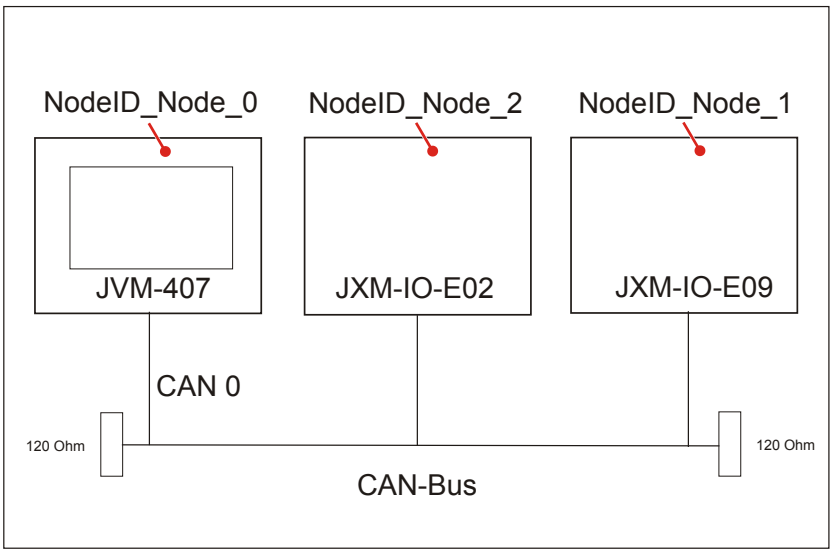
Default: 11-bit identifier

Using this Function

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

JetSym STX Program

JCM-350-E03 with node ID 10 wants to receive a PDO from two CANopen® devices with node ID 64 and 102. The function CanOpenAddPDORx () is called up for this purpose. After running the program, the JCM-350-E03 receives the cyclic PDO telegrams.




```
#Include "CanOpen.stxp"

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

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

Task main autorun

Var
    SW_Version: String;
End_Var;
```

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

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

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

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

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

End_Task;
```

STX Function CanOpenAddPDOTx

Introduction

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

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

Notes

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

Function Declaration

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

Function Parameters

The CanOpenAddPDOTx () function has the following parameters.

Parameter	Description	Value
CANNo	CAN channel number	0 ... CANMAX
CANID	CAN identifier 11-bit 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	Volume of data for the global variable VarAddr	
VarAddr	Global variable into which the value to be sent is entered	
EventTime	Time lag between two telegrams (> Inhibit Time)	
InhibitTime	Minimum time lag between two telegrams sent (< EventTime)	
Paramset	Parameter bit-coded	

Return Value

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

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

Parameter CANNo

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

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

Parameter CANID

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

Macro definitions:

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

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

Example for calling up the macro:

CANOPEN_PDO2_RX (64)

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

Default CAN Identifier Distribution

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

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

Parameter DataType

The following data types can be received.

Byte types	CANopen® format	Jetter format
1	CANOPEN_INTEGER8 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	-

Byte types	CANopen® format	Jetter format
1	CANOPEN_INTEGER8 CANOPEN_UNSIGNED8	Byte
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

Parameter Paramset

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

CANOPEN_ASYNCPDORTRONLY
Send asynchronous PDOs by receiving an RTR frame.

CANOPEN_ASYNCPDO
Send asynchronous PDO.

CANOPEN_PDOINVALID
PDO not sent.

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

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

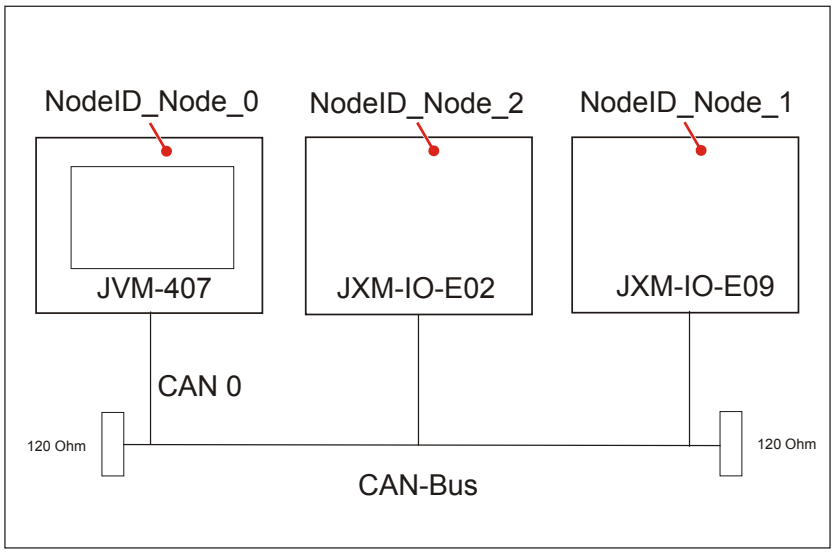
Using this Function

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

JetSym STX Program

JCM-350-E03 sends process data to two CANopen® devices with the node ID 74 and 112. After running the program and for changes, the JCM-350-E03

sends cyclic PDO telegrams every 3,000 ms (Event Time). As a maximum, the PDO telegram is sent every 10 ms (Inhibit Time).




```
#Include "CanOpen.stxp"

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

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

Task main autorun

Var
    SW_Version: String;
End_Var;

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

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

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

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

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

End_Task;
```

7 CANopen® Objects

Introduction

This chapter covers the CANopen® objects implemented on the JCM-350-E03 and their functions, as well as the permanently mapped process data objects (PDO).

JCM-350-E03 - Configuration

The JCM-350-E03 consists of the controller JCM-350 and the I/O module JXM-IO-E02 which are internally connected via CAN bus. The CAN bus is brought out to allow communication with other CANopen® nodes. The default node ID of the JXM-IO-E02 is 16, the default node ID of the JCM-350 is 127. This way, both components within the JCM-350-E03 can be addresses separately.

Restrictions

Due to design constraints the following restrictions/limitations apply to the CANopen® interface of the JXM-IO-E02.

- SDO expedited transfer only supports 4 byte transfers. Any smaller data element must be extended to 32 bit before the SDO transfer.
- SDO segmented transfer is only supported on certain objects. Most notably the OS update feature makes use of segmented transfer, but also some other objects that need to transfer strings implement SDO segmented transfer for this purpose. Unless an object is documented to support segmented transfers, assume that it does not.
- SDO block transfer is NOT implemented.
- PDOs are not user configurable.
- PDOs are transmitted only asynchronous on request unless otherwise specified.
- Although emergency messages are transmitted to notify of detected faults, the CANopen® emergency handling system is not fully implemented.
- The Error Register does not save its state in non volatile memory. After each reset or power cycle, the error list is cleared.

Contents

7.1 CANopen® Object Dictionary for JCM-350-E03

Purpose of this Chapter

This chapter describes the CANopen® objects implemented on the JCM-350 and their function.

Supported Objects

The following objects are supported by the operating system for JCM-350:

Index (hex)	Object Name	Object (Code)	Type	see
1000	Device Type	VAR	Unsigned32	**Page** 97
1001	Error Register	VAR	Unsigned8	**Page** 97
1002	Manufacturer Status	VAR	Unsigned32	-
1003	Pre-defined Error Field	ARRAY	Unsigned32	**Page** 97
1008	Manufacturer Device Name	VAR	String	**Page** 97
1009	Manufacturer Hardware Version	VAR	String	**Page** 97
100A	Manufacturer Software Version	VAR	String	**Page** 97
100B	Node ID	VAR	Unsigned32	**Page** 97
1017	Producer Heartbeat Time	VAR	Unsigned16	**Page** 97
1018	Identity	RECORD	Identity (23h)	-
1200	Server 1 - SDO Parameter	RECORD	SDO Parameter (22h)	-
1201	Server 2 - SDO Parameter	RECORD	SDO Parameter (22h)	-
1203	Server 3 - SDO Parameter	RECORD	SDO Parameter (22h)	-
1203	Server 4 - SDO Parameter	RECORD	SDO Parameter (22h)	-

Contents

Supported CANopen® SDO Objects

Device Type (index 0x1000)

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

Index	Sub-Index	Default	Description	Attributes
0x1000	0	0x0000012D	Device type	ro (read only)

Error Register (index 0x1001)

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

Index	Sub-Index	Default	Description	Attributes
0x1001	0	0	Error Register	ro (read only)

This object implements the CANopen® Error Register functionality.

Bit 0 = Generic error

None of the other bits are currently in use.

Pre-Defined Error Field (index 0x1003)

The structure of the object "Pre-Defined Error Field" is shown in the following table.

Index	Sub-Index	Default	Description	Attributes
0x1003	0	0	Number of errors entered in the Array's Standard Error Field	rw (read & write)
	1	0	Most recent error 0 indicates no error	ro (read only)
	2 ... 254	-	Earlier Errors	ro (read only)

This object shows a history list of errors that have been detected by the JCM-350. The maximum length of the list is 254 errors. The list content is deleted on restart.

Composition of the Standard Error Field

2-byte LSB: Error Code

2-byte MSB: Additional information

Manufacturer Device Name (index 0x1008)

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

Index	Sub-Index	Default	Description	Attributes
0x1008	0	JCM-350-E03	Hardware name	const

Manufacturer Hardware Version Object (Index 0x1009)

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

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

Manufacturer Software Version Object (Index 0x100A)

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

Index	Sub-Index	Default	Description	Attributes
0x100A	0		Software version of the application program that runs on the JCM-350-E03	const

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

Node ID Object (Index 0x100B)

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

Index	Sub-Index	Default	Description	Attributes
0x100B	0		Own Node ID	ro (read only)

Producer Heartbeat Time Object (Index 0x1017)

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

Index	Sub-Index	Default	Description	Attributes
0x1017	0	1,000 [ms]	Heartbeat time	rw (read & write)

7.2 CANopen® Object Dictionary for JXM-IO-E02

Purpose of this Chapter

This chapter describes the CANopen® objects implemented on the JXM-IO-E02 and their function.

Supported Objects

A summary of the objects covered in this document are given in the table below. There are also a few more objects which are mandatory according to the CANopen® specification which are not covered in this document:

Index (hex)	Object Name	Object (Code)	Type	see
1000	Device Type	VAR	Unsigned32	**Page** 101
1001	Error Register	VAR	Unsigned8	**Page** 101
1003	Pre-defined Error Field	ARRAY	Unsigned32	**Page** 101
100A	Manufacturer Software Version	VAR	String	**Page** 101
1017	Producer Heartbeat Time	VAR	Unsigned16	**Page** 101
1018	Identity	RECORD	Identity (23h)	-
2000	Features	ARRAY	Unsigned32	**Page** 101
2100	Digital Inputs	ARRAY	Unsigned32	**Page** 103
2101	Universal I/O	ARRAY	Unsigned32	**Page** 105
2102	Tri-state Inputs	ARRAY	Unsigned32	**Page** 107
2103	Switch Feed Outputs	ARRAY	Unsigned32	**Page** 109
2200 - 2203	Analog Input	ARRAY	Unsigned32	**Page** 110
2210	Voltage Sense Analog Input	ARRAY	Unsigned32	**Page** 112
2211	Feed Currents	ARRAY	Unsigned32	**Page** 113
2300	Analog Output	ARRAY	Unsigned32	**Page** 114
2400 - 2402	PWM Output	ARRAY	Unsigned32	**Page** 116
2500	H-Bridge	ARRAY	Unsigned32	**Page** 120
2600, 2601	Frequency Input	ARRAY	Unsigned32	**Page** 122
4554	OS Update	ARRAY	Unsigned32	**Page** 124
4555	Electronic Datasheet	ARRAY	Unsigned32	**Page** 124
4556	System Parameters	ARRAY	Unsigned32	**Page** 125
4559	Detailed Software Version	ARRAY	Unsigned32	**Page** 133
5000	User EEPROM Access	ARRAY	Unsigned32	**Page** 134

Contents

Objects Ranging from Index 0x1000 through 0x2000

Device Type (Index 0x1000)

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

Index	Sub-Index	Default	Description	Attributes
0x1000	0	0x00030191	Type of device	ro (read only)

Error Register (Index 0x1001)

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

Index	Sub-Index	Default	Description	Attributes
0x1001	0	0	Error Register	ro (read only)

This object implements the CANopen® Error Register functionality.

Bit 0 = Generic error
Bit 1 = Current Errors
Bit 2 = Voltage Errors
Bit 3 = Temperature Errors
Bit 4 = Communication error
Bit 5 = Parameter mismatch
Bit 7 = Manufacturer-specific error, for example, hardware error

None of the other bits are currently in use.

Pre-defined Error Field (Index 0x1003)

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

Index	Sub-Index	Default	Description	Attributes
0x1003	0	0	Number of errors entered in the Array's Standard Error Field	rw (read & write)
	1	0	Most recent error 0 indicates no error	ro (read only)
	2 ... 64	-	Earlier Errors	ro

This object shows a history list of errors that have been detected by the JXM-IO-E02. The maximum length of the list is 64 errors. The list content is deleted on restart.

By writing the value 0 to sub-index 0, the list can be cleared, as per the CANopen® specification.

Composition of the Standard Error Field

2-byte LSB: Error Code

2-byte MSB: Additional information

Manufacturer Software Version (Index 0x100A)

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

Index	Sub-Index	Default	Description	Attributes
0x100A	0		Software version	const

Use only the STX function CanOpenUploadSDO () to determine the version of the software running in the JXM-IO-E02.

The version string is at least 9 characters long and is of the format "2.00.0.00". The first digit is the major revision followed by the minor revision and the branch and beta indicators (which will usually be zero). This value is read-only (ro).

Producer Heartbeat Time (Index 0x1017)

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

Index	Sub-Index	Default	Description	Attributes
0x1017	0	1,000 [ms]	Heartbeat time	rw (read & write)

The legal range for values is 250 ... 65,535.

Features Object (Index 0x2000)

The structure of the object "Features" is shown in the following table.

Index	Sub-Index	Default	Description	Attributes
0x2000	0		Features Object	ro (read only)

The "Features" object is provided for compatibility reasons.

Digital Inputs Object (Index 0x2100)

Digital Inputs (Index 0x2100)

The structure of the object "Digital Inputs" is shown in the following table. This object is for configuring the digital inputs IN 1 through IN 5 and for obtaining their states.

Index	Sub-Index	Default	Description	Attributes
0x2100	0	6	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Active-high / Active-low Selection	rw (read & write)
	3	1	Not used	
	4	0	Process value 0: Input States (2 bits/channel)	ro
	5	0	Process value 1: Input States (1 bit/channel)	ro
	6	5	Parameter 0: Number of Inputs	ro

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-Index 2 is used to set inputs IN 1 through IN 5 to either active-high (internal pull down resistor) or active-low (internal pull up resistor) mode.
- A bit value of "0" selects active-low (input state "OFF") and a bit value of "1" selects active-high (input state "ON"). The value can also be read back to confirm.
- Sub-index 2 uses the one bit per channel data structure described below:
  - Bit 0: Digital input IN 1
  - Bit 1: Digital input IN 2
  - Bit 2: Digital input IN 3
  - Bit 3: Digital input IN 4
  - Bit 4: Digital input IN 5

Sub-Index 4

The function of sub-index 4 is described below:

- Sub-index 4 can be read to obtain the latest measured states of IN 1 through IN 5.
- Sub-index 4 returns the data in a two bit per channel format (provided for backwards compatibility).
- Sub-index 4 uses the two bit per channel data structure described below:
  - Bit 1, 0: Digital input IN 1
  - Bit 3, 2: Digital input IN 2
  - Bit 5, 4: Digital input IN 3
  - Bit 7, 6: Digital input IN 4
  - Bit 9, 8: Digital input IN 5

- In the two bit per channel configuration, the following data values are possible:
  - 0b00: Not used
  - 0b01: Input state OFF
  - 0b10: Input state ON
  - 0b11: Not used

Sub-Index 5

The function of sub-index 5 is described below:

- Sub-index 5 can be read to obtain the latest measured states of IN 1 through IN 5.
- Sub-index 5 returns the data in the one bit per channel data structure described below:
- Sub-index 5 uses the one bit per channel data structure described below:
  - Bit 0: Digital input IN 1
  - Bit 1: Digital input IN 2
  - Bit 2: Digital input IN 3
  - Bit 3: Digital input IN 4
  - Bit 4: Digital input IN 5
- In the one bit per channel configuration, the following data values are possible:
  - 0: Input state OFF
  - 1: Input state ON

Sub-Index 6

Sub-index 6 can be read to obtain the number of available inputs. In this case, five inputs are available.

Universal I/O Object (Index 0x2101)

Universal I/O

A universal I/O can be used as digital input or digital output. Therefore, universal I/Os must be configured correspondingly.

- Any universal I/O can be used as digital input or output.
- If a universal I/O is used as digital input, the related digital output must be disabled (OFF).

Universal I/O (Index 0x2101)

The structure of the object "Universal I/O" is shown in the following table. This object is for configuring universal I/Os. It allows either to read out the state of the digital inputs IN 6 through IN 21, or to set the digital outputs OUT 1 through OUT 16.

Index	Sub-Index	Default	Description	Attributes
0x2101	0	6	Number of entries	ro (read only)
	1	0	Enabling channel	rw (read & write)
	2	0	Disabling channel	rw
	3	4	Not used	
	4	0	Process value 0: Reading back output states / reading out input states	rw
	5	0	Process value 1: Output States	rw
	6	16	Parameter 0: Number of inputs/outputs	ro

Sub-Index 1

The function of sub-index 1 is described below:

- Sub-index 1 can be used to enable individual channels.
- To enable a channel enter its number (1 through 16) into sub-index 1.
- Reading out sub-index 1 will always return the value "0".

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-index 2 can be used to disable individual channels.
- To disable a channel enter its number (1 through 16) into sub-index 2.
- Reading out sub-index 2 will always return the value "0".

Sub-Index 4

The function of sub-index 4 is described below:

- Sub-index 4 can be read to obtain the latest measured states of IN 6 through IN 21.
- Or it can be read to obtain the states of outputs OUT 1 through OUT 15.
- In sub-index 4 each bit is assigned to a channel:
  - Bit 0: Channel 1 (IN 6 or OUT 1)

	▪ Bit 1: Channel 2 (IN 7 or OUT 2) ▪ ... ▪ Bit 14: Channel 15 (IN 20 or OUT 15) ▪ Bit 15: Channel 16 (IN 21 or OUT 16) ▪ If a universal I/O is used as digital input, the related digital output must be disabled (OFF).
Sub-Index 5	The function of sub-index 5 is described below: ▪ Sub-index 5 can be used to set or reset the digital outputs OUT 1 through OUT 16. ▪ In sub-index 5 each bit is assigned to a channel: ▪ Bit 0: Channel 1 (OUT 1) ▪ Bit 1: Channel 2 (OUT 2) ▪ ... ▪ Bit 14: Channel 15 (OUT 15) ▪ Bit 15: Channel 16 (OUT 16) ▪ Depending on the bit value, the output state is as follows: ▪ 0: Output state is OFF ▪ 1: Output state is ON
Sub-Index 6	Sub-index 6 can be read to obtain the number of available inputs/outputs. In this case, 16 inputs/outputs are available.

Tri-State Inputs Object (Index 0x2102)

Purpose of Tri-State Inputs

Tri-state inputs are generally used for obtaining the node ID or changing the default node ID (device coding). However, in applications where device coding is not required, these inputs can be freely used as general purpose digital inputs. This may be the case if only one JCM-350-E03 or JXM-IO-E02 is connected to the CAN bus. The System Parameters object (index 0x4556, sub-index 38) allows disabling the "Tristate Coding Enable" flag by writing "0" to it.

Tri-State Inputs (Index 0x2102)

The structure of the object "Tri-State Inputs" is shown in the following table. Read this object to obtain the states of the tri-state inputs. The object "Tri-State Inputs" is read-only.

Index	Sub-Index	Default	Description	Attributes
0x2102	0	6	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Not used	
	3	1	Not used	
	4	0	Process value 0: Input states	ro
	5	0	Process value 1: Unused	
	6	5	Parameter 0: Number of Inputs	ro

Sub-Index 4

The function of sub-index 4 is described below:

- Sub-index 4 can be read to obtain the latest measured states of the two tri-state inputs.
- Because each input can be in one of three different input states, sub-index 4 uses the two bit per channel data structure described below:
  - Bit 1, 0: Tri-state input 1 (pin 67)
  - Bit 3, 2: Tri-state input 2 (pin 68)
- For each input, the following values can be returned:
  - 0b00: Tri-state (not connected)
  - 0b01: Input state OFF
  - 0b10: Input state ON
  - 0b11: Not used

Sub-Index 6

Sub-index 6 can be read to obtain the number of available inputs. In this case, two inputs are available.

Calculating the Node ID Based on Tri-State Input State

The following table shows the effective node ID given that the default base ID of 0x10 is used:

State of pin 67	State of pin 68	CANopen® Node ID
Not Connected	Not Connected	0x10
Not Connected	OFF	0x11
Not Connected	ON	0x12
OFF	Not Connected	0x13
OFF	OFF	0x14
OFF	ON	0x15
ON	Not Connected	0x16
ON	OFF	0x17
ON	ON	0x18

Changing the Default Node ID 0x10

The node ID is stored in the internal EEPROM and is read during the boot process. For special applications it is possible to change the value stored in the EEPROM by using the object "System Parameters" (index 0x4556, sub-index 4).

Switch Feed Output Object (Index 0x2103)

Switch Feed Outputs (Index 0x2103)

The structure of the object "Switch Feed Output" is shown in the following table. This object is for enabling or disabling the two switch feed outputs.

Index	Sub-Index	Default	Description	Attributes
0x2103	0	6	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Not used	
	3	4	Not used	
	4	0	Process value 0: Unused	
	5	0	Process value 1: Output state	rw (read & write)
	6	2	Parameter 0: Number of Outputs	ro

Sub-Index 5

The function of sub-index 5 is described below:

- Write to Sub-Index 5 to set the output state (enable/disable) of each channel.
- Sub-index 5 uses the one bit per channel data structure described below:
  - Bit 0: Switch feed output 1
  - Bit 1: Switch feed output 2
- For each channel the following values can be entered:
  - 0: Disable the switch feed output
  - 1: Enable the active high switch feed output

Sub-Index 6

Sub-index 6 can be read to obtain the number of available switch feed outputs.

Analog Input Objects (Index 0x2200 through 0x2203)

Analog Input (Index 0x2200 through 0x2203)

The structure of the objects "Analog Input" is shown in the following table. This object is for configuring the analog inputs 1 through 4. The analog input signal can be read out as process value.

Index	Sub-Index	Default	Description	Attributes
0x2200 - 0x2203	0	7	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Functional mode	rw (read & write)
	3	0x30	Not used	
	4	0	Process value 0: Analog input signal	ro
	5	0	Process value 1: Analog input signal [mV]	ro
	6	8.191	Parameter 0: Max. output value	ro
	7	40.000	Parameter 1: Max. output value	ro

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-index 2 can be used to select between two modes of operation. One of these modes supports Automatic Gain Control (AGC). The other mode of operation supports either voltage measurement or current measurement.
- To select the functional mode, set or reset bit 0 and bit 4:
  - Bit 0 = 0: AGC disabled
  - Bit 0 = 1: AGC enabled
  - Bit 4 = 0: Voltage measurement
  - Bit 4 = 1: Current measurement
- With AGC enabled, the analog input will be able to measure input signals in the range of 0 ... 40 V.

  With AGC disabled, the analog input will be able to measure input signals in the range of 0 ... 5 V.
- With current measurement enabled, the analog input can be used for 0 or 4 ... 20 mA current signals.

  Effectively this mode changes the analog input's impedance to 240 Ω. 20 mA generates a 4.8 V signal, 4 mA generates a 960 mV signal, and 0 mA results in a 0 V input.
- To allow for backwards compatibility, writing a value of 0x81 to Sub-Index 2 will disable the AGC mode. The value of 0x90 will disable the current input mode to enable voltage measurement.

Sub-Index 4	The function of sub-index 4 is described below: ▪ Sub-index 4 can be read to obtain the value of the latest measured analog input signal. ▪ With AGC enabled, the measured value will range between 0 ... 8,191. ▪ With AGC disabled, the measured value will range between 0 ... 1,023.
Sub-Index 5	The function of sub-index 5 is described below: ▪ Sub-index 5 can be read to obtain the value of the latest measured analog input signal, too. ▪ Sub-index 5 also reports the measured analog signal, but the reported value is in millivolt (mV) units. ▪ With AGC enabled, the measured value will range between 0 ... 40,000. ▪ With AGC disabled, the measured value will range between 0 ... 5,000.
Sub-Index 6	The function of sub-index 6 is described below: ▪ Sub-index 6 can be read to obtain the maximum value that can be output via Sub-Index 4.
Sub-Index 7	The function of sub-index 7 is described below: ▪ Sub-index 7 can be read to obtain the maximum value that can be output via Sub-Index 5.

Voltage Sense Analog Input Object (Index 0x2210)

Voltage Sense Analog Input (Index 0x2210)

The structure of the object "Voltage Sense Analog Input" is shown in the following table. This read-only object returns as process value the measured state of the three power feeds.

Index	Sub-Index	Default	Description	Attributes
0x2210	0	6	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Not used	
	3	0	Not used	
	4	0	Process value 0: Standard Feed Voltage [mV]	ro
	5	0	Process value 1: Ignition Feed Voltage [mV]	ro
	6	0	Process value 2: Safety Feed ON/OFF	ro

Sub-Index 4

The function of sub-index 4 is described below:

- Sub-index 4 reports the measured voltage of STANDARD FEED in millivolts.

Sub-Index 5

The function of sub-index 5 is described below:

- Sub-index 5 reports the measured voltage of IGNITION FEED in millivolts.

Sub-Index 6

The function of sub-index 6 is described below:

- Sub-Index 6 will simply report whether the SAFETY FEED (after the safety switch/relay) is enabled or disabled:
  - 0: SAFETY FEED disabled
  - 1: SAFETY FEED enabled
- This object does not have an analog measurement.

Feed Currents Object (Index 0x2211)

Feed Currents (Index 0x2211)

The structure of the object "Feed Currents" is shown in the following table. This read-only object provides the latest measurements of the Standard Feed and Safety Feed currents.

Index	Sub-Index	Default	Description	Attributes
0x2211	0	2	Number of entries	ro (read only)
	1	0	STANDARD FEED current measurement	ro
	2	0	SAFETY FEED current measurement	ro

Sub-Index 1

The function of sub-index 1 is described below:

- Sub-index 1 reports the measured current of STANDARD FEED in milliamp.

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-index 2 reports the measured current of SAFETY FEED in milliamp.

Analog Output Object (Index 0x2300)

Analog Output (Index 0x2300)

The structure of the object "Analog Output" is shown in the following table. This object is for configuring the analog output. Also, the analog output voltage/current can be set as process value.

Index	Sub-Index	Default	Description	Attributes
0x2300	0	5	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Functional Mode	rw (read & write)
	3	0x05	Not used	
	4	0	Process value 0: Output Voltage	rw
	5	0	Process value 1: Output Current	rw

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-index 2 can be used to select between the following modes of operation.
  - 0x00: Disabled: No output generated
  - 0x01: Constant output current
  - 0x02: Constant output current (ratiometric value specified)
  - 0x03: Constant output current (absolute value specified)
- To select one of the above modes, write the corresponding value to sub-index 2.
- When reading sub-index 2, the currently set mode is returned.
  The following information can be obtained:
  - 0x00: Disabled: No output generated
  - 0x01: Constant output current
  - 0x02: Constant output current (ratiometric value specified)
  - 0x03: Constant output current (absolute value specified)
  - 0x08: Short-circuit to ground fault has been detected

Sub-Index 4

The function of sub-index 4 is described below:

- Sub-index 4 is used to enter the analog output voltage.
- With mode "Constant output voltage (ratiometric value specified)" enabled, the value will range between 0 ... 1,023.
  This value range relates to 0 ... 100 % of the input voltage.
- With mode "Constant output voltage (absolute value specified)" enabled, the value specifies the output voltage in mV units.
  If a value larger than this maximum is specified, the output will clip.
- The analog output's maximum output voltage will always be slightly less than STANDARD FEED voltage.

	▪ Sub-index 4 can be read to obtain the recently measured output voltage in mV units.
Sub-Index 5	The function of sub-index 5 is described below: ▪ Sub-index 5 is used to set the analog output current. ▪ With mode "Constant output current" enabled, the value specifies the output current in 10 µA units. ▪ With mode "Constant output voltage (ratiometric/absolute value specified)" enabled, the value specifies the desired maximum output current. If the specified output voltage causes the output current to exceed this value, the output is clipped to control the output current. ▪ The analog output's maximum output voltage will always be slightly less than STANDARD FEED voltage. ▪ Sub-index 5 can be read to obtain the recently measured output current in 10 µA units.

Objects "PWM Output" (Index 0x2400 through 0x2402)

PWM Output 1 - 3 (Index 0x2400 through 0x2402)

The structure of the objects "PWM Output" is shown in the following table. These objects are for configuring the three PWM outputs. Also, the controlled output current or a PWM duty cycle can be set as process value.

Index	Sub-Index	Default	Description	Attributes
0x2300	0	11	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Functional mode	rw (read & write)
	3	0	Not used	
	4	0	Process value 0: Output current	rw
	5	0	Process value 1: Duty cycle	rw
	6	2.500	Parameter 0: Max. value	ro
	7	1.023	Parameter 1: Max. value	ro
	8	0	Predictor parameter	rw
	9	0	Proportional parameter	rw
	10	0	Integrator parameter	rw
	11	0	PWM predictor auto-tune function	rw

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-index 2 can be used to select between the following modes of operation.
  - 0x01: Current-controlled PWM output
  - 0x02: PWM output with static duty cycle
- To select one of the above modes, write the corresponding value to sub-index 2.
- In static PWM duty-cycle output mode the output current will not be controlled.

  However, it will be monitored. If the measured current exceeds a user set threshold, the PWM output will be disabled and a fault will be reported by the JXM-IO-E02. The maximum value is to be entered into sub-index 6.

Sub-Index 4

The function of sub-index 4 is described below:

- In current-controlled PWM output mode, write to sub-index 4 to set the output current.
- The value is in the range of 0 ... 2499 mA.
- Sub-index 4 can be read to obtain the recently measured output current in mA units.

Sub-Index 5

The function of sub-index 5 is described below:

- Sub-index 5 is used to set the PWM duty cycle.
- The value is in the range of 0 ... 2499 mA. This value corresponds to a duty cycle between 0 ... 100 %.
- Use the mode "PWM Output with Static Duty Cycle" in order to use the PWM output as a digital output.
- Reading sub-index 5 returns the most recent PWM duty cycle as a value in the range 0 ... 1,023.

Sub-Index 6

The function of sub-index 6 is described below:

- Sub-index 6 can be read to obtain the maximum value that can be input via Sub-Index 4.

Sub-Index 7

The function of sub-index 7 is described below:

- Sub-index 7 can be read to obtain the maximum value that can be input via Sub-Index 5.

Sub-Index 8

The function of sub-index 8 is described below:

- In "Current-Controlled PWM Output" mode sub-index 8 is used to set the Predictor parameters for the current control algorithm.
- This parameter is an unsigned 16-bit word where the least significant byte is the divisor and the most significant byte is the multiplier.
- The least significant byte of this parameter is not allowed to be zero because it is a divisor.

Sub-Index 9

The function of sub-index 9 is described below:

- In "Current-Controlled PWM Output" mode sub-index 9 is used to set the Proportional parameters for the current control algorithm.
- This parameter is an unsigned 16-bit word where the least significant byte is the divisor and the most significant byte is the multiplier.
- The least significant byte of this parameter is not allowed to be zero because it is a divisor.

Sub-Index 10

The function of sub-index 10 is described below:

- In "Current-Controlled PWM Output" mode sub-index 10 is used to set the Integrator parameters for the current control algorithm.
- This parameter is an unsigned 16-bit word where the least significant byte is the divisor and the most significant byte is the multiplier.
- The least significant byte of this parameter is not allowed to be zero because it is a divisor.

Current Control

In "Current-Controlled PWM Output" mode the PWM duty cycle is controlled using the above three parameters in the following formula:

$$PWM_{DutyCycle} = \frac{Pre_{mul} \cdot Current_{Demand}}{Pre_{div}} + \frac{Pro_{mul} \cdot Error}{Pro_{div}} + \frac{Int_{mul}}{Int_{div} \cdot IntegratedError}$$

Where:

- Pre_{mul} and Pre_{div} are the Predictor multiplication and division factors (sub-index 8),
- Pro_{mul} and Pro_{div} are the Proportional multiplication and division factors (sub-index 9),
- Int_{mul} and Int_{div} are the Integrator multiplication and division factors (sub-index 10),
- $Current_{Demand}$ is the user input in milliamp,
- Error is the difference between the measured and commanded output current (also in milliamp).
- Integrated Error is the integral of the error signal.

Calculating the PWM Pulse Control Factor

When a new output current is requested, the "Error" and "Integrated Error" terms are zero. The output duty cycle is therefore calculated based on the user input and the Predictor parameters. To ensure that this first output level is accurate, the predictor parameter must be set for the load that it will be driving.

After the initial duty cycle calculation, the PWM algorithm uses the difference between the measured output current and the current demand to adjust the PWM duty cycle. The proportional and integrator parameters influence how fast the algorithm responds to a difference between the measured and demanded current. These two parameters also determine how much overshoot there will be.

All three parameters are highly dependent on the load that is being driven. Therefore it is the user’s responsibility to tune these parameters for their own application.

If the PWM is already driving an output at a certain current level and a new output current is requested, the algorithm will not use the predictor parameters but instead it will use the current output current to calculate the new duty cycle. This method reduces the sensitivity to incorrect predictor parameters, but does not remove it – these parameters will still affect the normal operation.

Sub-Index 11

The function of sub-index 11 is described below:

- Sub-index 11 offers an automatic tuning of the Predictor parameter.
- To activate this function, write a 16-bit current value to sub-index 11.
- The system will then attempt to drive this current on the PWM and calculate what the predictor parameters must be in order to accurately guess the PWM duty cycle when a new current is requested.
- While this function is still running, the Predictor Parameter in sub-index 8 will read as "0".

	▪ As soon as the function completes (this may take up to 10 seconds, but is usually faster), the calculated Predictor parameters are available for reading from sub-index 8. The PWM algorithm will also use these parameters immediately. ▪ However, the newly calculated parameters will not be written to non-volatile memory. When the JXM-IO-E02 is reset or power cycled, these parameters will fall back to the previous parameters stored in the non-volatile memory. It is up to the user to first test the new parameters, read them from sub-index 8, and then write it to the System Parameters for permanent storage.
No Load Detection	The function "No Load Detection" is described below: ▪ No load detection can be activated for each PWM channel individually. ▪ When "Current Control" mode is selected, no load will be detected if the duty cycle reaches maximum and the load current remains below the specified threshold. ▪ In the "Static PWM Duty Cycle Output" mode, no load is detected whenever the duty cycle is non-zero and the load current is below the threshold. ▪ The no-load threshold is set in object "System Parameter" (index 0x4556, sub-index 40).

H-Bridge Object (Index 0x2500)

H-Bridge (Index 0x2500)

The structure of the object "H-Bridge" is shown in the following table. This object is for configuring the H-Bridge. Also, this object can be used to read the output states. It returns the PWM duty cycle when the H-Bridge is in a PWM-controlled mode.

Index	Sub-Index	Default	Description	Attributes
0x2500	0	7	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Functional mode	rw (read & write)
	3	0	Not used	
	4	0	Process value 0: Current measured	ro
	5	0	Process value 1: Output states / duty cycle	rw
	6	1.023	Parameter 0: Max. output value	ro
	7	7	Parameter 1: Bridge configuration	ro

Sub-Index 2

The function of sub-index 2 is described below:

- Sub-index 2 can be used to select between the following modes of operation.
  - 0x01: The two output channels (pins 69 and 70) are used as independent digital outputs.
  - 0x02: The output connected to pin 69 is a PWM-controlled high-side output, whereas the output connected to pin 70 is always low.
  - 0x04: The output connected to pin 70 is a PWM-controlled high-side output, whereas the output connected to pin 69 is always low.
- To select one of the above modes, write the corresponding value to sub-index 2.

Sub-Index 4

The function of sub-index 4 is described below:

- Sub-index 4 can be read to obtain the recently measured current in mA units.
- Note that this measurement is not available when the H-Bridge outputs are used as independent digital outputs.

Sub-Index 5	The function of sub-index 5 is described below:
	▪ When the H-Bridge outputs are used as two independent digital outputs, the least significant byte sets the output state:
	  ▪ Bit 1, 0: Pin 69 is set as output
	  ▪ Bit 5, 4: Pin 70 is set as output
	▪ In the configuration as two independent digital outputs, the following data values are possible:
	  ▪ 0b00: Tri-state output
	  ▪ 0b01: Output state is OFF
	  ▪ 0b10: Output state is ON
	▪ In PWM-controlled mode, a value in the range of 0 ... 1023 sets the PWM duty cycle.
	▪ In PWM-controlled mode, sub-index 5 is used to set the PWM duty cycle.

Sub-Index 6	The function of sub-index 6 is described below: ▪ Sub-index 6 can be read to obtain the maximum value for the duty cycle that can be input via Sub-Index 5.

Frequency Input Objects (Index 0x2600 through 0x2601)

Frequency Inputs (Index 0x2600 through 0x2601)

The structure of the objects "Frequency Input" is shown in the following table. This object is for configuring input pins 54 and 55 as frequency inputs or as simple digital inputs. In frequency input mode, the period length of the incoming signal is measured.

Index	Sub-Index	Default	Description	Attributes
0x2600 - 0x2601	0	7	Number of entries	ro (read only)
	1	0	Not used	
	2	0	Functional mode	rw (read & write)
	3	1	Not used	
	4	0	Process value 0: Period length [ns]	ro
	5	0	Process value 1: Digital input state	ro
	6	0xFFFFFFFF	Parameter 0: Frequency maximum value	ro
	7	0	Pulse Count	ro

Sub-Index 2

The function of sub-index 2 is described below:

- The functional mode is selected in sub-index 2 by entering the following values:
  - 0: Frequency input mode (no pull-up/pull-down resistor)
  - 1: Digital input (active-low)
  - 2: Digital input (active-high)
  - 3: Frequency input (with pull-up resistor)
  - 4: Frequency input (with pull-down resistor)
- Reading sub-index 2 returns the current functional mode.

Sub-Index 4

The function of sub-index 4 is described below:

- In frequency input mode, sub-index 4 can be read to obtain the value of the latest measured period length.
- The returned value is a 32-bit unsigned integer specifying the period length of the signal in nanoseconds.
- This result is updated every 17 - 18 cycles of the external signal.

Sub-Index 5

The function of sub-index 5 is described below:

- In digital input mode, sub-index 5 can be read to obtain the current state of input pin 54 or 55.
- The following values are possible:
  - 0: Input state OFF
  - 1: Input state ON

Sub-Index 7

The function of sub-index 7 is described below:

- The two Frequency Input circuits will always count pulses on these inputs regardless of their input mode. The pulse period must not be less than 1 millisecond and the pulse must be active for at least 1 millisecond each period in order to be counted.
- The Pulse Count always starts at zero at power on/reset. The value is also reset to zero every time that it is read via this SDO.
- The returned value is a 32-bit unsigned integer.

OS Update (Index 0x4554) and EDS Objects (Index 0x4555)

OS Update (Index 0x4554)

This object is used for OS updates. Do not access this object. Contact Jetter AG if you intend to update the operation system.

Electronic Data Sheet (Index 0x4555)

The Electronic Data Sheet (EDS) is provided for production and support purposes. It is user readable.

Index	Sub-Index	Default	Description	Attributes
0x4555	0	15	Number of entries	ro (read only)
	1	0	Status	ro
	2	0	Command	ro
	3		Page 0: Version	ro
	4		Page 0: Module code	ro
	5		Page 0: Module name (string)	ro
	6		Page 0: PCB revision	ro
	7		Page 0: PCB options	ro
	8		Page 1: Revision	ro
	9		Page 1: Module serial number (string)	ro
	10		Page 1: Production date: Day	ro
	11		Page 1: Production date: Month	ro
	12		Page 1: Production date: Year	ro
	13		Page 1: Test device number	ro
	14		Page 1: Test device revision	ro
	15		Page 0: Minimum OS version	ro

Object "System Parameters" (Index 0x4556)

System Parameters (Index 0x4556)

Use the object "System Parameters" to permanently change the parameters mentioned below. Any changes made to these parameters are stored in non-volatile memory and are therefore recovered when the JXM-IO-E02 is next powered up.

Note that some of these settings can also be set using other SDO objects. However, the System Parameters object is the only way to make these changes permanently.

Index	Sub-Index	Default	Description	Attributes
0x4556	0	51	Number of entries	ro (read only)
	1	1	Version (Read-only)	ro
	2	1	CAN Termination	rw (read & write)
	3	1	CAN Baud rate 0 = 125 kBaud, 1 = 250 kBaud, 2 = 500 kBaud, 3 = 1 MBaud	rw
	4	0x10	CANopen® Node ID	rw
	5	1.000	CANopen® Heartbeat time period	rw
	6	0x0A16	PWM 1: Predictor parameter	rw
	7	0x0302	PWM 1: Proportional parameter	rw
	8	0x0101	PWM 1: Integrator parameter	rw
	9	0x0A16	PWM 2: Predictor parameter	rw
	10	0x0302	PWM 2: Proportional parameter	rw
	11	0x0101	PWM 2: Integrator parameter	rw
	12	0x0A16	PWM 3: Predictor parameter	rw
	13	0x0302	PWM 3: Proportional parameter	rw
	14	0x0101	PWM 3: Integrator parameter	rw
	15	0	Analog input # 1: Mode selection	rw
	16	0	Analog input # 2: Mode selection	rw
	17	0	Analog input # 3: Mode selection	rw
	18	0	Analog input # 4: Mode selection	rw

Index	Sub-Index	Default	Description	Attributes
	19	25	Digital output # 1 (STANDARD): Current limit	rw
	20	25	Digital output # 2 (STANDARD): Current limit	rw
	21	25	Digital output # 3 (STANDARD): Current limit	rw
	22	25	Digital output # 4 (STANDARD): Current limit	rw
	23	25	Digital output # 5 (STANDARD): Current limit	rw
	24	25	Digital output # 6 (STANDARD): Current limit	rw
	25	25	Digital output # 7 (STANDARD): Current limit	rw
	26	25	Digital output # 8 (STANDARD): Current limit	rw
	27	25	Digital output # 9 (SAFETY): Current limit	rw
	28	25	Digital output # 10 (SAFETY): Current limit	rw
	29	25	Digital output # 11 (SAFETY): Current limit	rw
	30	25	Digital output # 12 (SAFETY): Current limit	rw
	31	25	Digital output # 13 (SAFETY): Current limit	rw
	32	25	Digital output # 14 (SAFETY): Current limit	rw
	33	50	Digital output # 15 (SAFETY): Current limit	rw
	34	50	Digital output # 16 (SAFETY): Current limit	rw
	35	0x01	PWM output # 1: Functional mode	rw
	36	0x01	PWM output # 2: Functional mode	rw
	37	0x01	PWM output # 3: Functional mode	rw
	38	1	Tri-state coding enable	rw
	39	100	Digital output: No-load threshold	rw
	40	100	PWM output: No-load threshold	rw
	41	0	Frequency input # 1: Functional mode	rw

Index	Sub-Index	Default	Description	Attributes
	42	0	Frequency input # 2: Functional mode	rw
	43	2.500	H-Bridge: Current limit	rw
	44	100	H-Bridge: No-load threshold	rw
	45	2.499	PWM output: Current limit	rw
	46	0	Digital inputs IN 1 through IN 5: Active-high / Active-low selection	rw
	47	0	Switch feed output: Initial output state	rw
	48	0	Digital outputs: No-load detection enable	rw
	49	0	PWM: No-load detection enable	rw
	50	0	H-Bridge: No-load detection enable	rw
	51	0	Event-triggered transmission of a PDO message	rw

Version / Reset

- When reading this parameter, the version number of the System Parameters is returned.
- When writing to this sub-index, a “Reset to Factory Defaults” option is enabled. To enable this function proceed as follows:
  - 1. Write 0x01042006.
  - 2. Write 0xC1EA5AFE.
  - 3. Wait a few seconds.
  - 4. Power cycle the JXM-IO-E02.

Delay

When writing to the System Parameters, **make sure** that a delay is implemented after the SDO command. The JXM-IO-E02 will reply to the SDO request to write to System Parameters and will then write the data to non-volatile memory. This process can take as much as 50 ms. Therefore, it is recommended that a delay of 100 ms be implemented before the next SDO or PDO access to the same JXM-IO-E02.

CAN Bus Termination

- This parameter selects whether the CAN termination resistors (120 Ω) inside the JCM-350-E03 must be activated (one each at both ends of the CAN bus).
- Valid options are:
  - 0x00: Neither resistor is enabled
  - 0x01: Resistor at the end of the CAN bus is enabled (default value)
  - 0x02: Resistor at the beginning of the CAN bus is enabled
  - 0x03: Both resistors enabled

CAN Baud Rate

- This parameter selects the CAN Baud rate to use.
- Valid options are:
  - 0: 125 kBaud
  - 1: 250 kBaud
  - 2: 500 kBaud
  - 3: 1 MBaud

CANopen® Node ID

- This parameter changes the node ID stored to the internal EEPROM.
- With coding via tri-state inputs 1 (pin 67) and 2 (68) enabled, this parameter sets the node ID of the JXM-IO-E02 if neither of the tri-state inputs is connected.
- If the device is configured NOT to use the tri-state inputs for selecting the node ID, then the value stored in this parameter will be the final node ID.
- The value is in the range of 0x01 through 0x76.
- The default value is 0x10.

CANopen® Heartbeat Time Period

- This is the time period, specified in milliseconds (ms), at which the JXM-IO-E02 will transmit a CANopen® heartbeat message. The own heartbeat status is sent as content of this message.
- The legal range for values is between 250 and 65,535 ms.
- Time periods less than 250 ms are allowed by CANopen® but do not make practical sense for the JXM-IO-E02 and are therefore not allowed.

PWM - Predictor, Proportional and Integrator Parameters

Please refer to the section on PWM object with index 0x2400 and sub-index 8 through 10.

Analog Input Mode Selection

- This parameter sets the initial functional mode for the analog inputs at power on.
- Please refer to the section on Analog Input object with index 0x2200 and sub-index 2.
- The default value is "0" selecting "Voltage Measurement" operation ranging from 0 to 5 V.

Digital Outputs # 1 - 8 (STANDARD) - Current Limit

- This parameter stores an output current limit value.
- The data is in units of 100 mA (i.e. 1 = 100 mA; 25 = 2.5 A).
- The current limit is in the range 1 ... 30 (100 mA ... 3 A).
- The default value is 25 (2.5 A).

Digital Outputs - Current Measurement

The current measurement of the JXM-IO-E02 is temperature dependent. At low temperatures the output current will be slightly larger than the limit above before being limited and at high temperatures the output current will be slightly smaller.

The following formula gives the relation between the specified current and the actual measured current:

$$I_{Ist} = I_{Soll} * \frac{K}{9{,}500}$$

Where K is taken from the following table:

Load current	K at T = -40 °C	K at T = 25 °C	K at T = 125 °C
0.5 A	12.000	12.000	12.000
2.5 A	10.000	9.700	9.300
5.0 A	10.000	9.700	9.300

The temperature specified in the above table is not the ambient temperature, but rather an internal device temperature.

This temperature will be at least 20°C higher than the ambient when the JXM-IO-E02 has been working for a few minutes.

Digital Outputs # 9 - 16 (SAFETY) - Current Limit

- This parameter stores an output current limit value.
- The data is in units of 100 mA (i.e. 1 = 100 mA; 25 = 2.5 A).
- The current limit is in the range 1 ... 55 (100 mA ... 5.5 A).
- The default value for the digital outputs 9 through 14 is 25 (2.5 A).
- The default value for the digital outputs 15 through 16 is 50 (5 A).

PWM Outputs - Mode Selection

- This parameter sets the initial functional mode for the PWM outputs at power on.
- Please refer to the section on PWM object with index 0x2400 and sub-index 2.
- The default value is 0x01 selecting current-controlled PWM operation.

Tri-State Coding Enable

- When this parameter is set to 1, the JXM-IO-E02 will use the tri-state inputs to calculate its node ID.
- Set this parameter to "0" in order to disable this function.
- The default value is 1.

Digital Outputs - No-Load Threshold

- This parameter stores an output current limit value affecting all 16 digital outputs.
- The threshold is specified in milliamps.
- The threshold is in the range 50 ... 250 (50 mA ... 250 mA).
- The default value is 100 (100 mA).

	▪ When a digital output's load current is low (less than 1 A), measuring this current becomes inaccurate (refer to section "Digital Outputs - Current Measurement"). If a threshold current of 100 mA is specified, it is likely that the actual measured current is 126 mA.
PWM Outputs - No-Load Threshold	▪ This parameter stores an output current limit value affecting all 3 PWM outputs. ▪ The threshold is specified in milliamps. ▪ The threshold is in the range 10 ... 1,000 (10 mA ... 1,000 mA). ▪ The default value is 100 (100 mA). ▪ No-load detection is available only in "static PWM duty-cycle output mode" (no current control).
Frequency Input Mode Selection	▪ This parameter sets the initial functional mode for the frequency inputs at power on. ▪ Please refer to the section on Frequency Input object with index 0x2600 and sub-index 2. ▪ The default value is 0 selecting frequency input operation.
H-Bridge - Current Limit Value	▪ This parameter stores an output current limit value for the H-bridge. ▪ The data is in units of 1 mA (i.e. 1 = 1 mA; 2,500 = 2.5 A). ▪ The current limit is in the range 250 ... 3,000 (250 mA ... 3.0 A). ▪ The default value is 2,500 (2.5 A).
H-Bridge - No-Load Threshold	▪ This parameter stores a no-load threshold for the H-bridge. ▪ The threshold is specified in milliamps. ▪ The threshold is in the range 100 ... 250 (100 mA ... 250 mA). ▪ The default value is 100 (100 mA).
PWM Output - Current Limit	▪ This parameter stores an output current limit value for the PWM outputs. ▪ The data is in units of 1 mA (i.e. 1 = 1 mA; 2,500 = 2.5 A). ▪ The current limit is in the range 500 ... 2,499 (500 mA ... 2.5 A). ▪ The default value is 2,499 (2.5 A). ▪ This parameter applies only when the outputs are used in "static PWM duty-cycle output mode" without current control.
Digital Inputs IN 1 through IN 5 - Initial Bias	▪ This parameter sets the power on default biasing for the digital inputs IN 1 through IN 5. ▪ Please refer to the section on Digital Input object with index 0x2100 and sub-index 2. ▪ The default value is "0", i.e. inputs IN 1 through IN 5 are "Active-Low".
Switch Feed Output - Initial State	▪ This parameter sets the initial output state for the two switch feed outputs at power on. ▪ Please refer to the section on Switch Feed Output object with index 0x2103 and sub-index 5. ▪ The default value is "0" selecting both outputs to be disabled.

Digital Output - No-Load Detection Enable	▪ Use this parameter to enable/disable no-load detection on all 16 digital output channels.
	▪ Each channel is represented by a single bit in the 16-bit word.
	  ▪ Bit 0: Channel 1 (OUT 1)
	  ▪ Bit 1: Channel 2 (OUT 2)
	  ▪ ...
	  ▪ Bit 14: Channel 15 (OUT 15)
	  ▪ Bit 15: Channel 16 (OUT 16)
	▪ To enable/disable no-load detection set the corresponding bit value:
	  ▪ 0: No-load detection is disabled
	  ▪ 1: No-load detection is enabled
	▪ The default value is "0" disabling no-load detection on all channels.

PWM Output - No-Load Enable	▪ Use this parameter to enable/disable no-load detection on all 3 PWM output channels.
	▪ Each channel is represented by a single bit in the 8-bit word.
	  ▪ Bit 0: PWM output # 1
	  ▪ Bit 1: PWM output # 2
	  ▪ Bit 2: PWM output # 3
	▪ To enable/disable no-load detection set the corresponding bit value:
	  ▪ 0: No-load detection is disabled
	  ▪ 1: No-load detection is enabled
	▪ The default value is "0" disabling no-load detection on all 3 PWM channels.

H-Bridge - No-Load Enable	▪ Use this parameter to enable/disable no-load detection on the H-bridge.
	▪ This parameter can have the following values:
	  ▪ 0: No-load detection is disabled
	  ▪ 1: No-load detection is enabled
	▪ The default value is "0" disabling no-load detection on the H-bridge.
	▪ This parameter applies only when the H-bridge is used in PWM mode.

Event-triggered transmission of a PDO message	▪ Use this parameter to enable sending of a PDO message when an event on one of the digital inputs occurs.
	▪ Each of the 21 inputs is allocated to a bit of the 32-bit word:
	  ▪ Bit 0: Digital input IN 1
	  ▪ Bit 1: Digital input IN 2
	  ▪ Bit 2: Digital input IN 3
	  ▪ Bit 3: Digital input IN 4
	  ▪ Bit 4: Digital input IN 5
	  ▪ Bit 16: Digital input IN 6
	  ▪ Bit 17: Digital input IN 7
	  ▪ ...
	  ▪ Bit 30: Digital input IN 20
	  ▪ Bit 31: Digital input IN 21

- To enable/disable event-triggered transmission of a PDO message set the corresponding bit value:
  - 0: Event-triggered transmission is disabled
  - 1: Event-triggered transmission is enabled
- The default value is "0" disabling event-triggered transmission of a PDO message for all inputs.

Detailed Software Version Object (Index 0x4559)

Detailed Software Version (Index 0x4559)

The structure of the object "Detailed Software Version" is shown in the following table. This read-only object supplies the same software version as object 0x100A, but in a 32-bit unsigned integer format which is compatible with the standard IP-type version numbers used at Jetter AG.

In addition, this object will also return the software version number for the two processors including their bootloader version numbers.

Index	Sub-Index	Default	Description	Attributes
0x4559	0	5	Number of entries	ro (read only)
	1	-	Software version	ro
	2	0	Master OS version	ro
	3	0	Master bootloader version	ro
	4	0	Slave OS version	ro
	5	0	Slave bootloader version	ro

User EEPROM Access Object (Index 0x5000)

User EEPROM Access (Index 0x5000)

The structure of the object "User EEPROM Access" is shown in the following table. This object grants the user read/write access to the EEPROM.

Index	Sub-Index	Default	Description	Attributes
0x5000	0	6	Number of entries	ro (read only)
	1	0	Byte offset inside memory space	rw (read & write)
	2	1.024	Size of memory (in bytes)	ro
	3	1	Auto increment	ro
	4	-	Byte R/W access	rw
	5	-	16-bit word R/W access	rw
	6	-	32-bit word R/W access	rw

Sub-Index 1

The function of sub-index 1 is described below:

- To use this object, enter the byte offset inside the memory space in sub-index 1.
- If the byte offset is less than zero, the CANopen® error “Value of parameter written too low” is returned.
- If the byte offset is larger than the value in sub-index 2 (default value: 1,024), the CANopen® error “Value of parameter written too high” is returned.
- Also, if the byte offset is set to one of the last byte values and an attempt is made to read or write a 16-bit or 32-bit word which would cause reading/writing outside the memory space, the “General error” message is returned.

  Unfortunately CANopen® doesn’t have an error code that accurately describes this condition.

  Example:

  If the byte offset is 1,022 and an attempt is made to read a 32-bit word, this would normally try to read beyond the last memory address of 1023. This is not allowed and the message "General error" is returned.

Sub-Index 2

The function of sub-index 2 is described below:

- The JXM-IO-E02 offers 1 kB of EEPROM memory space, but for some special devices the amount may differ.
- Reading sub-index 2 returns the available memory size in bytes.
- This sub-index is read-only.

Sub-Index 3

The function of sub-index 3 is described below:

- Use sub-index 3 to enable/disable the function "Auto Increment":
  - 0: Auto increment is disabled
  - 1: Auto increment is enabled
- Auto increment works as follows:
  - After either a read or write operation, the object will increment the offset in the memory space by the number of bytes that were transferred.
  - **Example:**
    After a byte read the byte offset is incremented by 1.
    After a 32-bit write the byte offset is incremented by 4.

Sub-Index 4

The function of sub-index 4 is described below:

- Read sub-index 4 to read a byte from the memory.
- Enter a value into sub-index 4 to write a byte in the memory.

Sub-Index 5

The function of sub-index 5 is described below:

- Read sub-index 5 to read a 16-bit word from the memory.
- Enter a value into sub-index 5 to write a 16-bit word in the memory.

Sub-Index 6

The function of sub-index 6 is described below:

- Read sub-index 6 to read a 32-bit word from the memory.
- Enter a value into sub-index 6 to write a 32-bit word in the memory.

Delay

When writing to the EEPROM, a delay **must** be implemented after the SDO command. The JXM-IO-E02 first writes to the EEPROM memory which takes a while before transmitting the SDO reply. This process can take at least 50 ms. Therefore, it is recommended that a delay of 100 ms be implemented before the next SDO or PDO access to the same JXM-IO-E02.

7.3 CANopen® PDO Specification

Introduction

This chapter describes the CANopen® PDO specification implemented on the JXM-IO-E02. PDO is short for Process Data Object. The PDO data allocation is fixed and cannot be changed by the application. The JXM-IO-E02 allows PDO access when the node has been set to operational state.

Contents

TX PDO Allocation on the JXM-IO-E02

PDO Assignment and Parameters

The tables below show the allocation of TX PDOs implemented on the JXM-IO-E02. CANopen® objects are linked with their corresponding PDOs.

From the controller point of view, the following data can be read back from the JXM-IO-E02 via the macro PDO1_RX (0x180 + node ID):

Byte Offset	Index / Sub-index	Size [byte]	Description
0	0x2101/04	2	Digital outputs read back
2	0x2100/04	2	Digital inputs
4	0x2100/04	1	Tri-state input

From the controller point of view, the following data can be read back from the JXM-IO-E02 via the macro PDO2_RX (0x280 + node ID):

Byte Offset	Index / Sub-index	Size [byte]	Description
0	0x2200/04	2	Analog input # 1
2	0x2201/04	2	Analog input # 2
4	0x2202/04	2	Analog input # 3
6	0x2203/04	2	Analog input # 4

From the controller point of view, the following data can be read back from the JXM-IO-E02 via the macro PDO3_RX (0x380 + node ID):

Byte Offset	Index / Sub-index	Size [byte]	Description
0	0x2600/04	2	Frequency input # 1
2	0x2601/04	2	Frequency input # 2

For PDO-3_RX, the frequency input fields change to sub-index 5 of the respective objects when the frequency input is used as a digital input. This allows monitoring of the digital input level by means of PDO.

Normally, the PDOs are transmitted asynchronously on request. However, PDO1_RX can also be enabled to be transmitted asynchronously on events. This is done using the “Event-based PDO TX enable” option in the System Parameters interface.

Additionally, from OS version 2.10.0.01 PDO3_RX is also transmitted asynchronously on events. The event that triggers this is the completion of a frequency measurement. This function cannot be disabled and is only available for frequency measurement at this time.

RX PDO Allocation on the JXM-IO-E02

PDO Assignment and Parameters

The tables below show the allocation of RX PDOs implemented on the JXM-IO-E02. CANopen® objects are linked with their corresponding PDOs. Therefore, writing to that PDO will be the same as writing to that SDO index and sub-index.

From the controller point of view, the following data on the JXM-IO-E02 can be accessed via the macro PDO1_TX (0x200 + node ID):

Byte Offset	Index / Sub-index	Size [byte]	Description
0	0x2101/05	2	Digital outputs
2	0x2103/05	1	Switch feed outputs

From the controller point of view, the following data on the JXM-IO-E02 can be accessed via the macro PDO2_TX (0x300 + node ID):

Byte Offset	Index / Sub-index	Size [byte]	Description
0	0x2500/05	2	H-bridge output state
2	0x2400/04	2	PWM-1 current
4	0x2401/04	2	PWM-2 current
6	0x2402/04	2	PWM-3 current

Please note that specification of current values is allowed only in mode "Current-Controlled PWM Output".

If the PWM output is set to static duty-cycle mode, this parameter will actually change to sub-index 5 to allow writing to the duty-cycle register. The PDO interface can therefore be used to also select the duty cycle.

From the controller point of view, the following data on the JXM-IO-E02 can be accessed via the macro PDO3_TX (0x400 + node ID):

Byte Offset	Index / Sub-index	Size [byte]	Description
0	0x2300/04	2	Analog output - voltage
2	0x2300/05	2	Analog output - current

8 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 JCM-350-E03 and other ECUs in the vehicle. As a rule, engine data e.g. rpm, speed or coolant temperature are read and displayed.

Documentation

The key SAE J1939 specifications are:

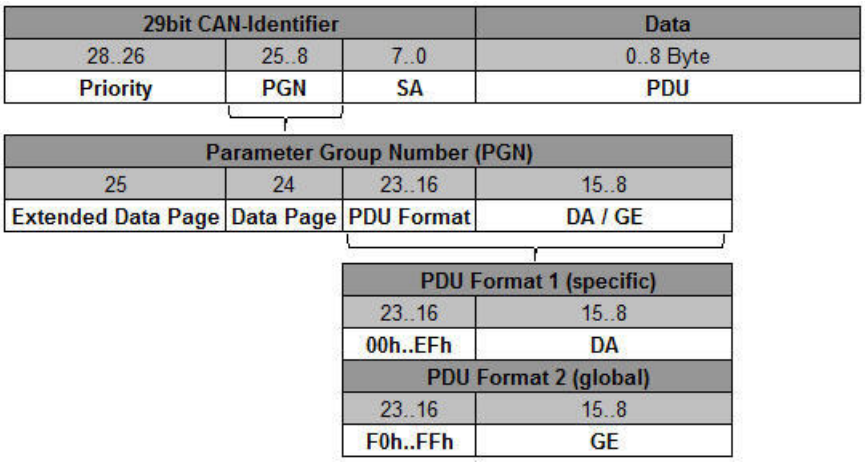
- J1939-11 - Information on the physical layer
- J1939-21 - Information on the data link layer
- J1939-71 - Information on the application layer vehicles
- J1939-73 - Information on the application layer range analysis
- J1939-81 - Network management

Contents

Content of a J1939 Message

Content of a J1939 Message

The following diagram shows the content of a J1939 message:



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

Meaning of the Parameter Group Number (PGN)

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

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

PGN 65262 Engine Temperature 1 - ET1

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

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

STX Function SAEJ1939Init

Introduction

Calling up the SAEJ1939Init () function initializes one of the CAN busses (not CAN 0 as this is reserved for CANopen®) available for the J1939 protocol. From then on, the JCM-350-E03 has the SA (Source Address) assigned by the function parameter mySA. It thus has its own device address on the bus.

Function Declaration

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

Function Parameters

The function SAEJ1939Init () has the following parameters.

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

Return Value

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

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

Parameter CANNo

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

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

Using this Function

Initializing the CAN-Bus 1. The JCM-350-E03 has Node-SA 20 (0x14).

The JCM-350-E03 can now send messages with the set SA (and only these messages).

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

Address Claiming

Address Claiming has not been implemented.

STX Function SAEJ1939SetSA

Introduction

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

Function Declaration

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

Function Parameters

The function SAEJ1939SetSA () has the following parameters.

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

Using this Function

The SA is changed during runtime.

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

Important Note

Messages are immediately sent/received with the new SA.

STX Function SAEJ1939GetSA

Introduction

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

Function Declaration

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

Function Parameters

The function SAEJ1939GetSA () has the following parameters.

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

Using this Function

This function returns the currently set SA.

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

STX Function SAEJ1939AddRx

Introduction

Calling up the function SAEJ1939AddRx () prompts the JCM-350-E03 to receive a specific message. This message is sent from another bus node. The address of this bus node is transferred to this function as a bySA parameter. If the message is not sent, the value received last remains valid. Cyclical reading continues until the function SAEJ1939Init () is called up again.

Function Declaration

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

Function Parameters

The function SAEJ1939AddRx () has the following parameters.

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

Parameter DataType

Data types can include the following.

Byte types	Bit types	SAEJ1939
1	-	SAEJ1939_UNSIGNED8 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;
```

Using this Function

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

JetSym STX Program

The device JCM-350-E03 with the own SA of 20 wants to receive and display the current fuel temperature. The parameters InhibitTime and EventTime are not explicitly specified when calling up the function. In this case, the default values are used. The controller that measures the fuel temperature has the SA of 0. In practice, the address of the controller can be found in the engine manufacturer's documentation.

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

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;

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

Task main autorun

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

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

End_Task;
```

Engine Manufacturer's Manual

For information on the data (priority, PGN, SA and data byte structure) refer to the manual provided by the engine manufacturer.

STX Function SAEJ1939AddTx

Introduction

Calling up the function SAEJ1939AddTx () prompts the device JCM-350-E03 to cyclically send a specific message via the bus.

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

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

Function Declaration

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

Function Parameters

The function SAEJ1939AddTx () has the following parameters.

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

Parameter DataType

Data types can include the following.

Byte types	Bit types	SAEJ1939
1	-	SAEJ1939_UNSIGNED8 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;
```

Using this Function

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

JetSym STX Program

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

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;

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

Task main autorun

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

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

End_Task;
```

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

STX Function SAEJ1939RequestPGN

Introduction

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

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

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

This function must be constantly recalled in cycles.

Function Declaration

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

Function Parameters

The function SAEJ1939RequestPGN () has the following parameters.

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
byDA	Destination Address 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

Parameter CANNo

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

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

Parameter DataType

Data types can include the following.

Byte types	Bit types	SAEJ1939
1	-	SAEJ1939_UNSIGNED8 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

Using this Function

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

JetSym STX Program

JCM-350-E03 with own SA of 20 wants to request the PGN 65253 "Engine Hours" from an engine control unit with the SA 0. The SPN 247 "Engine Total Hours of Operation" should be read from this PGN. It is therefore necessary to register receipt of the SPN 247 by calling up the function SAEJ1939AddRx ().

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

```
#Include "SAEJ1939.stxp"

Var
    bySAEJ1939Channel  : Byte;
    own_Source_Address : Byte;

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

Task main autorun

// Initializing CAN 1
```

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

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

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

End_Task;


Task t_RequestPGN_5000 cycle 5000

Var
    Return_value : Int;
End_Var;

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

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

End_Task;
```

STX Function SAEJ1939GetDM1

Introduction

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

Function Declaration

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

Function Parameters

The function SAEJ1939GetDM1 () has the following parameters.

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
bySA	Source Address of message sender	0 ... 253 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

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

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)

Using this Function

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

JetSym STX Program

By calling up the function SAEJ1939GetDM1 (), the JCM-350-E03 requests the current diagnostics error code (PGN 65226).

```
#Include "SAEJ1939.stxp"

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

Task main autorun

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

TimerStart (MyTimer, T#2s);

Loop

When (TimerEnd (MyTimer)) Continue;

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

TimerStart (MyTimer, T#2s);

End_Loop;

End_Task;
```

STX Function SAEJ1939GetDM2

Introduction

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

Function Declaration

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

Function Parameters

The function SAEJ1939GetDM2 () has the following parameters.

Parameter	Description	Value
CANNo	CAN channel number	1 ... CANMAX
bySA	Source Address of message sender	0 ... 253 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

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

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)

Using this Function

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

JetSym STX Program

By calling up the function SAEJ1939GetDM2 (), the JCM-350-E03 requests the current diagnostics error code (PGN 65227).

```
#Include "SAEJ1939.stxp"

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

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

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

End_Task;


Task t_RequestPGN_5000 cycle 5000

Var
    Return_value  : Int;
End_Var;

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

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

End_Task;
```

STX Function SAEJ1939SetSPNConversion

Introduction

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

Function Declaration

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

Function Parameters

The function SAEJ1939SetSPNConversion () has the following parameters.

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

Using this Function

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

STX Function SAEJ1939GetSPNConversion

Introduction

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

Function Declaration

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

Function Parameters

The function SAEJ1939GetSPNConversion () has the following parameters.

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

Return Value

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

Return Value	
0	ok
-1	Error when checking parameters

Parameter CANNo

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

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

Using this Function

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

9 Programming

Purpose of this Chapter

This chapter is for supporting you in programming the JCM-350-E03 in the following fields of activity:

- Programming Additional Functions

Prerequisites

To be able to program the JCM-350-E03 the following prerequisites must be fulfilled:

- A USB CAN adaptor between PC and JCM-350-E03 and the driver software for this adaptor have been installed.
  The device is now connected to a PC via CAN bus.
- The programming tool JetSym 4.3 or higher is installed on the PC.

Contents

Abbreviations, Module Register Properties and Formats

Abbreviations

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

Abbreviation	Meaning
R 100	Register 100
MR 150	Module register 150

Module Register Properties

Each module register is characterized by certain properties. For many module registers most properties are identical. For example, their value after reset is 0. In the following description, module register properties are mentioned only if a property deviates from the following default properties.

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

Number Formats

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

Notation	Number Format
100	Decimal
0x100	Hexadecimal
0b100	Binary

JetSym Sample Programs

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

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

9.1 Memory Overview

Introduction

The JCM-350-E03 features several types of program and data memories. There is volatile memory that requires power to maintain the stored information, and non-volatile memory which does not require power to maintain the stored information. This memory is located directly on the CPU. This chapter gives an overview of the available memory.

Contents

File System Memory

Introduction	The file system memory is for storing data and program files.
Properties	▪ Internal flash disk ▪ Non-volatile ▪ Slow access: milliseconds up to seconds ▪ Limited number of write/delete cycles: approx. 1 million ▪ Internal flash disk size: 4 MBytes
Memory Access	▪ By operating system ▪ By JetSym ▪ By means of file commands from within the application program

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 JCM-350-E03 is powered up.

Features

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

Memory Access

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

Related Topics

- **Updating the Operating System** on page 209

Application Program Memory

Introduction	By default, the application program is uploaded from JetSym to the controller and is stored to it.
Properties	▪ Stored as file within the file system ▪ Default directory: "/app" ▪ Files may also be stored to other directories (or on SD card) ▪ Size: 256 KByte max.
Memory Access	▪ By operating system ▪ By JetSym ▪ By means of file commands from within the application program
Related Topics	▪ **Application Program** on page 211

Memory for Non-Volatile Application Program Registers

Introduction

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

Properties

- Global variables assigned to permanent addresses (%VL)
- Register variables always occupy 4 bytes
- Register variables are not initialized by the operating system
- Number of register variables: 6.000
- Register numbers: 1,000,000 through 1,005,999

Memory Access

- By JetSym
- From within the application program

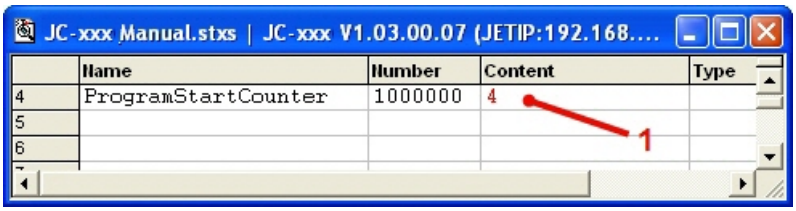
JetSym STX Program

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

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

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

Setup Pane

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



Number	Content	Description
1	Present content of the register variable	The content of the register variable is incremented by 1 every time the program is launched.

Memory for Non-Volatile Application Program Variables

Introduction

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

Properties

- Global variables assigned to permanent registers (%RL)
- Variables are stored in a compact way
- Size: 120,000 bytes
- Register numbers: 1,000,000 through 1,005,999

Memory Access

- By JetSym
- From within the application program

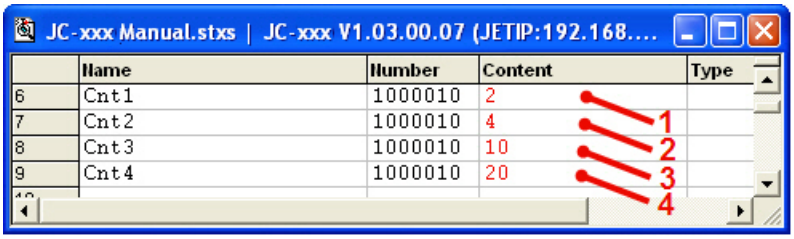
JetSym STX Program

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

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

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

Setup Pane

The JetSym setup pane displays the content of the variable. As the type of the 4 counters is byte, this will result in counter overflow after a relatively short time:



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

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

Special Registers

Introduction	Special registers are used to control OS functions and to retrieve status information.
Properties	▪ Global variables assigned to permanent addresses (%VL) ▪ When the operating system is launched, special registers are initialized using default values. ▪ Register numbers: 100,000 through 999,999
Memory Access	▪ By JetSym ▪ From within the application program

Flags

Introduction

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

Properties of User Flags

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

Properties of Overlaid User Flags

- Global variables assigned to permanent addresses (%MX)
- Non-volatile
- Overlaid by registers 1000000 through 1000055
- Quantity: 1.792
- Flag numbers: 256 through 2047

Properties of Special Flags

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

Memory Access

- By JetSym
- From within the application program

JetSym STX Program

In the following program the variable Counter1 is incremented every 500 ms if flag 1 is set.

```
Var
    Flag1:     Bool At %MX 1;
    Counter1:    Int At %VL 1000000;
End_Var;

Task Flag Autorun
    Flag1:= False;
    Loop
        When Flag1 Continue;
        Inc(Counter1);
        Delay(T#500ms);
    End_Loop;
End_Task;
```

9.2 Runtime Registers

Introduction

The JCM-350-E03 provides several registers which are incremented by the operating system at regular intervals.

Application

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

Contents

Description of Runtime Registers

Overview of Registers

The following registers are used in this manual:

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

R 201000

Application time base in milliseconds

Every millisecond this register is incremented by 1.

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

R 201001

Application time base in seconds

Every second this register is incremented by 1.

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

R 201002

Application time base in application time base units

Every [201003] * 10 milliseconds this register is incremented by 1. Using the reset value in register 201003 of 10, this register is incremented every 100 milliseconds.

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

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

This register contains the multiplier for runtime register R 201002.

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

R 201004 **System time base in milliseconds**

Every millisecond this register is incremented by 1.

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

Sample Program - Runtime Registers

Task

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

Solution

Before storing the values register 201000 is set to 0. Once the values have been stored, from this register can be seen how much time it took to store the values [in milliseconds].

JetSym STX Program

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

    MilliSec:     Int At %VL 201000;
End_Var;

Task WriteToFile Autorun
    Loop
        // clear start flag
        WriteIt := False;
        // wait until start flag set by user
        When WriteIt Continue;
        // open file in write mode
        If FileOpen(File1, '/Test.dat', fWrite) Then
            // restart timer register
            MilliSec := 0;
            // write array data to file
            FileWrite(File1, DataArray,
                      SizeOf(DataArray));
            // capture time
            WriteTime := MilliSec;
            FileClose(File1);
            // show measured time
            Trace(StrFormat('Time : %d [ms]$n',
                            WriteTime));
        Else
            // show error message
            Trace('Unable to open file!$n');
        End_If;
    End_Loop;
End_Task;
```

9.3 Addressing the JXM-IO-E02 via CANopen®

Purpose of this Chapter

This chapter describes how to address the JXM-IO-E02 by means of JetSym STX.

JCM-350-E03 - Configuration

The JCM-350-E03 consists of the controller JCM-350 and the I/O module JXM-IO-E02 which are internally connected via CAN bus. The CAN bus is brought out to allow communication with other CANopen® nodes. The default node ID of the JXM-IO-E02 is 16, the default node ID of the JCM-350 is 127. This way, both components within the JCM-350-E03 can be addresses separately.

Contents

9.4 Digital Outputs

Introduction

This chapter describes how to address digital outputs by using PDO and SDO.

Contents

Reading In the Number of Available Digital Outputs Per SDO

Task

Read in the number of available digital outputs on the JXM-IO-E02.

Solution

SDO is used to access the object "Universal I/O" in the object dicitionary and to obtain its value.

Prerequisites

Initial commissioning of JCM-350-E03 has been completed. This means:

- Installation of the device is completed
- The device is connected via USB CAN adaptor to the PC.
- In JetSym an active connection to the JCM-350-E03 exists.

How it Works

The program accesses the object "Universal I/O" with index **0x2101** and sub-index **6** on the JXM-IO-E02 by means of the CANopen® STX-API function **CanOpenUploadSDO()** and reads out its value. This value is stored to the variable **Data_Outputs**. The content of this variable can be displayed in the JetSym setup pane.

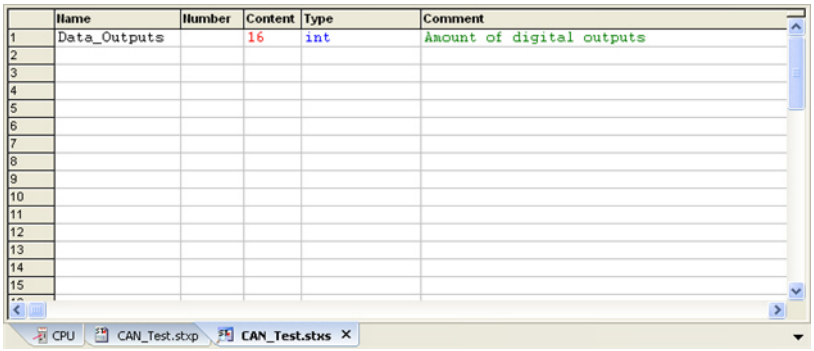
JetSym STX Program

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;
End_Const;

Var
    SW_Version: String;
    busy: int;
    Data_Outputs: Long;
    Objectindex: Long;
    Subindex: Byte;
End_Var;

Task Main Autorun
// Software version of the controller
SW_Version := 'v4.3.0';

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

// Obtaining the number of available digital outputs per SDO

// Digital Output Object
Objectindex := 0x2101;
Subindex := 6;
```

```
CanOpenUploadSDO(CAN_CONTROLLER_0, NodeID_Node_1, Objectindex,
Subindex, CANOPEN_DWORD, sizeof(Data_Outputs), Data_Outputs,
busy);

End_Task;
```

Setup Pane

If the variable "Data_Outputs" has been selected in the JetSym setup pane, the value in the column "Content" shows that 16 outputs are available:



Setting Digital Outputs Per PDO

Task

Set a digital output on the JXM-IO-E02.

Solution

By means of a PDO the message for setting the digital output is sent to the CAN bus.

Prerequisites

Initial commissioning of JCM-350-E03 has been completed. This means:

- Installation of the device is completed
- The device is connected via USB CAN adaptor to the PC.
- In JetSym an active connection to the JCM-350-E03 exists.

How it Works

First, the controller JCM-350 is initialized. Then, it sends the data required for setting the digital output to the CAN bus by using the function **CanOpenAddPDOTx()**. Please note that the I/O module JXM-IO-E02 receives process data on the CAN bus only on request. This is achieved by the parameter "CANopen_ASYNCPDORTRONLY". Following this, the JXM-IO-E02 is set into the state "operational". Now, the JXM-IO-E02 receives the data in question and sets the digital outputs as requested.

JetSym STX Program

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;

  Event_Time = 100;
  Inhibit_Time = 20;
End_Const;


Var
// Variable for setting outputs
    Data_Outputs: Word;
    SW_Version: String;
End_Var;


Task Main Autorun
// Setting output 1
Data_Outputs:= 1;

// Software version of the controller
SW_Version := 'v4.3.0';
```

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

// Sending process data to the bus
CanOpenAddPDOTx(CAN_CONTROLLER_0,
CANOPEN_PDO1_TX(NodeID_Node_1), 0, CANOPEN_WORD,
sizeof(Data_Outputs), Data_Outputs, Event_Time, Inhibit_Time,
CANOPEN_ASYNCPDORTRONLY);

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

End_Task;
```

9.5 Digital Inputs

Introduction

This chapter describes how to read and configure digital inputs by using PDO and SDO.

Contents

Digital Inputs SDO

Task

Set a digital input on the JXM-IO-E02 to "Active-High" by means of the internal pulldown resistors.

Solution

SDO is used to access the object "Digital Inputs" and to set input 1 to "Active-High".

Prerequisites

Initial commissioning of JCM-350-E03 has been completed. This means:

- Installation of the device is completed
- The device is connected via USB CAN adaptor to the PC.
- In JetSym an active connection to the JCM-350-E03 exists.

How it Works

The program accesses the object "Digital Inputs" with index **0x2100** and sub-index **2** by means of the CANopen® STX-API function **CanOpenDownloadSDO()**. Then, input 1 is set to "Active-High" (bit 0 = 1).

JetSym STX Program

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;
End_Const;

Var
    busy: Int;
    SW_Version: String;
    Inputs_Mode: Long;
    Objectindex: Word;
    Subindex: Byte;
End_Var;

Task Main Autorun

// Software version of the controller
SW_Version := 'v4.3.0';

// First input Active-High
Inputs_Mode:= 1;

// Initializing CAN 0

CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);
Objectindex := 0x2100;
Subindex := 2;
```

```
// Set input 1 to Active-High
CanOpenDownloadSDO(CAN_CONTROLLER_0, NodeID_Node_1, Objectindex,
Subindex, CANOPEN_DWORD, sizeof(Inputs_Mode), Inputs_Mode, busy);

End_Task;
```

Digital Inputs PDO

Task

Read in the digital inputs on the JXM-IO-E02.

Solution

PDO is used to enter the process data to be received.

Prerequisites

Initial commissioning of JCM-350-E03 has been completed. This means:

- Installation of the device is completed
- The device is connected via USB CAN adaptor to the PC.
- In JetSym an active connection to the JCM-350-E03 exists.

How it Works

First, the controller JCM-350 is initialized. Then, it enters the process data required for reading the digital inputs by using the function **CanOpenAddPDORx()**. Please note that the I/O module JXM-IO-E02 sends process data only on request. This is achieved by the parameter "CANOPEN_ASYNCPDORTRONLY". Following this, the JXM-IO-E02 is set into the state "operational". Now, the JXM-IO-E02 sends the requested data.

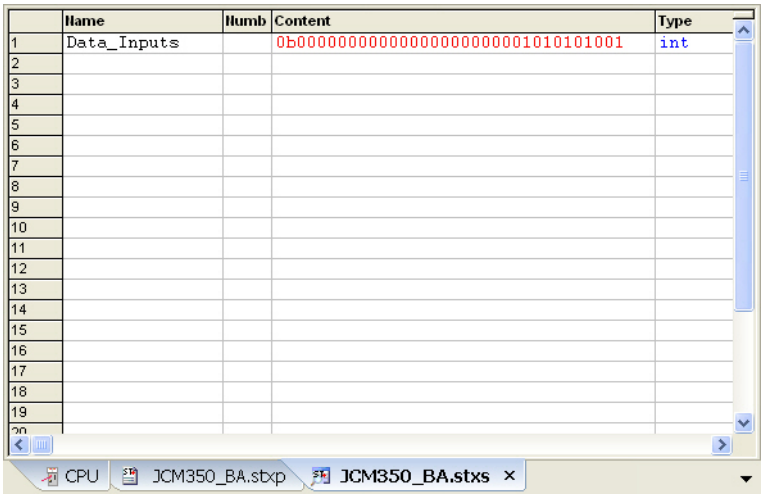
JetSym STX Program

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;

  Event_Time = 100;
  Inhibit_Time = 20;
End_Const;

Var
// State of the digital inputs
    Data_Inputs: Word;
    SW_Version: String;
End_Var;


Task Main Autorun

// Software version of the controller
SW_Version := 'v4.3.0';

// Initializing CAN 0

CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// Entering process data to be sent
```

```
CanOpenAddPDORx(CAN_CONTROLLER_0,
CANOPEN_PDO1_RX(NodeID_Node_1), 2, CANOPEN_WORD,
sizeof(Data_Inputs), Data_Inputs, Event_Time, Inhibit_Time,
CANOPEN_ASYNCPDORTRONLY);

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

End_Task;
```

Setup Pane

If the variable "Data_Inputs" has been selected in the JetSym setup pane, its content is displayed as shown below. The value of this variable depends on the state of the digital inputs and on their configuration (active-high or active-low). In the given case, input 1 has been configured as active-high and the other four inputs as active-low. These inputs are not connected.



9.6 H-Bridge

Introduction

This chapter describes how the H-bridge is configured by means of SDO and how a PWM signal with a static duty-cycle is output by using PDO.

Contents

Configuring the H-Bridge by Using SDO and PDO

Task

A PWM signal with a static duty-cycle is to be output at the H-bridge outputs.

Solution

First, SDO is used to access the object "H-bridge" and to select the operating mode. Then, a PWM signal with a static duty-cycle is output by means of PDO.

Prerequisites

Initial commissioning of JCM-350-E03 has been completed. This means:

- Installation of the device is completed
- The device is connected via USB CAN adaptor to the PC.
- In JetSym an active connection to the JCM-350-E03 exists.

How it Works

The program has two main functions:

- The program first accesses the object "H-Bridge" with index **0x2500** and sub-index **2** by means of the CANopen® STX-API function **CanOpenDownloadSDO()**. The value **0x02** is entered into sub-Index 2 to select the output mode. In this mode, the output connected to pin 69 is a PWM-controlled active-high output, whereas the output connected to pin 70 is always low.
- Then, the CANopen® STX-API function **CanOpenAddPDOTx()** is used to set the PWM duty cycle of the H-bridge to 150. Please note that the I/O module JXM-IO-E02 receives process data on the CAN bus only on request. This is achieved by the parameter "CANopen_ASYNCPDORTRONLY". Following this, the JXM-IO-E02 is set into the state "operational". Now, the JXM-IO-E02 receives the data in question and sets the PWM duty cycle as requested.

JetSym STX Program

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;

  Event_Time = 100;
  Inhibit_Time = 20;
End_Const;
Var
    busy: Int;
    SW_Version: String;
    HBridge_Mode: Long;
    PWM_Value: Long;
    Objectindex: Word;
    Subindex: Byte;
End_Var;
```

```
Task Main Autorun

// Software version of the controller
SW_Version := 'v4.3.0';

// Mode
HBridge_Mode:= 0x02;

// Init PWM
PWM_Value:= 150;

// Initializing CAN 0

CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// SDO
Objectindex := 0x2500;

Subindex := 2;
// Mode
CanOpenDownloadSDO(CAN_CONTROLLER_0, NodeID_Node_1, Objectindex,
Subindex, CANOPEN_DWORD, sizeof(HBridge_Mode), HBridge_Mode,
busy);

// PWM Value
CanOpenAddPDOTx(CAN_CONTROLLER_0,
CANOPEN_PDO2_TX(NodeID_Node_1), 0, CANOPEN_WORD,
sizeof(PWM_Value), PWM_Value, Event_Time, Inhibit_Time,
CANOPEN_ASYNCPDORTRONLY);

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

End_Task;
```

9.7 PWM Outputs

Introduction

This chapter describes how the PWM output 1 is configured by means of SDO and how a PWM signal with a static duty-cycle is output by using PDO.

Contents

Configuring the PWM Output 1 by Using SDO and PDO

Task

A PWM signal with a static duty-cycle is to be output at PWM output 1.

Solution

First, SDO is used to access the object "PWM" and to select the operating mode. Then, a PWM signal with a static duty-cycle is output at PWM output 1 by means of PDO.

Prerequisites

Initial commissioning of JCM-350-E03 has been completed. This means:

- Installation of the device is completed
- The device is connected via USB CAN adaptor to the PC.
- In JetSym an active connection to the JCM-350-E03 exists.

How it Works

The program has two main functions:

- The program first accesses the object "PWM" with index **0x2400** and sub-index **2** by means of the CANopen® STX-API function **CanOpenDownloadSDO()**. The value **0x02** is entered into sub-Index 2 to select the output mode. In this mode, a PWM signal with static duty-cycle is output.
- Then, the CANopen® STX-API function **CanOpenAddPDOTx()** is used to set the PWM duty-cycle of PWM output 1. Please note that the I/O module JXM-IO-E02 receives process data on the CAN bus only on request. This is achieved by the parameter "CANopen_ASYNCPDORTRONLY". Following this, the JXM-IO-E02 is set into the state "operational". Now, the JXM-IO-E02 receives the data in question and sets the PWM duty cycle at PWM output 1 as requested.

JetSym STX Program

```
Const
    CAN_CONTROLLER_0 = 0;

    //Node ID of the controller
    NodeID_Node_0 = 0x7F;

    //Node ID of the I/O module
    NodeID_Node_1 = 0x10;

  Event_Time = 100;
  Inhibit_Time = 20;
End_Const;

Var
    busy: Int;
    SW_Version: String;
    PWM_Mode: Long;
    PWM_Value: Long;
    Objectindex: Word;
    Subindex: Byte;
End_Var;
```

```
Task Main Autorun

// Software version of the controller
SW_Version := 'v4.3.0';

// Mode
PWM_Mode:= 0x02;

// Init PWM
PWM_Value:= 150;

// Initializing CAN 0

CanOpenInit(CAN_CONTROLLER_0, NodeID_Node_0, SW_Version);

// SDO
Objectindex := 0x2400;

Subindex := 2;
// Mode
CanOpenDownloadSDO(CAN_CONTROLLER_0, NodeID_Node_1, Objectindex,
Subindex, CANOPEN_DWORD, sizeof(PWM_Mode), PWM_Mode, busy);

// PWM Value
CanOpenAddPDOTx(CAN_CONTROLLER_0,
CANOPEN_PDO2_TX(NodeID_Node_1), 2, CANOPEN_WORD,
sizeof(PWM_Value), PWM_Value, Event_Time, Inhibit_Time,
CANOPEN_ASYNCPDORTRONLY);

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

End_Task;
```

10 Protection and Diagnostic Features - JXM-IO-E02

Purpose of this Chapter

This chapter describes the available protection and diagnostic features implemented on the JXM-IO-E02. The following features are currently supported:

- Detecting faults in the application program or visualization.
- Identifying the root cause of a fault.
- Troubleshooting an error that caused a fault message.

Prerequisites

To be able to troubleshoot a fault on the JXM-IO-E02 module the following prerequisites must be fulfilled:

- The JXM-IO-E02 module is connected to a controller or is integrated into the controller JCM-350-E03.
- The controller is connected to a PC.
- The programming tool JetSym is installed on the PC.
- The minimum requirements regarding modules, controllers and software are fulfilled.

Background

When a fault is detected, the module JXM-IO-E02 will disable the function that caused the fault. It will transmit a CANopen® Emergency Object to inform the controller of the problem. The fault is also recorded in a history list of error events. These error events are compliant to the CANopen® "Pre-defined Error Field".

The external controller can immediately reactivate the function, but as long as the fault remains, the module JXM-IO-E02 will again disable the function and retransmit the error notification.

Contents

Standard Feed Power Input (STANDARD FEED)

Detecting the Error

The input current on STANDARD FEED is monitored by software. The software will issue an over-current error notification if the current exceeds 30 A.

The software implements a function allowing temporary over-current. This is useful in situations where high peak currents are required.

Root Cause of Error

This error may be caused by the following root causes:

- The maximum current of 30 A has been exceeded.
- The time limit for overcurrent has been exceeded.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Over-current	0x2323	2

Fixing the Root Cause

The controller must respond to the error message and disable the outputs on the module JXM-IO-E02.

Safety Feed Power Input (SAFETY FEED)

Detecting the Error

The input current on SAFETY FEED is monitored by software. The software will issue an over-current error notification if the current exceeds 30 A.

The solid state switch used to disable the safety outputs (safety switch) also implements a hardware limit. The safety switch will switch off automatically if the switch temperature rises too high and the set actual current is exceeded. The actual current that will cause the safety switch to disconnect is dependent on the ambient temperature.

The software implements a function allowing temporary over-current. This is useful in situations where high peak currents are required.

Root Cause of Error

This error may be caused by the following root causes:

- The maximum current of 30 A has been exceeded.
- The time limit for over-current has been exceeded.
- If the safety switch temperature rises too high and the actual current is at least 30 A.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Over-current	0x2322	2
Safety Switch Failure	0x5001	8

If the switch fails in the ON state, the JCM-350-E03 will additionally issue the "Safety Switch Failure" notification.

Digital Outputs 1 ... 8 (Standard Outputs)

Detecting the Error

A threshold can be programmed for both over-current and cable breakage (no load) via the System Parameters interface. Over-current limit can be set to between 100 mA and 2.5 A per channel. The no-load threshold can be set between 50mA and 250mA.

Note that this no-load threshold is shared for all digital outputs. No-load detection can be enabled or disabled for individual output channels. A no-load fault can only be detected when a channel is switched on (enabled).

The software implements a function allowing temporary over-current. This is useful in situations where high peak currents are required.

Root Cause of Error

This error may be caused by the following root causes:

- The programmed limit for over-current has been exceeded.
- The load current has exceeded 10 A and the over-current situation has exceeded 180 ms.
- The programmed limit for no-load has been exceeded.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Short to GND	0x9000 - 0x9007	1
Over-current	0x2300 - 0x2307	2
No load (cable breakage)	0x23A0 - 0x23A7	2

Digital Outputs 9 ... 16 (Safety Outputs)

Detecting the Error

A threshold can be programmed for both over-current and cable breakage (no load) via the System Parameters interface. Over-current limit can be set to between 100 mA and 5 A per channel. The no-load threshold can be set between 50mA and 250mA.

Note that this no-load threshold is shared for all digital outputs. No-load detection can be enabled or disabled for individual output channels. A no-load fault can only be detected when a channel is switched on (enabled).

The software implements a function allowing temporary over-current. This is useful in situations where high peak currents are required.

Root Cause of Error

This error may be caused by the following root causes:

- The programmed limit for over-current has been exceeded.
- The load current has exceeded 10 A and the over-current situation has exceeded 180 ms.
- The programmed limit for no-load has been exceeded.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Short to GND	0x9010 - 0x9017	1
Over-current	0x2310 - 0x2317	2
No load (cable breakage)	0x23B0 - 0x23B7	2

Analog Output

Detecting the Error

The analog output will detect short circuit to ground faults.

No other faults are tested for on the Analog Output because the output is both current and voltage controlled. The controller algorithm is responsible to keep the output voltage and current within specified limits.

Root Cause of Error

This error may be caused by the following root cause:

- When a short to ground is detected, the output is disabled and the fault notification is sent out.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error. The analog output will remain disabled until the module is instructed to set the analog output to a normal mode again or until a power cycle has occurred.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Short to GND	0x9020	1

PWM Outputs 1 ... 3

Detecting the Error

The PWM outputs can be used in one of two modes:

- Current-controlled output
- PWM output with static duty cycle.

When these outputs are used as current-controlled outputs, the module JXM-IO-E02 will detect short circuit to ground and no load faults. No load is defined by a current threshold which is user selectable through the system parameters interface.

When a PWM output is set as a static duty-cycle output, the module JXM-IO-E02 will additionally detect over-current faults. These faults are also defined by a user selectable current threshold.

Root Cause of Error

This error may be caused by the following root causes:

- The programmed limit for over-current has been exceeded.
- The programmed limit for no-load has been exceeded.
- A short-circuit to ground has occurred.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Short to GND	0x90D0 - 0x90D2	1
Over-current	0x23D0 - 0x23D2	2
No load (cable breakage)	0x23C0 - 0x23C2	2

H-Bridge

Detecting the Error

Full protection is only available when the H-Bridge is used in the H-Bridge PWM output modes.

If the H-Bridge is used as two independent digital outputs, only short-circuit to ground fault detection is possible. This detection is unable to detect which of the two outputs caused the problem and will disable both outputs if a problem has been detected.

Over-current and no load faults have user selectable thresholds. These can be set through the System Parameters interface.

Root Cause of Error

This error may be caused by the following root causes:

- The programmed limit for over-current has been exceeded.
- The programmed limit for no-load has been exceeded.
- A short-circuit to ground has occurred.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Short to GND	0x9021	1
Over-current	0x2321	2
No load (cable breakage)	0x2331	2

Switch Feed Outputs 1 ... 2

Detecting the Error

Although the fault condition is "over-temperature", this fault encompasses both short-circuit to ground and over-current faults. If either fault occurs, the module JXM-IO-E02 will issue an over-temperature error for the output.

Root Cause of Error

This error may be caused by the following root causes:

- The programmed limit for over-current has been exceeded.
- A short-circuit to ground has occurred.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Over-temperature	0x4231 - 0x4232	4

Safety Switch (Relay)

Root Cause of Error

This error may be caused by the following root cause:

- The safety switch (relay) fails to disable the safety outputs.

Response of the Module to this Error

The module will set the corresponding bit in the CANopen® error register and will send the following error code to the controller:

Error Type	Error Code	Error Register
Over-temperature	0x4231 - 0x4232	4

5 V Reference Output

Root Cause of Error

This error may be caused by the following root causes:

- The limit for over-current has been exceeded.
- A short-circuit to ground has occurred.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Over-current	0x2320	2

Generic Fault Detection

Fault Description

The module JXM-IO-E02 can also detect certain faults which are not directly linked to a specific input or output, such as:

- Internal Communication Failure
- Parameter Mismatch

Detection of internal communcation errors

If the internal communications of the module JXM-IO-E02 fail, this error is reported. If this event occurs, certain inputs and/or output may no longer be controllable and the external controller should consider it a serious failure.

Detection of parameter mismatch

This fault indicates that the two copies of System Parameters stored inside the module JXM-IO-E02 are no longer synchronized. Detection of this fault is not currently implemented, but it can be added in the near future.

Response of the Module to this Error

The module responds to this error in the following levels:

Level	Description
1	The module will send a CANopen® emergency object to the controller.
2	The module will block the function that has caused the error.

The module will set the corresponding bit in the CANopen® error register and will send the following error code:

Error Type	Error Code	Error Register
Internal Communication Failure	0x5002	5
Parameter Mismatch	0x6300	6

11 Operating System Update

Introduction

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

This chapters describes how to perform an operating system update for a system equipped with a JCM-350-E03 controller.

Downloading an Operating System

You can download operating systems from the Jetter AG homepage at **www.jetter.de http://www.jetter.de**. You get to the OS files by clicking on the quick link "Operating System Download" located on the website of the corresponding controller or module.

Mobile Controllers

The operating system of the following mobile controllers can be updated:

- Controller JCM-350-E03

Contents

11.1 Updating the Operating System of the Controller

Introduction

This chapter describes how to update the OS of the controller JCM-350-E03. To transfer the OS file to the controller the following options are available:

- Using the OS update feature of the programming tool JetSym

Contents

Operating System Update Using JetSym

Introduction

The programming tool JetSym offers an easy way to transfer an OS file to the JCM-350-E03.

Prerequisites

- An OS file for the JCM-350-E03 must be available.
- An active CAN connection between JetSym and the controller is set up.
- During booting, the controller is waiting for the OS update, or the OS is already running.
- Make sure that the controller is not de-energized during OS update.

Updating the OS

To update the OS proceed as follows:

Step	Action
1	In JetSym, click on the menu **Build** and select item **Update OS...**, or click in the configuration window of the Hardware Manager on **OS Update**. **Result:** The file selection box 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	Reboot the controller to launch the updating operating system.

12 Application Program

Introduction

This chapter explains how the application program is stored to the JCM-350-E03 and how the user selects the program to be executed.

Required Programmer's Skills

This chapter requires knowledge on how to create application programs in JetSym and how to transmit them via the JCM-350-E03 file system.

Contents

Loading an Application Program

Introduction

If mode selector S11 is in RUN position, the application program is loaded and executed by the file system either on relaunch of the application program via JetSym or on re-boot of the controller.

Loading Process

The application program is loaded by the controller's OS as follows:

State	Description
1	The OS reads the file "/app/start.ini" from the internal flash disk.
2	The OS reads out the path to the application program from the entry "Project".
3	The OS reads out the program name from the entry "Program". The path is relative to the directory "/app".
4	The OS loads the application program from the file <Project>/<Program>.

Application Program - Default Path

Introduction

When uploading the application program from JetSym to the JCM-350-E03, it is stored as file to the internal flash disk. Path and file name are entered into the file "/app/start.ini".

Path and File Name

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

File "/app/start.ini"

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

Element	Description
[Startup]	Section name
Project	Path to the application program. This path is relative to "/app".
Program	Name of the application program file

Example:

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

Result: The application program is loaded from the file "/app/test_program/test_program.es3".

Related Topics

- Storing the Application Program to the SD Card

13 Quick Reference - JCM-350

OS version

This quick reference summarizes the registers and flags of the controller JCM with OS version 1.09.0.200

General Overview - Registers

100000 ... 100999	Electronic Data Sheet (EDS)
101000 ... 101999	Configuration
200000 ... 209999	General system registers
210000 ... 219999	Application program
310000 ... 319999	File system / data files
1000000 ... 1005999	JCM-350: Application registers (remanent; Int/Float)

General Overview - Flags

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

Electronic Data Sheet (EDS)

100500	Interface (0 = CPU)
[Identification]	
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]	**I/O Module**
100808	Features
100809	Diagnostics mask
[Features]	**JCM-350**
100800	Internal version number
100801	MAC Address (Jetter)
100802	MAC Address (device)
100803	Serial interface
100804	Switch
100805	STX
100806	Remanent registers
100808	CAN bus
100809	SD memory card
100810	Motion control
100811	Intelligent slave modules
100812	HTTP / e-mail
100813	Modbus/TCP
100815	LED for SD memory card
100816	User LEDs
100817	RTC

Configuration

	From file /system/ config.ini
101100	IP address
101101	Subnet mask
101102	Default gateway
101103	DNS server
101132	Host name suffix type
101133 ...	Host name (register string)
101151	
101164	JetIP port number
101165	STX debugger port number
	Used by the system
101200	IP address
101201	Subnet mask
101202	Default gateway
101203	DNS server
101232	Host name suffix type
101233 ...	Host name (register string)
101251	
101264	JetIP port number
101265	STX debugger port number

General System Registers

200000	OS version (major * 100 + minor)
200001	Application program is running (bit 0 = 1)
200008	Error register (identical with 210004) Bit 1: Error on JX3 bus Bit 2: Error on JX2 bus 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 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)
200170	Controller type (340/350)
201000	Runtime registers in milliseconds (rw)
201001	Runtime registers in seconds (rw)
201002	Runtime register in register 201003 Units (rw)
201003	* 10 ms units for register 201002 (rw)
201004	Runtime registers in milliseconds (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
202936	Control register - File System 0xc4697a4b: Formatting the flash disk
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
203000	Interface monitoring: JetIP
203001	Interface monitoring: SER
203005	Interface monitoring: Debug server

203100 ... 203107	32-bit overlaying - Flag 0 ... 255
203108 ... 203123	16-bit overlaying - Flag 0 ... 255
203124 ... 203131	32-bit overlaying - Flag 2048 ... 2303
203132 ... 203147	16-bit overlaying - Flag 2048 ... 2303
209700	System logger: Global enable
209701 ... 209739	Enabling system components

Application Program

210000	Application program is running (bit 0 = 1)
210001	JetVM version
210004	Error register (bit-coded) Bit 1: Error on JX3 bus Bit 2: Error on JX2 bus 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 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 a 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 register 210061)
210061	Task priority for the task [reg. 210060]
210063	Length of scheduler table
210064	Index in scheduler table
210065	Task ID in scheduler table
210070	Task ID (for register 210071)
210071	Timer number (0 ... 31)
210072	Manual triggering of a timer event (bit-coded)
210073	End of cyclic task (task ID)
210074	Command for cyclic tasks
210075	Number of timers
210076	Timer number (for register 210077)
210077	Timer value in milliseconds
210100 ... 210199	Task - state
210400 ... 210499	Task - programm address
210600	Task ID of a cyclic task (for register 210601)
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.)

File System / Data File Function

312977	Status of file operation
312978	Task ID

Application Registers

1000000 ... 1005999	JC-350: 32-bit integer or floating point number (permanent)

Special Flags - Interface Monitoring

2088	OS flag - JetIP
2089	User flag - JetIP
2090	OS flag - SER
2091	User flag - SER
2098	OS flag - debug server
2099	User flag - debug server

32 Combined Flags

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

16 Combined Flags

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

32 Combined Special Flags

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

16 Combined Special Flags

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

Overlaid 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 Functions

4	BCD to HEX conversion
5	HEX to BCD conversion
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
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 data file
91	Appending data file
92	Reading data file
96	Deleting data file
110	E-mail feature
150	Configuring NetCopyList
151	Deleting NetCopyList
152	Sending NetCopyList

Appendix

Introduction

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

Contents

A: Technical Data

Introduction

This chapter contains information on electrical and mechanical data, as well as on operating data of the JCM-350-E03.

Contents

Technical Specifications

Connector

Parameter	Description
Manufacturer/Model	Tyco AMP
Article #	963484
Design	70-pin
Coding	A 1

Electrical Data - Power Supply

Parameter	Description
Operating voltage	DC 8.0 ... 32.0 V
Operating voltage - IGNITION FEED	min. DC 5.9 V
Peak Current:	
IGNITION FEED	max. 2.0 A
STANDARD FEED	max. 52.0 A
SAFETY FEED	max. 40.0 A
Overcurrent detection	Yes

Communication

Parameter	Description
Bus type	CAN bus
Protocol	CANopen®
Baud rate	250 kBaud (1 MBaud)
Terminating resistor	Can be activated by means of software

Technical Data - Tri-State Inputs

Parameter	Description
Application	■ for device coding ■ as digital inputs
Type of inputs	Pull-up resistor to IGNITION FEED and pull-down resistor to ground
Tri-state detection	Tri-state operation is detected by a pull-down resistor to ground.
Rated voltage	IGNITION FEED
Threshold level OFF	$\leq$ 1.0 V
Threshold level ON	$\geq$ 4.0 V

Technical Data - Digital Inputs IN 1 through IN 5

Parameter	Description
Type of inputs	Software selectable with either 2 kΩ pull-up to STANDARD FEED or 2 kΩ pull-down to ground.
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Threshold level OFF	≤ 1.0 V
Threshold level ON	≥ 3.5 V

Technical Data - Digital Inputs IN 6 through IN 13

Parameter	Description
Type of inputs	Can be configured as active-high inputs
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Threshold level OFF	51 % of IGNITION FEED
Threshold level ON	51 % of IGNITION FEED
Input impedance	100 kΩ

Technical Data - Digital Outputs (STANDARD FEED)

Parameter	Description
Type of outputs	Active-high output
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Signal voltage OFF	< 1.0 V
Signal voltage ON	$U_{STANDARD}$ - 0.5 V
Load current of OUT 1 through OUT 8	max. 2.5 A
Maximum inrush current	tbd
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Technical Data - Digital Inputs IN 14 through IN 21

Parameter	Description
Type of inputs	Can be configured as active-high inputs
Rated voltage	SAFETY FEED
Permissible voltage range	DC 8 ... 32 V
Threshold level OFF	< 51 % of IGNITION FEED
Threshold level ON	> 51 % of IGNITION FEED
Input impedance	100 kΩ

Technical Data - Digital Outputs (SAFETY FEED)

Parameter	Description
Type of outputs	Active-high output
Rated voltage	SAFETY FEED
Permissible voltage range	DC 8 ... 32 V
Signal voltage OFF	< 1.0 V
Signal voltage ON	U_{SAFETY} - 0.5 V
Load current of OUT 9 through OUT 10	max. 2.5 A
Load current of OUT 11 through OUT 16	max. 5.0 A
Maximum inrush current	tbd
Can be switched off by electronic safety switch	Yes
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Technical Data - Switch Outputs

Parameter	Description
Type of switch outputs	Active-high output
Rated voltage	STANDARD FEED
Permissible voltage range	DC 8 ... 32 V
Signal voltage OFF	< 1.0 V
Signal voltage ON	$U_{STANDARD}$ - 0.5 V
Load current	each 2.5 A max.
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Technical Data - PWM Outputs

Parameter	Description
Operating Modes	■ Current-controlled output ■ PWM output with static duty cycle
Dither function	Yes, at PWM freq: 2 kHz
Resolution	8 bits
Load current	0 ... 2.5 A
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Technical Data - Analog Output

Parameter	Description
Voltage range at 50 mA	0 ... STANDARD FEED
Current range	0 ... 100 mA
Resolution	10 bits
Electrical isolation	none
Short circuit detection	Yes

Technical Data - Analog Inputs

Parameter	Description
Voltage range	■ 0 ... 5 V ■ 0 ... IGNITION FEED
Current range	■ 0 ... 20 mA ■ 4 ... 20 mA
Input impedance at 0 ... 5 V	100 kΩ
Input impedance at 0 ... IGNITION FEED	50 kΩ
Input impedance at 0 ... 20 mA	240 Ω
Resolution	10 bits
Electrical isolation	none

Technical Data - Frequency Inputs

Parameter	Description
Application	■ as frequency counter ■ as two digital inputs
Type of inputs	Software selectable with either 2 kΩ pull-up to STANDARD FEED or 2 kΩ pull-down to ground.
Frequency measurement range	5 Hz ... 20 kHz
Measurement method	time-based
Result of measurement	Period of the signal in nanoseconds
Resolution	62.5 ns

Technical Data - H-Bridge

Parameter	Description
Application	■ used as H-Bridge ■ as two independent digital inputs
Rated output current	max. 2.5 A
Accuracy of current measurement (H-bridge)	< 100 mA
Short-circuit proof	Yes
Overcurrent detection	Yes
No-load detection	Yes

Technical Data - Regulated Output

Parameter	Description
Regulated voltage	DC 5 V
Load current	max. 250 mA
Overcurrent detection	Yes

Protective and Diagnostic Functions

Type of Fault	Response
Short circuit	■ The faulty function is disabled automatically ■ A CANopen® emergency object is sent to the controller ■ The error message is stored to a history list which is compatible with the CANopen® standard
Overload	
No load (cable breakage)	

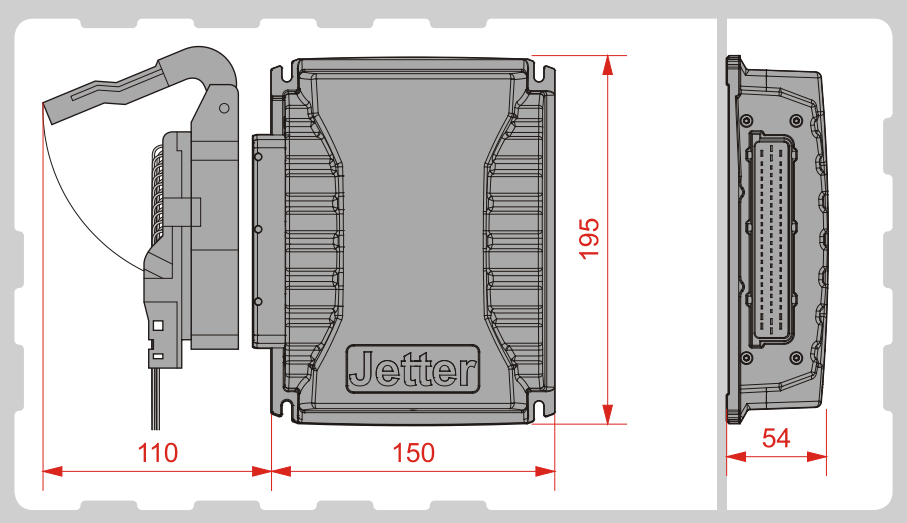
Physical Dimensions

Introduction

This chapter details the physical dimensions of the JCM-350-E03 and the conditions for installation.

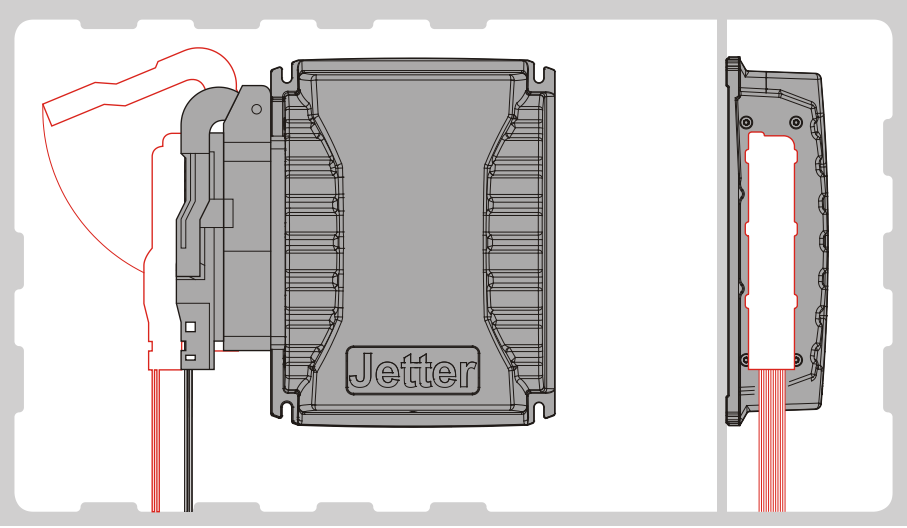
Physical Dimensions

The diagram shows the dimensions of the JCM-350-E03.



Space Required for Installation and Service

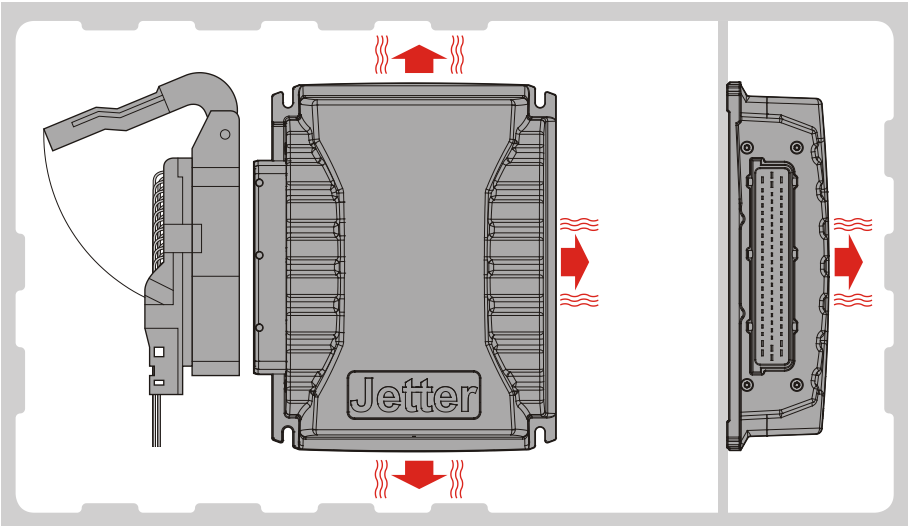
The diagram shows the space required for the JCM-350-E03.



Ensure there is enough space around the connector for servicing requirements. It should be possible to disconnect the connector at any time.

Space Required to Protect Against Overheating

The diagram indicates the safety distances to protect against overheating.



Please note:

- The JCM-350-E03 increases the temperature of the environment as a result of heat emission under load.
- The JCM-350-E03 operates without interruption at an ambient temperature of up to +85 °C.

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

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

Operating Parameters - Environment and Mechanics

Environment

Parameter	Value	Standard
Operating temperature range	-40 ... +85 °C	
Storage temperature range	-40 ... +85 °C	DIN EN 61131-2 DIN EN 60068-2-1 DIN EN 60068-2-2
Air humidity	10 ... 95 %	DIN EN 61131-2
Climate test	Humid heat	DIN EN 60068-2-30
Pollution degree	2	DIN EN 61131-2

Mechanical Parameters

Parameter	Value	Standard
Vibration resistance	Vibration, broadband noise	DIN EN 60068-2-6 Severity level 2
Shock resistance	30 g occasionally, 18 ms, sinusoidal half-wave, 3 shocks in the directions of all three spatial axes	DIN EN 60068-2-27
Degree of protection	IP68	DIN EN 60529 including all changes to date

Operating Parameters - EMC

EMC - Emitted Interference

As per Directive 72/245/EEC with all amendments up to 2009/19/EC checked and compliant.

EMC - Interference Immunity

Parameter	Value	Standard
Interference immunity to conducted faults	compliant	Directive 72/245/EEC with all changes up to 2009/19/EC
Interference immunity to external magnetic field	20 ... 1,000 MHz: 100 V/m 1,000 ... 2,000 MHz: 30 V/m	Directive 72/245/EEC with all changes up to 2009/19/EC
Load Dump	Impulse 5b 70 V	ISO 7637-2

B: Index

Jetter AG

Graeterstrasse 2

D-71642 Ludwigsburg

Germany

Phone: +49 7141 2550-0

Phone - Sales: +49 7141 2550-433

Fax - Sales: +49 7141 2550-484

Hotline: +49 7141 2550-444

Internet: http://www.jetter.de

E-Mail: sales@jetter.de

Jetter Subsidiaries

Jetter (Switzerland) AG

Münchwilerstrasse 19

CH-9554 Tägerschen

Switzerland

Phone: +41 71 91879-50

Fax: +41 71 91879-69

E-Mail: info@jetterag.ch

Internet: http://www.jetterag.ch

Jetter UK Ltd.

Old Witney Road

Eynsham

OX29 4PU Witney

Great Britain

Phone: +44 1865 883346

Fax: +44 1865 883347

E-Mail: info@jetter.uk.com

Internet: http://www.jetter.uk.com

Jetter USA Inc.

13075 US Highway 19 North

Florida - 33764 Clearwater

U.S.A

Phone: +1 727 532-8510

Fax: +1 727 532-8507

E-Mail: bschulze@jetterus.com

Internet: http://www.jetter.de