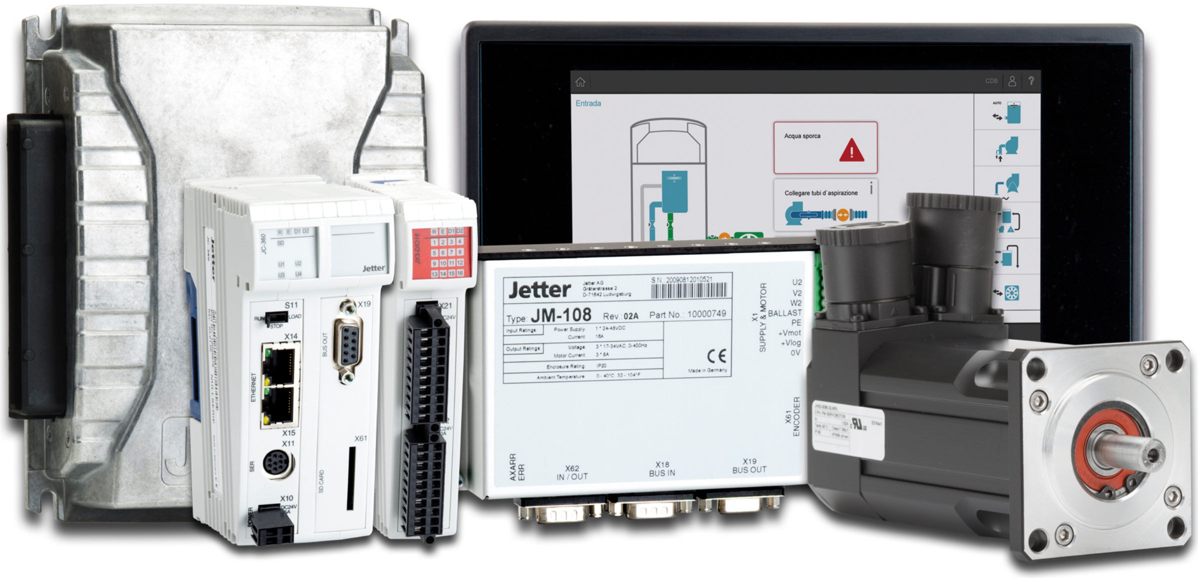


Application-Oriented Manual

User-programmable Prim Interface

60882048

We automate your success.

Item # 60882048

Revision 1.10

April 2017 / Printed in Germany

This document has been compiled by Jetter AG with due diligence, and based on the known state of the art.

In the case of modifications, further developments or enhancements to products shipped in the past, a revised document will be supplied only if required by law, or deemed appropriate by Jetter AG. Jetter AG shall not be liable for errors in form or content, or for missing updates, as well as for damages or disadvantages resulting from such failure.

Table of Contents

1 User-programmable serial interface

Introduction

This chapter describes how to address the serial interface of the controller from within the application program to allow sending and receiving characters.

Applications

The user-programmable serial interface lets you connect devices which use communication protocols that are not supported by the OS of the controller, Fields of application, for example, are:

- Scales
- Scanners
- Display elements
- Frequency inverters
- Temperature controllers
- etc.

Required programmer's skills

This chapter addresses programmers of application programs with experience in data transfer via asynchronuous serial interfaces. Expertise in the following areas is prerequisite:

- Wiring of serial interfaces
- Communication parameters (baud rate, parity, etc.)
- Transmit and receive buffers
- etc.

Contents

1.1 Connection

Introduction

This chapter covers the connection to a user-programmable serial interface of the device.

Contents

Topic **Page**

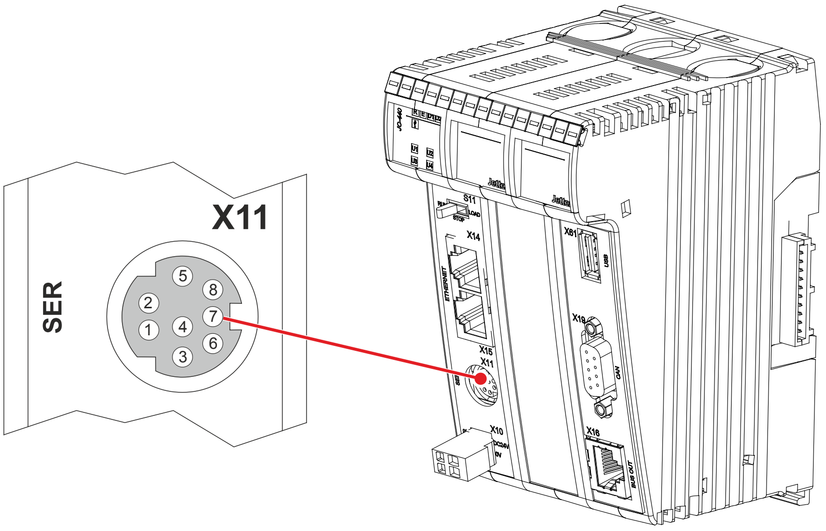
Serial interface port X11

Devices to connect with this port

Port X11 lets you connect the following devices:

- PC
- HMI by Jetter AG
- Any device with RS-232/422/485 interface

Pin assignment



Pin	Signal	Description
1	RDA	RS-422; receive data inverted
2	GND	Reference potential
3	RDB	RS-422; receive data not inverted
4	RxD	RS-232; receive data
5	SDB	RS-422; sending data not inverted RS-485; sending / receive data not inverted
6	DC 24 V	HMI supply voltage
7	SDA	RS-422; sending data inverted RS-485; sending / receive data inverted
8	TxD	RS-232; sending data

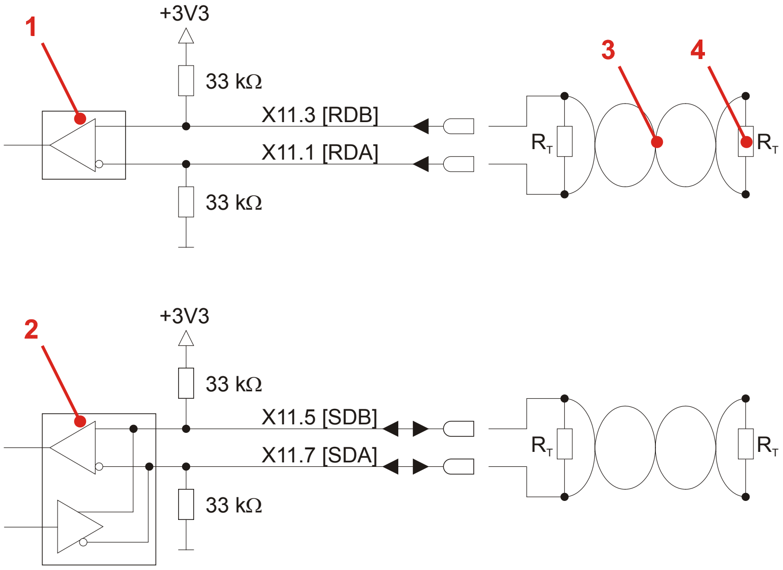
Restrictions

Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.

This means:

While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.

Block diagram



Number	Part	Function in the case of RS-422	Function in the case of RS-485
1	Receiver	Receives data	Unused
2	Receiver / sender	Sends data	Receives and sends data
3	Serial line	Twisted line of the serial interface	
4	R_T	Terminating resistor	

Terminating resistor

Connect a terminating resistor to both serial lines in the following cases:

- Long lines
- High baud rates

Select a terminating resistor which corresponds to the impedance of the line used.

Technical specifications

Parameter	Description
Type of terminal	MiniDIN, shielded
Number of pins	8
Electrical isolation	None
Number of interfaces	1 serial port
Interface standards	RS-232/RS-422/RS-485-2
Baud rates	JC-4xx: 1,200 ... 115,200 baud JC-3xx: 2,400 ... 115,200 baud
Bits per character	5, 6, 7, 8
Number of stop bits	1, 2
Parity	even, odd, none

Cables for port X11

For connecting devices to port X11 you can order the following cables:

Item no.	Item	Description
60867209	KAY_0576-0050	JetControl to modem with 9-pin Sub-D, length 0.5 m
60868359	Cable assy # 196 2.5M	JetControl to PC with 9-pin Sub-D, length 2.5 m
60860013	Cable assy # 196 5M	JetControl to PC with 9-pin Sub-D, length 5 m
60868956	Cable assy # 196 8M	JetControl to PC with 9-pin Sub-D, length 8 m
60860011	Cable assy # 192 2.5M	JetControl to HMI with 15-pin Sub-D, length 2.5 m
60860012	Cable assy # 193 5M	JetControl to HMI with 15-pin Sub-D, length 5 m
60872142	Cable assy # 192 10M	JetControl to HMI with 15-pin Sub-D, length 10 m
60872884	Cable assy # 192 15M	JetControl to HMI with 15-pin Sub-D, length 15 m
60864359	KAY_0386-0250	JetControl to LCD 60 with 15-pin Sub-D, length 2.5 m
60864360	KAY_0386-0500	JetControl to LCD 60 with 15-pin Sub-D, length 5 m
60864897	KAY_0533-0025	JetControl to LCD 52/54 with 15-pin Sub-D, length 0.25 m
60864257	Cable assy # 197 5M	JetControl to JetView 200/300 with 9-pin Sub-D, length 5 m
60871930	Cable assy # 197 12M	JetControl to JetView 200/300 with 9-pin Sub-D, length 12 m

1.2 Functioning principle of the user-programmable interface

Introduction

This chapter describes the functioning principle of the user-programmable serial interface.

Restrictions

When using the user-programmable serial interface the following restrictions apply:

- Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.

  This means: While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.
- The controller does not execute the pcomX protocol any more.

  This means: This protocol can no longer be used to communicate, for example, with JetSym, JetViewSoft or HMIs via this protocol.

Contents

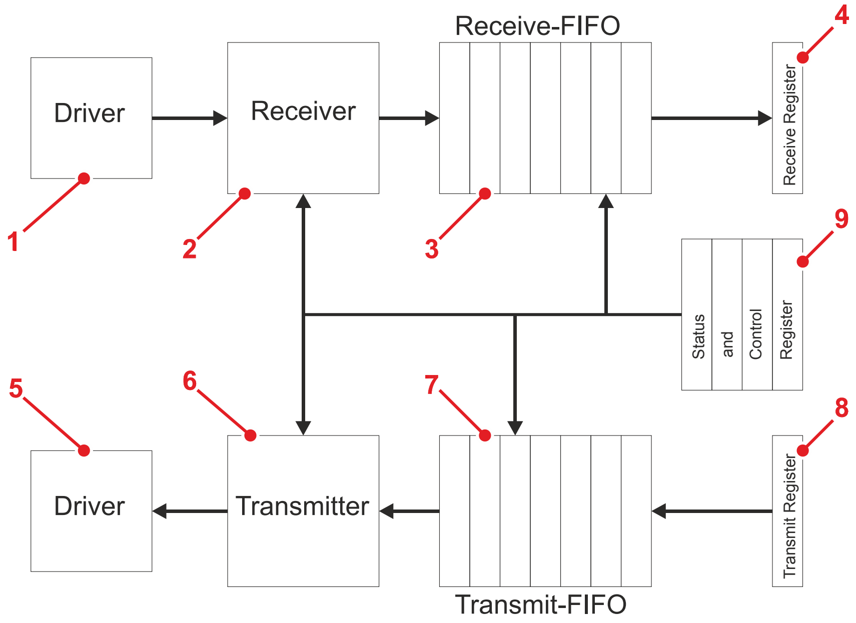
Functioning principle

Introduction

The OS of the device provides for the user-programmable serial interface a receive buffer and a transmit buffer. They can be used to adjust the transfer rate between application program and serial interface.

Block diagram

The following illustration shows the block diagram of the user-programmable serial interface:



Elements of the interface

The user-programmable serial interface consists of the following elements:

Number	Part	Function
1	Interface driver	Converts signals of different interface standards (RS-232, RS-422, RS-485) into internal signal levels
2	Addressee	Performs serial/parallel conversion
3	Receive buffer	Buffer for received characters
4	Receive register	Read access to this register reads the received characters in the receive buffer (3).
5	Interface driver	Converts internal signal levels into signals of different interface standards (RS-232, RS-422, RS-485)
6	Transmitter	Performs parallel/serial conversion
7	Transmit buffer	Buffer for characters to be sent
8	Transmit register	Write access to this register causes the characters to be entered into the transmit buffer (7) and to be sent by the transmitter (6).

Number	Part	Function
9	Status and control register	Query of filling levels and error states of buffers; setting of transmission parameters

Receiving a character

A character is received as described below:

Step	Description
1	The interface driver converts signals "on the line" into internal signal levels and forwards them to the receiver.
2	The receiver performs serial/parallel conversion of this character and checks the set communication parameters.
3	The receiver enters the character into the receive buffer if there is any place left. Otherwise, the character is discarded and buffer overflow is signaled.
4	Via receive register the character can be read out of the receive buffer.

Sending a character

A character is transmitted as described below:

Step	Description
1	Via transmit register the character is entered into the transmit buffer if there is any place left. Otherwise the character is discarded.
2	Once the transmitter has sent a character, it reads the next character from the transmit buffer.
3	The transmitter performs parallel/serial conversion and sends this character to the interface driver using the set communication parameters.
4	The interface driver converts internal signal levels into the various interface standards.

Error detection

When receiving characters, the following errors are detected by the controller and displayed in the register *Error state*:

Error	Description	Effect
Framing error	The format of the received character does not match the set parameters.	The erroneous character(s) is (are) stored in the receive buffer and error bit *Framing error* is set. The error counter is incremented.
Parity error	The parity bit of the received character is not correct.	The erroneous character is stored in the receive buffer and error bit *Parity error* is set. The error counter is incremented.
Buffer overflow	A character is received, although the receive buffer is full.	The character is discarded and error bit *Overflow* is set. The error counter is incremented.

Troubleshooting

As error bits cannot be assigned to individual characters in the receive buffer, all characters should be removed from the receive buffer and discarded when an error bit is set.

Possible causes of error and troubleshooting:

Error	Possible cause	Troubleshooting
Framing error	Jammed data transmission caused by EMC problems, defective cables or connectors	▪ Check the wiring and connectors. ▪ Use shielded cables. ▪ Do not lay cables near sources of interference.
	Incorrectly set communication parameters (baud rate, number of stop bits, etc.)	▪ Make sure the set communication parameters are consistent with the settings of the connected device.
Parity error	Jammed data transmission caused by EMC problems, defective cables or connectors	▪ Check the wiring and connectors. ▪ Use shielded cables. ▪ Do not lay cables near sources of interference.
	Incorrectly set parity	▪ Make sure the parity setting is consistent with the setting of the connected device.
Buffer overflow	The external device sends characters at too high a rate and the application program is not able to read them out of the receive buffer in due time.	▪ Program a software handshake. ▪ Set a lower baud rate. ▪ Make sure that characters are read out from the receive buffer faster. To achieve this the program code has to be optimized.

1.3 Registers

Introduction

This chapter describes the registers associated with the user-programmable serial interface. These registers are used for the following tasks:

- Parameterizing the interface
- Sending characters
- Receiving characters

Contents

Register numbers

Introduction

The registers of each interface are combined into one register block. The basic register number of this block is dependent on the controller.

Register numbers

Basic register number	Register numbers
103000	103000 ... 103019

Determining register numbers

In this chapter, only the last two figures of a register number are specified, e.g. MR 14. To calculate the complete register number, add the basic register number of the corresponding device to the respective device, e.g. 103000.

Registers - Overview

Register	Description
MR 0	Error state
MR 1	Protocol
MR 2	Baud rate
MR 3	Number of data bits per character
MR 4	Number of stop bits
MR 5	Parity
MR 6	Interface standard
MR 10	Transmit buffer
MR 11	Transmit buffer filling level
MR 12	Receive buffer (without deleting characters on reading)
MR 13	Receive buffer (with deleting characters on reading)
MR 14	Receive buffer filling level
MR 15	Receive buffer, 16-bit, little endian
MR 16	Receive buffer, 16-bit, big endian
MR 17	Receive buffer, 32-bit, little endian
MR 18	Receive buffer, 32-bit, big endian
MR 19	Error counter

Registers - Description

Introduction

When entering values into control registers MR 1 through MR 6, the entire interface is re-initialized and the sending and receive buffers are cleared.

MR 0

Error state

This register displays errors which have been detected on receiving a character as bit-coded value.

Meaning of the individual bits		
Bit 12	**Buffer overflow**	
	1 =	Although the receive buffer is full, one or more characters have been received
Bit 13	**Parity error**	
	1 =	The parity bit of the received character is not correct.
Bit 14	**Framing error**	
	1 =	The format of the received character does not match the set parameters.

Module register properties	
Type of access	Read / write (clearing)

MR 1

Protocol

This register lets you set the protocol which is supported by the OS of the controller. That is, this register is for defining how the interface is used.

Module register properties		
Values	1	System logger
	2	User-programmable interface
	3	PcomX
Value after reset	3	

MR 2

Baud rate

This register lets you set the baud rate.

Module register properties	
Values	JC-4xx: 1,200 ... 115,200 JC-3xx: 2,400 ... 115,200
Value after reset	9,600

MR 3

Number of data bits per character

This register lets you set the number of data bits per character.

Module register properties	
Values	5, 6, 7, 8
Value after reset	8

MR 4

Stop bits

This register lets you set the number of stop bits per character.

Module register properties		
Values	1	1 stop bit
	2	1.5 stop bits if MR 3 = 5
		2 stop bits if MR 3 = 6, 7, 8
Value after reset	1	

MR 5

Parity

This register lets you set the parity of a character.

Module register properties		
Values	0	None (no parity)
	1	Odd parity
	2	Even parity
	3	1 (mark)
	4	0 (space)
Value after reset	2	

MR 6

Interface standard

This register lets you set the hardware interface which is used to receive and send characters.

Module register properties		
Values	0	RS-232
	1	RS-422
	2	Reserved
	3	RS-485, 2-wire
Value after reset	1	

MR 10

Sending buffer

The character that has to be sent must be entered into this register.

- If the sending buffer is able to accommodate the character, it is entered into this buffer. This character will be sent once all previously entered characters have been sent.
- Prior to sending characters from the application program, it must be checked whether the sending buffer is able to accommodate characters. This can be checked by reading out MR 11.
- The sending buffer functions according to the FIFO principle. The first character entered is sent first.

Module register properties		
Values	0 ... 31	5 bits per character
	0 ... 63	6 bits per character
	0 ... 127	7 bits per character
	0 ... 255	8 bits per character
Type of access	Read access	Character written last
	Write	Sending a character

MR 11

Sending buffer filling level

This register shows how many characters the sending buffer accommodates.

There is space for 32,768 characters max. within the buffer.

Module register properties	
Values	0 ... 32,768

MR 12

Receive buffer, 8 bits (without deleting the character on reading)

This register shows the "oldest" character stored in the receive buffer. On reading, this character will not be removed from the buffer.

Module register properties		
Values	0 ... 31	5 bits per character
	0 ... 63	6 bits per character
	0 ... 127	7 bits per character
	0 ... 255	8 bits per character
Type of access	Read access	Oldest character in buffer
Takes effect	if MR 14 > 0	

MR 13

Receive buffer, 8 bits (with deleting the character on reading)

This register shows the "oldest" character stored in the receive buffer. This character is removed from the buffer. Thus, the character received next can be read out during the next read access.

Module register properties		
Values	0 ... 31	5 bits per character
	0 ... 63	6 bits per character
	0 ... 127	7 bits per character
	0 ... 255	8 bits per character
Type of access	Read access	Oldest character in the buffer
Takes effect	if MR 14 > 0	

MR 14

Receive buffer filling level

This register shows how many characters the receive buffer accommodates. Each read access to MR 13 decrements this register by 1.

Module register properties	
Values	0 ... 32,768

MR 15

Receive buffer, 16-bit, little endian

Read access to this register removes 2 characters from the receive buffer and returns them as 16-bit value.

Assignment:

Character	Bits in register
First	Bit 0 ... 7
Second	Bit 8 ... 15

Module register properties		
Values	0 ... 65,535	
Type of access	Read access	Removes 2 characters from the buffer
Takes effect	if MR 14 > 1	

MR 16

Receive buffer; 16-bit; big endian

Read access to this register removes 2 characters from the receive buffer and returns them as 16-bit value.
Assignment:

Character	Bits in register
First	Bit 8 ... 15
Second	Bit 0 ... 7

Module register properties		
Values	0 ... 65,535	
Type of access	Read access	Removes 2 characters from the buffer
Takes effect	if MR 14 > 1	

MR 17

Receive buffer, 32-bit, little endian

Read access to this register removes 4 characters from the receive buffer and returns them as 32-bit value.

Assignment:

Character	Bits in register
First	Bit 0 ... 7
Second	Bit 8 ... 15
Third	Bit 16 ... 23
Fourth	Bit 24 ... 31

Module register properties		
Values	-2,147,483,648 ... 2,147,483,647	
Type of access	Read access	Removes 4 characters from the buffer
Takes effect	if MR 14 > 3	

MR 18

Receive buffer; 32-bit; big endian

Read access to this register removes 4 characters from the receive buffer and returns them as 32-bit value.

Assignment:

Character	Bits in register
First	Bit 24 ... 31
Second	Bit 16 ... 23
Third	Bit 8 ... 15
Fourth	Bit 0 ... 7

Module register properties		
Values	-2,147,483,648 ... 2,147,483,647	
Type of access	Read access	Removes 4 characters from the buffer
Takes effect	if MR 14 > 3	

MR 19

Error counter

This register shows the number of detected errors.

Module register properties	
Values	0 ... 2,147,483,647
Type of access	Read/write (clearing)

1.4 Programming

Introduction

This chapter describes how to configure the serial interface of the controller for use as user-programmable serial interface and how to send receive characters via this interface.

Contents

Configuring the interface

Introduction

Module registers MR 1 through MR 6 are used to configure the user-programmable serial interface.

Prerequisites

This guide proceeds from the assumption that wiring between controller and remote device is according to the standard of the selected interface.

Configuring the interface

To configure the user-programmable serial interface proceed as follows:

Step	Action
1	Enter value 1 into MR 2.
2	Enter the desired communication parameters into MR 2 through MR 6.

Result: The serial interface is set as user-programmable interface. Both the transmit buffer and receive buffer are cleared.

Sending characters

Introduction

A character is sent by entering it into the register *Transmit buffer*.

Prerequisites

This guide proceeds from the assumption that the user-programmable serial interface has been configured.

Sending characters

To send characters via user-programmable serial interface proceed as follows:

Step	Action
1	Check the transmit buffer filling level, whether there is enough space in the transmit buffer.
2	If there is no space in the transmit buffer, wait, until there is enough space.
3	Enter the character to be sent into register *Transmit buffer*.

Result: The character is written into the transmit buffer and will be sent from there.

Sending texts

Introduction

An easy way to send texts via the user-programmable serial interface is redirecting the instructions `DisplayText()` and `DisplayText2()` to **Device 9**.

Prerequisites

This guide proceeds from the assumption that the following conditions are met:

- The user-programmable serial interface is configured.
- The user is familiar with the options of the instructions `DisplayText()` and `DisplayText2()` (refer to the online help which comes with JetSym).

Restrictions

When redirecting the instructions `DisplayText()` and `DipslayText2()` to the user-programmable serial interface the following restrictions apply:

- The cursor position will not be evaluated.
- The characters for "Delete Screen" and "Delete to End of Line" are of no special significance and will be output without any changes.

Sending texts

To send texts via user-programmable serial interface proceed as follows:

Step	Action
1	Use the instruction `DisplayText()` or `DisplayText2()`.
2	Specify **Device 9**.

Result: The task waits at this instruction until all characters have been entered into the transmit buffer.

Sending values

Introduction

The instruction `DisplayValue()` allows redirection of values to **Device 9**. This way, values can easily be sent via user-programmable serial interface.

Prerequisites

This guide proceeds from the assumption that the following conditions are met:

- The user-programmable serial interface is configured.
- The user is familiar with the options of the instruction `DisplayValue()` (refer to the online help which comes with JetSym).

Restrictions

When redirecting instruction `DisplayValue()` to the user-programmable serial interface the following restriction applies:

- The cursor position will not be evaluated.

Sending values

To send values via user-programmable serial interface proceed as follows:

Step	Action
1	The special registers for formatting the display, which are used in connection with the instruction `DisplayValue()`, have to be set to the desired values.
2	Use the instruction `DisplayValue().`
3	Specify **Device 9**.

Result: The task waits at this instruction until all characters have been entered into the transmit buffer.

Receiving characters

Introduction

A character is received by reading characters from register *Receive buffer*.

Prerequisites

This guide proceeds from the assumption that the user-programmable serial interface has been configured.

Receiving characters

To receive characters via user-programmable serial interface proceed as follows:

Step	Action
1	Check the filling level of the receive buffer to make sure that it contains at least 1 character.
2	Read the character from the register *Receive buffer*.

Result: The character is taken from the receive buffer.

Receiving values

Introduction

Values are received by reading characters from *Receive buffer* registers MR 15 through MR 18.

Prerequisites

This guide proceeds from the assumption that the user-programmable serial interface has been configured.

Receiving values

To receive values via user-programmable serial interface proceed as follows:

Step	Action
1	Check the filling level of the receive buffer to make sure that it contains at least 2 or 4 characters.
2	Read the values from *Receive buffer* registers MR 15 through MR 18.

Result: The characters are read from the receive buffer.

2 User-programmable IP interface

The user-programmable IP interface

The user-programmable IP interface allows to send or receive any data via Ethernet interface on the device using TCP/IP or UDP/IP. When using this feature, data processing is completely carried out by the application program.

Applications

The user-programmable IP interface allows the programmer to carry out data exchange via Ethernet connections which do not use standard protocols, such as FTP, HTTP, JetIP or Modbus/TCP. The following applications are possible:

- Server
- Client
- TCP/IP
- UDP/IP

Required programmer's skills

To be able to program user-programmable IP interfaces the following knowledge of data exchange via IP networks is required:

- IP addressing (e.g. IP address, port number, subnet mask)
- TCP (e.g. connection establishment/termination, data stream, data backup)
- UDP (e.g. datagram)

Restrictions

For communication via user-programmable IP interface, the programmer must not use any ports which are already used by the operating system. Therefore, do not use the following ports:

Protocol	Port number	Default value	User
TCP	Depending on the FTP client	20	FTP server (data)
TCP	21		FTP server (controller)
TCP	23		System logger
TCP	80		HTTP server
TCP	From the file /EMAIL/email.ini	25, 110	E-mail client
TCP	502		Modbus/TCP server
TCP, UDP	1024 - 2047		Various
TCP, UDP	IP configuration	50000, 50001	JetIP
TCP	IP configuration	52000	Debug server

Contents

2.1 Programming

Introduction

The user-programmable IP interface is used to carry out data exchange between application program and network client via TCP/IP or UDP/IP connections. For this purpose, function calls are used. These function calls are included in the programming language of the device. To program this feature, proceed as follows:

Step	Action
1	Initializing the user-programmable IP interface
2	Open connections
3	Transfer data
4	Terminate the connections

Technical specifications

Technical data of the user-programmable IP interface:

Function	Description
Number of connections	20
Maximum data size	4,000 bytes
Number of receive buffers per connection	4

Restrictions

While the device is processing one of the functions of the user-programmable IP interface, tasks having called the functions should not be stopped through `TaskBreak` or restarted through `TaskRestart`.

Failure to do so could result in the following errors:

- Connections do not open
- Data loss during sending or receiving
- Connections remain open unintentionally
- Connections are closed unintentionally

Table of contents

Initializing the user-programmable IP interface

Introduction

This function must be initialized each time the application program is launched.

Function declaration

```
Function ConnectionInitialize():Int;
```

Return value

The following return value is possible:

Return value	
0	Always

How to use this function

The function is used and its return value assigned to a variable for further utilization in the following way:

```
Result := ConnectionInitialize();
```

Operating principle

The device processes this function in the following steps:

Step	Description
1	The device closes all open connections of the user-programmable IP interface.
2	The device initializes all OS-internal data structures of the user-programmable IP interface.

Related topics

- **Establishing a connection** (see page 35)
- **Terminating a connection** (see page 44)
- **Sending data** (see page 39)
- **Receiving data** (see page 41)

Establishing a connection

Introduction

Before data can be sent or received, a connection has to be established. Here, the following criteria have to be discerned:

- Which transaction log (TCP or UDP) has to be used?
- Is it a client or a server that has to be installed?

Function declaration

```
Function ConnectionCreate(ClientServerType:Int,
                          IPType:Int,
                          IPAddr:Int,
                          IPPort:Int,
                          Timeout:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment

`ClientServerType`	Client = 1 = CONNTYPE_CLIENT
	Server = 2 = CONNTYPE_SERVER	

`IPType`	UDP/IP = 1 = IPTYPE_UDP TCP/IP = 2 = IPTYPE_TCP	
`IPAddr`	Valid IP address	Required only for TCP/IP client
`IPPort`	Valid IP port number	Will be ignored for UDP/IP client
`Timeout`	0 ... 1,073,741,824 [ms]	0 = infinitely

Return value

If the return value was positive, the connection could be established. If the returned value was negative, an error occurred and the connection could not be established.

Return value	
> 0	A positive return value must be stored in a variable. It must be made available as a handle at activating the functions `Send data`, `Receive data`, and `Terminate connection`.
-1	Error during connection set-up
-2	Internal error
-3	Invalid parameter
-8	Timeout

Using this function with a TCP/IP client

If a client is to establish a TCP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_CLIENT,
                           IPTYPE_TCP,
                           IP#192.168.75.123,
                           46000,
                           T#10s);
```

Functioning principle with a TCP/IP client

The task stops at the program line until the connection is established or the specified timeout has elapsed. This function is processed in the following steps:

Step	Description	
1	The device tries to establish a TCP/IP connection via port 46000 to the network client with IP address 192.168.75.123.	
2	**If ...**	**... then ...**
	the network client has accepted the connection,	the function is terminated and a positive value is returned as handle for further access to the connection.
	the connection could not be established and the timeout of 10 seconds has not elapsed yet,	step 1 is carried out.
	an error has occurred or the timeout has elapsed,	the function is terminated and a negative value is returned.

Using this function with a TCP/IP server

If a server is to establish a TCP/IP connection to a client, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_SERVER,
                           IPTYPE_TCP,
                           0,
                           46000,
                           T#100s);
```

Functioning principle with a TCP/IP server

The task stops at the program line until the connection is established or the specified timeout has elapsed. This function is processed in the following steps:

Step	Description	
1	The device sets up TCP/IP port 46000 for receiving connection requests.	
2	**If ...**	**... then ...**
	the network client has established a connection,	no further connection requests to this port are accepted, the function is terminated and a positive value is returned as handle for further access to the connection.
	the connection could not be established and the timeout of 100 seconds has not elapsed yet,	the system waits for a connection to be established.
	an error has occurred or the timeout has elapsed,	the function is terminated and a negative value is returned.

Using this function with a UDP/IP client

If a client is to establish a UDP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_CLIENT,
                           IPTYPE_UDP,
                           0,
                           0,
                           0);
```

Functioning principle with a UDP/IP client

UDP is a connectionless communication mode. For this reason, the device opens only one communication channel for sending data to a network client. This function is processed in the following steps:

Step	Description	
1	The device sets up a UDP/IP communication channel for sending data.	
2	**If ...**	**... then ...**
	no error has occurred,	the function is terminated and a positive value is returned as handle for further access to the connection.
	an error has occurred,	the function is terminated and a negative value is returned.

Using this function with a UDP/IP server

If a server is to establish a UDP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_SERVER,
                           IPTYPE_UDP,
                           0,
                           46000,
                           0);
```

Functioning principle with a UDP/IP server

UDP is a connectionless communication mode. For this reason, the device opens only one communication channel for receiving data from a network client. This function is processed in the following steps:

Step	Description	
1	The device sets up a UDP/IP communication channel at port 46000 for receiving data.	
2	**If ...**	**... then ...**
	no error has occurred,	the function is terminated and a positive value is returned as handle for further access to the connection.
	an error has occurred,	the function is terminated and a negative value is returned.

Related topics

- **Terminating a connection** (see page 44)
- **Sending data** (see page 39)
- **Receiving data** (see page 41)
- **Initializing the user-programmable IP interface** (see page 34)

Sending data

Introduction

Data can be sent via a previously established connection.

Function declaration

```
Function ConnectionSendData(IPConnection:Int,
                            IPAddr:Int,
                            IPPort:Int,
                            Const Ref SendData,
                            DataLen:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
IPConnection	Handle	Return value of the function ConnectionCreate()
IPAddr	Valid IP address	Required only for UDP/IP client
IPPort	Valid IP port number	Required only for UDP/IP client
SendData	address of the data block to be sent	
DataLen	1 ... 4,000	Data block length in bytes

Return value

The following return values are possible:

Return value	
0	Data have been sent successfully
-1	Error when sending, e.g. connection interrupted
-3	Invalid handle, e.g. sending via a UDP/IP server

Using this function with a TCP/IP connection

If data are to be sent via a TCP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionSendData(hConnection,
                             0,
                             0,
                             SendBuffer,
                             SendLen);
```

Functioning principle with a TCP/IP connection

When using TCP/IP, data are sent via a previously opened connection. Therefore, specification of the IP address and IP port number is not required anymore and can be ignored in the function.

In the following situations, the task is not processed further after issuing this function call:

- The data have been sent and their reception has been confirmed.
- An error has occurred.

Using this function with a UDP/IP connection

If, with a client, data are to be sent via a UDP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionSendData(hConnection,
                             IP#192.168.75.123,
                             46000,
                             SendBuffer,
                             SendLen);
```

Functioning principle with a UDP/IP connection

With UDP/IP there is no connection between two given network clients. Therefore, with each function call data can be sent to another client or another port. The task will pause at this function call, until the data are sent.

You will not get any acknowledgment of the remote network client having received the data.

UDP/IP-client and -server

A UDP/IP-client connection is for sending data only. The sending port is set by the operating system.

A UDP/IP-server connection is for both sending and receiving data. The port which was specified at opening up the communication is used as sending port.

Related topics

- **Initializing the user-programmable IP interface** (see page 34)
- **Establishing a connection** (see page 35)
- **Terminating a connection** (see page 44)
- **Receiving data** (see page 41)

Receiving data

Introduction

Data can be sent via a previously established TCP/IP connection or via a UDP/IP connection of a server.

Via UDP/IP connection of a client data can not be received, but only sent.

Restrictions

Data packets which are received via network must be fetched by the application program. Per connection, four packets as a maximum are stored temporarily in the operating system of the controller. All further packets are discarded.

Function declaration

```
Function ConnectionReceiveData(IPConnection:Int,
                               Ref IPAddr:Int,
                               Ref IPPort:Int,
                               Ref ReceiveData,
                               DataLen:Int,
                               Timeout:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
IPConnection	Handle	Return value of the function ConnectionCreate()
IPAddr	Address of a variable for saving the IP address of the sender	Required only for UDP/IP server
IPPort	Address of a variable for saving the IP port number of the sender	Required only for UDP/IP server
ReceiveData	Address of the data block to be received	
DataLen	1 ... 4,000	Maximum data block length in bytes
Timeout	0 ... 1,073,741,824 [ms]	0 = infinite

Return value

The following return values are possible:

Return value	
> 0	Number of received data bytes
-1	Error when receiving data, e.g. connection interrupted
-3	Invalid handle, e.g. receiving data via a UDP/IP client
-8	Timeout

Using this function with a TCP/IP connection

If data are to be received via a TCP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionReceiveData(hConnection,
                                Dummy,
                                Dummy,
                                ReceiveBuffer,
                                sizeof(ReceiveBuffer),
                                T#10s);
```

Functioning principle with a TCP/IP connection

When using TCP/IP, data are sent via a previously opened connection. Therefore, specification of the IP address and IP port number is not required anymore and can be ignored in the function.

In the following situations, the task is not processed further after issuing this function call:

- The data have been received.
- An error has occurred.

In case of a TCP/IP connection, data are sent as data stream.

The device processes this function in the following steps:

Step	Description	
1	The device waits until data have been received, but no longer than the specified timeout.	
2	**If ...**	**... then ...**
	the timeout has elapsed or the connection has been terminated,	the function is exited and an error message is issued.
	data have been received,	they are copied to the receive buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3.
3	**If ...**	**... then ...**
	more data have been received than could have been copied into the receive buffer,	these are buffered by the device to be fetched by further function calls.
4	The function is exited and the number of data, which have been copied into the receive buffer, is returned.	

Using this function with a UDP/IP server

If, with a server, data are to be received via a UDP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionReceiveData(hConnection,
                                IPAddr,
                                IPPort,
                                ReceiveBuffer,
                                sizeof(ReceiveBuffer),
                                T#10s);
```

Functioning principle with a UDP/IP server

In the following situations, the task is not processed further after issuing this function call:

- All data have been received.
- An error has occurred.

In case of a UDP/IP connection, data are sent as datagram.
The device processes this function in the following steps:

Step	Description	
1	The device waits until all data of a datagram have been received, but no longer than the specified timeout.	
2	**If ...**	**... then ...**
	the timeout has elapsed or the connection has been terminated,	the function is exited and an error message is issued.
	data have been received,	they are copied to the receive buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3.
3	**If ...**	**... then ...**
	... more data have been received than could be copied into the receive buffer - that is, if the sent datagram is too large,	... these data are discarded.
4	The sender's IP address and IP port number are transferred into the variables which are given along with the data.	
5	The function is exited and the number of data, which have been copied into the receive buffer, is returned.	

Related topics

- **Initializing the user-programmable IP interface** (see page 34)
- **Establishing a connection** (see page 35)
- **Terminating a connection** (see page 44)
- **Sending data** (see page 39)

Terminating a connection

Introduction

Clear all connections which are no longer required as the number of concurrently opened connections is limited.

Function declaration

```
Function ConnectionDelete(IPConnection:Int):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Comment
`IPConnection`	Handle	Return value of the function `ConnectionCreate()`

Return value

The following return values are possible:

Return value	
0	Connection terminated and deleted
-1	Invalid handle

How to use this function

This way, you can invoke the function and assign its return value to a variable for further utilization:

```
Result := ConnectionDelete(hConnection);
```

Related topics

- **Establishing a connection** (see page 35)
- **Sending data** (see page 39)
- **Receiving data** (see page 41)
- **Initializing the user-programmable IP interface** (see page 34)

2.2 Registers

Introduction

This chapter describes the registers of the device which contain the current connection list of the user-programmable IP interface. These registers can be used for debugging or diagnostic purposes. However, they can't be used for other functions such as establishing or terminating a connection.

Contents

Register numbers

Introduction

Data of one connection each are displayed within the registers of a coherent register block. The basic register number of this block is dependent on the controller.

Register numbers

Basic register number	Register numbers
350000	350000 ... 350007

Determining the register number

In this chapter only the last figure of a register number is specified, for example MR 1. To calculate the complete register number, add the basic register number of the corresponding device to this figure, e.g. 350000.

Registers - Overview

Register	Description
MR 0	Selecting a connection
MR 1	Type of connection
MR 2	Transport protocol
MR 3	IP address
MR 4	IP port number
MR 5	State
MR 6	Number of sent bytes
MR 7	Number of received bytes
MR 8	Number of discarded bytes
MR 9	Number of discarded packets

Registers - Description

Introduction

The operating system manages the established connections in a list. Module register MR 0 *Selection of a connection* is used to copy connection details into other registers of a register block.

MR 0

Selecting a connection

Connections are selected by writing values to this register. This register is used to display whether the following registers contain usage data.

Module register properties		
Reading values	0	Connection exists
	-1	Connection does not exist

Module register properties		
Writing values	0	Address the first connection in the list
	> 0	Address the next connection in the list
	< 0	Address the previous connection in the list

MR 1

Type of connection

The value in this register shows whether the connection is a client or a server connection.

Module register properties		
Values	1	Client
	2	Server

MR 2

Transport protocol

The value in this register shows whether TCP or UDP is used as transport protocol.

Module register properties		
Values	1	UDP
	2	TCP

MR 3

IP address

The value in this register shows the configured IP address.

Module register properties	
Values	0.0.0.0 ... 255.255.255.255

MR 4

IP port number

The value in this register shows the configured IP port number.

Module register properties	
Values	0 ... 65.535

MR 5

Indication

The value in this register shows status the connection is currently in.

Module register properties		
Values	0	Connection terminated
	1	Connection is being established
	2	Connection is established
	3	TCP/IP server: Waiting for connection request from client
	4	Internal usage

MR 6

Number of sent bytes

The value in this register shows the number of data bytes sent via the given connection. Since this is a signed 32-bit register and the sent bytes are added each time, the number range may be exceeded from the positive maximum value to the negative maximum value.

Module register properties	
Values	-2.147.483.648 ... 2.147.483.647

MR 7

Number of received bytes

The value in this register shows the number of data bytes received via the given connection. Since this is a signed 32-bit register and the received bytes are added each time, the number range may be exceeded from the positive maximum value to the negative maximum value.

Module register properties	
Values	-2.147.483.648 ... 2.147.483.647

MR 8

Number of discarded bytes

The value in this register indicates the data bytes which could not be received, because the application program had not taken the cached data bytes.

Module register properties	
Values	0 ... 2.147.483.647

MR 9

Number of discarded packets

The value in this register indicates the data packets which could not be received, because the application program had not taken the cached data packages.

Module register properties	
Values	0 ... 2.147.483.647

3 User-programmable CAN-Prim interface

CAN-Prim interface

The user-programmable CAN-Prim interface allows to send and receive CAN messages. The CAN messages are completely processed in the application program.

CAN-Prim - The benefit

This feature is not only apt for CANopen® devices. The customer can rather communicate with third-party devices which are based on a CAN protocol.

Applications

The user-programmable CAN-Prim interface can be used for the following applications:

- Devices which are equipped with a CAN interface can be controlled via proprietary protocols.
- Controlling of CANopen® capable devices
- ...

Required programmer's skills

To be able to program user-programmable CAN-Prim interfaces basic knowledge of Controller Area Networks (CAN) is required. These are some of them:

- Structure of CAN messages
- CANopen® features

Hardware prerequisites

As a hardware, a JetControl is required which has got a CAN interface and/or a JX2 system bus.

Register numbers for JC-3xx

A JC-3xx register number consists of the following elements:

2	0	0	0	z	z	z	z	z

Element	Meaning	Value range
zzzzz	Module register number	2029, 2077 10500 ...10599

Register numbers for JC-9xx

Register numbers for the submodule JX6-SB(-I) connected to a JC-9xx consist of the following elements:

2	0	S	J	z	z	z	z	z

Element	Meaning	Value range
S	Number of the riser card	1 ... 3
J	Number of the submodule JX6-SB-I located on the riser card	1 ... 2
zzzzz	Module register number	2029, 2077 10500 ...10599

3 User-programmable CAN-Prim interface

Contents

Restrictions regarding the CAN-Prim interface

Restrictions regarding connectable modules

When using the user-programmable CAN-Prim interface, the following restrictions apply:

- If 29-bit CAN identifiers are used, the serial number of non-intelligent JX2-I/O module must start with 2.

CAN messages during boot phase

Between launching the device and starting the application program (boot phase of the JX2 system bus), the connected CAN modules are not permitted to send any CAN messages.

Time response

The interval between two CAN messages received via CAN interface should be at least 10 ms. In case of shorter time intervals, the device is not able to process all CAN messages for CAN-Prim reception.

If several CAN messages of the same CAN-ID are to be received, an application program featuring a high reaction and processing speed is required to prevent buffer overflows (overrun-bit). Adjusting the task switch procedure and task prioritization (TASKPRIORITY) do not necessarily grant processing all CAN messages.

Earmarked CAN-IDs

When peripheral modules are simultaneously operated on the JX2 system bus and the CAN-Prim interface, certain CAN-IDs are earmarked.

Modules on the JX2 system bus	Earmarked CAN-IDs
For all modules	0x100, 0x701, 0x702, 0x703, 0x704, 0x705, 0x706, 0x707, 0x708, 0x709, 0x70A, 0x732, 0x733, 0x734, 0x735, 0x736, 0x737, 0x738, 0x739, 0x73A, 0x73B, 0x746, 0x747, 0x748, 0x749, 0x74A, 0x74B, 0x74C, 0x74D, 0x74E, 0x74F
JX2-I/O modules	0x180, 0x181, 0x182, 0x183, 0x184, 0x185, 0x186, 0x187, 0x188, 0x189, 0x18A, 0x18B, 0x18C, 0x18D, 0x18E, 0x18F, 0x190, 0x191, 0x192, 0x193, 0x194, 0x195, 0x196, 0x197, 0x198, 0x199, 0x19A, 0x19B, 0x19C, 0x19D, 0x19E, 0x19F, 0x1A0, 0x1A1, 0x1A2, 0x1A3, 0x1A4, 0x1A5, 0x1A6, 0x1A7, 0x1A8, 0x1A9, 0x1AA, 0x1AB, 0x1AC, 0x1AD, 0x1AE, 0x1AF, 0x1B0, 0x1B1, 0x1B2, 0x1B3, 0x1B4, 0x1B5, 0x1B6, 0x1B7, 0x1B8, 0x1B9, 0x1BA, 0x1BB, 0x1BC, 0x1BD, 0x1BE, 0x1BF, 0x380, 0x381, 0x382, 0x383, 0x384, 0x385, 0x386, 0x387, 0x388, 0x389, 0x38A, 0x38B, 0x38C, 0x38D, 0x38E, 0x38F, 0x390, 0x391, 0x392, 0x393, 0x394, 0x395, 0x396, 0x397, 0x398, 0x399, 0x39A, 0x39B, 0x39C, 0x39D, 0x39E, 0x39F, 0x3A0, 0x3A1, 0x3A2, 0x3A3, 0x3A4, 0x3A5, 0x3A6, 0x3A7, 0x3A8, 0x3A9, 0x3AA, 0x3AB, 0x3AC, 0x3AD, 0x3AE, 0x3AF, 0x3B0, 0x3B1, 0x3B2, 0x3B3, 0x3B4, 0x3B5, 0x3B6, 0x3B7, 0x3B8, 0x3B9, 0x3BA, 0x3BB, 0x3BE, 0x3BF

Modules on the JX2 system bus	Earmarked CAN-IDs
JX2 slave modules	0x081, 0x082, 0x083, 0x084, 0x085, 0x086, 0x087, 0x088, 0x089, 0x08A, 0x08B, 0x08C, 0x08D, 0x08E, 0x08F, 0x090, 0x09F, 0x0A0, 0x0A1, 0x0A2, 0x0A3, 0x0A4, 0x0A5, 0x0A6, 0x0A7, 0x0A8, 0x0A9, 0x0AA, 0x0AB, 0x0AC, 0x0AD, 0x0AE, 0x0AF, 0x161, 0x162, 0x163, 0x164, 0x165, 0x166, 0x167, 0x168, 0x169, 0x16A, 0x16B, 0x16C, 0x16D, 0x16E, 0x16F, 0x1D1, 0x1D2, 0x1D3, 0x1D4, 0x1D5, 0x1D6, 0x1D7, 0x1D8, 0x1D9, 0x1DA, 0x1DB, 0x1DC,0x1DD, 0x1DE, 0x1DF
JX3 modules	0x180, 0x181, 0x182, 0x183, 0x184, 0x185, 0x186, 0x187, 0x188, 0x189, 0x18A, 0x18B, 0x18C, 0x18D, 0x18E, 0x18F, 0x190, 0x191, 0x192, 0x193, 0x194, 0x195, 0x196, 0x197, 0x198, 0x199, 0x19A, 0x19B, 0x19C, 0x19D, 0x19E, 0x19F, 0x1A0, 0x1A1, 0x1A2, 0x1A3, 0x1A4, 0x1A5, 0x1A6, 0x1A7, 0x1A8, 0x1A9, 0x1AA, 0x1AB, 0x1AC, 0x1AD, 0x1AE, 0x1AF, 0x1B0, 0x1B1, 0x1B2, 0x1B3, 0x1B4, 0x1B5, 0x1B6, 0x1B7, 0x1B8, 0x1B9, 0x1BA, 0x1BB, 0x1BC, 0x1BD, 0x1BE, 0x1BF, 0x320, 0x321, 0x322, 0x323, 0x324, 0x325, 0x326, 0x327, 0x328, 0x329, 0x32A, 0x32B, 0x32C, 0x32D, 0x32E, 0x32F, 0x330, 0x331, 0x332, 0x333, 0x334, 0x335, 0x336, 0x337, 0x338, 0x339, 0x33A, 0x33B, 0x33C, 0x33D, 0x33E, 0x380, 0x381, 0x382, 0x383, 0x384, 0x385, 0x386, 0x387, 0x388, 0x389, 0x38A, 0x38B, 0x38C, 0x38D, 0x38E, 0x38F, 0x390, 0x391, 0x392, 0x393, 0x394, 0x395, 0x396, 0x397, 0x398, 0x399, 0x39A, 0x39B, 0x39C, 0x39D, 0x39E, 0x39F, 0x3A0, 0x3A1, 0x3A2, 0x3A3, 0x3A4, 0x3A5, 0x3A6, 0x3A7, 0x3A8, 0x3A9, 0x3AA, 0x3AB, 0x3AC, 0x3AD, 0x3AE, 0x3AF, 0x3B0, 0x3B1, 0x3B2, 0x3B3, 0x3B4, 0x3B5, 0x3B6, 0x3B7, 0x3B8, 0x3B9, 0x3BA, 0x3BB, 0x3BE, 0x3BF, 0x3E0, 0x3E1, 0x3E2, 0x3E3, 0x3E4, 0x3E5, 0x3E6, 0x3E7, 0x3E8, 0x3E9, 0x3EA, 0x3EB, 0x3EC, 0x3ED, 0x3EE, 0x3EF, 0x3F0, 0x3F1, 0x3F2, 0x3F3, 0x3F4, 0x3F5, 0x3F6, 0x3F7, 0x3F8, 0x3F9, 0x3FA, 0x3FB, 0x3FC, 0x3FD, 0x3FE

Modules on the JX2 system bus	Earmarked CAN-IDs
JX-SIO and CANopen® modules	0x1C6, 0x1C7, 0x1C8, 0x1C9, 0x1CA, 0x1CB, 0x1CC, 0x1CD, 0x1CE, 0x1CF, 0x246, 0x247, 0x248, 0x249, 0x24A, 0x24B, 0x24C, 0x24D, 0x24E, 0x24F, 0x2C6, 0x2C7, 0x2C8, 0x2C9, 0x2CA, 0x2CB, 0x2CC, 0x2CD, 0x2CE, 0x2CF, 0x346, 0x347, 0x348, 0x349, 0x34A, 0x34B, 0x34C, 0x34D, 0x34E, 0x34F, 0x3C6, 0x3C7, 0x3C8, 0x3C9, 0x3CA, 0x3CB, 0x3CC, 0x3CD, 0x3CE, 0x3CF, 0x446, 0x447, 0x448, 0x449, 0x44A, 0x44B, 0x44C, 0x44D, 0x44E, 0x44F, 0x4C6, 0x4C7, 0x4C8, 0x4C9, 0x4CA, 0x4CB, 0x4CC, 0x4CD, 0x3CE, 0x4CF, 0x581, 0x582, 0x583, 0x584, 0x585, 0x586, 0x587, 0x588, 0x589, 0x58A, 0x5B2, 0x5B3, 0x5B4, 0x5B5, 0x5B6, 0x5B7, 0x5B8, 0x5B9, 0x5BA, 0x5BB, 0x5C6, 0x5C7, 0x5C8, 0x5C9, 0x5CA, 0x5CB, 0x5CC, 0x5CD, 0x5CE, 0x5CF, 0x601, 0x602, 0x603, 0x604, 0x605, 0x606, 0x607, 0x608, 0x609, 0x60A, 0x632, 0x633, 0x634, 0x635, 0x636, 0x637, 0x638, 0x639, 0x63A, 0x63B, 0x646, 0x647, 0x648, 0x649, 0x64A, 0x64B, 0x64C, 0x64D, 0x64E, 0x64F, 0x732, 0x733, 0x734, 0x735, 0x736, 0x737, 0x738, 0x739, 0x73A, 0x73B, 0x746, 0x747, 0x748, 0x749, 0x74A, 0x74B, 0x74C, 0x74D, 0x74E, 0x74F
Festo CP-FB modules	0x010, 0x110, 0x120, 0x130, 0x140, 0x150, 0x1E0, 0x1F0, 0x250, 0x260, 0x270, 0x350, 0x360, 0x370, 0x3B0
LioN-S modules	0x2E0, 0x2E1, 0x2E2, 0x2E3, 0x2E4, 0x2E5, 0x2E6, 0x2E7, 0x2E8, 0x2E9, 0x2EA, 0x2EB, 0x2EC, 0x2ED, 0x2EE, 0x2EF, 0x2F0, 0x2F1, 0x2F2, 0x2F3, 0x2F4, 0x2F5, 0x2F6, 0x2F7, 0x2F8, 0x2F9, 0x2FA, 0x2FB, 0x2FC, 0x2FD, 0x2FE, 0x360, 0x361, 0x362, 0x363, 0x364, 0x365, 0x366, 0x367, 0x368, 0x369, 0x36A, 0x36B, 0x36C, 0x36D, 0x36E, 0x36F, 0x370, 0x371, 0x372, 0x373, 0x374, 0x375, 0x376, 0x377, 0x378, 0x379, 0x37A, 0x37B, 0x37C, 0x37D, 0x37E, 0x581, 0x582, 0x583, 0x584, 0x585, 0x586, 0x587, 0x588, 0x589, 0x58A, 0x58B, 0x58C, 0x58D, 0x58E, 0x58F, 0x590, 0x591, 0x592, 0x593, 0x594, 0x595, 0x596, 0x597, 0x598, 0x599, 0x59A, 0x59B, 0x59C, 0x59D, 0x59E, 0x59F, 0x5A0, 0x601, 0x602, 0x603, 0x604, 0x605, 0x606, 0x607, 0x608, 0x609, 0x60A, 0x60B, 0x60C, 0x60D, 0x60E, 0x60F, 0x610, 0x611, 0x612, 0x613, 0x614, 0x615, 0x616, 0x617, 0x618, 0x619, 0x61A, 0x61B, 0x61C, 0x61D, 0x61E, 0x61F, 0x620, 0x701, 0x702, 0x703, 0x704, 0x705, 0x706, 0x707, 0x708, 0x709, 0x70A, 0x70B, 0x70C, 0x70D, 0x70E, 0x70F, 0x710, 0x711, 0x712, 0x713, 0x714, 0x715, 0x716, 0x717, 0x718, 0x719, 0x71A, 0x71B, 0x71C, 0x71D, 0x71E, 0x71F, 0x720

Modules on the JX2 system bus	Earmarked CAN-IDs
BWU1821	0x281, 0x282, 0x283, 0x284, 0x285, 0x286, 0x287, 0x288, 0x289, 0x28A, 0x28B, 0x28C, 0x28D, 0x28E, 0x28F, 0x290, 0x291, 0x292, 0x293, 0x294, 0x295, 0x296, 0x297, 0x298, 0x299, 0x29A, 0x29B, 0x29C, 0x29D, 0x29E, 0x29F, 0x301, 0x302, 0x303, 0x304, 0x305, 0x306, 0x307, 0x308, 0x309, 0x30A, 0x30B, 0x30C, 0x30D, 0x30E, 0x30F, 0x310, 0x311, 0x312, 0x313, 0x314, 0x315, 0x316, 0x317, 0x318, 0x319, 0x31A, 0x31B, 0x31C, 0x31D, 0x31E, 0x31F, 0x481, 0x482, 0x483, 0x484, 0x485, 0x486, 0x487, 0x488, 0x489, 0x48A, 0x48B, 0x48C, 0x48D, 0x48E, 0x48F, 0x490, 0x491, 0x492, 0x493, 0x494, 0x495, 0x496, 0x497, 0x498, 0x499, 0x49A, 0x49B, 0x49C, 0x49D, 0x49E, 0x49F, 0x501, 0x502, 0x503, 0x504, 0x505, 0x506, 0x507, 0x508, 0x509, 0x50A, 0x50B, 0x50C, 0x50D, 0x50E, 0x50F, 0x510, 0x511, 0x512, 0x513, 0x514, 0x515, 0x516, 0x517, 0x518, 0x519, 0x51A, 0x51B, 0x51C, 0x51D, 0x51E, 0x51F, 0x5C6, 0x5C7, 0x5C8, 0x5C9, 0x5CA, 0x5CB, 0x5CC, 0x5CD, 0x5CE, 0x5CF, 0x646, 0x647, 0x648, 0x649, 0x64A, 0x64B, 0x64C, 0x64D, 0x64E, 0x64F, 0x746, 0x747, 0x748, 0x749, 0x74A, 0x74B, 0x74C, 0x74D, 0x74E, 0x74F
LJX7-CSL	0x481, 0x482, 0x483, 0x484, 0x485, 0x486, 0x487, 0x488, 0x489, 0x48A, 0x48B, 0x48C, 0x48D, 0x48E, 0x48F, 0x490, 0x491, 0x492, 0x493, 0x494, 0x495, 0x496, 0x497, 0x498, 0x499, 0x49A, 0x49B, 0x49C, 0x49D, 0x49E, 0x49F, 0x501, 0x502, 0x503, 0x504, 0x505, 0x506, 0x507, 0x508, 0x509, 0x50A, 0x50B, 0x50C, 0x50D, 0x50E, 0x50F, 0x510, 0x511, 0x512, 0x513, 0x514, 0x515, 0x516, 0x517, 0x518, 0x519, 0x51A, 0x51B, 0x51C, 0x51D, 0x51E, 0x51F, 0x581, 0x582, 0x583, 0x584, 0x585, 0x586, 0x587, 0x588, 0x589, 0x58A, 0x58B, 0x58C, 0x58D, 0x58E, 0x58F, 0x590, 0x591, 0x592, 0x593, 0x594, 0x595, 0x596, 0x597, 0x598, 0x599, 0x59A, 0x59B, 0x59C, 0x59D, 0x59E, 0x59F, 0x5A0, 0x601, 0x602, 0x603, 0x604, 0x605, 0x606, 0x607, 0x608, 0x609, 0x60A, 0x60B, 0x60C, 0x60D, 0x60E, 0x60F, 0x610, 0x611, 0x612, 0x613, 0x614, 0x615, 0x616, 0x617, 0x618, 0x619, 0x61A, 0x61B, 0x61C, 0x61D, 0x61E, 0x61F, 0x620, 0x701, 0x702, 0x703, 0x704, 0x705, 0x706, 0x707, 0x708, 0x709, 0x70A, 0x70B, 0x70C, 0x70D, 0x70E, 0x70F, 0x710, 0x711, 0x712, 0x713, 0x714, 0x715, 0x716, 0x717, 0x718, 0x719, 0x71A, 0x71B, 0x71C, 0x71D, 0x71E, 0x71F, 0x720

User-programmable CAN-Prim interface - Operating principle

Funktion

The user-programmable CAN-Prim interface uses message boxes for data exchange between CAN bus and application program. Each message box is able to accommodate a complete CAN message.

16 message boxes are available to the user. Each of these boxes can be configured either as inbox or as outbox with a specific CAN-ID.

Technical specifications

Function	Description
CAN-ID	11-bit or 29-bit
Number of message boxes	16

Enabling the user-programmable CAN-Prim interface

The CAN-Prim interface is enabled via Bits in MR 2077 *JX2-system bus - special functions Register description MR 2077***.** (see page 59)

Internal processes of the CAN-Prim interface

Introduction

The CAN-Prim interface processes the following tasks independently:

- Sending of CAN messages
- Reception of CAN messages
- Filtering of CAN messages on reception

Internal reception of CAN messages

The CAN-Prim interface receives new CAN messages in the following way:

<table>
<tr><th>Step</th><th colspan="2">Description</th></tr>
<tr><td>1</td><td colspan="2">The CAN bus receives a valid CAN message.</td></tr>
<tr><td>2</td><td colspan="2">The CAN-ID matches the receiving mask.</td></tr>
<tr><td>3</td><td colspan="2">The CAN-ID matches the CAN-ID of a message box which has been configured as inbox.</td></tr>
<tr><td rowspan="3">4</td><th>If in MR 10530 + message box number*20 of the message box ...</th><th>... then ...</th></tr>
<tr><td>... bit 1 *NEW-DAT* = 0,</td><td>... bit 1 *NEW-DAT* becomes = 1;
proceed with step 5</td></tr>
<tr><td>... bit 1 *NEW-DAT* = 1,</td><td>... bit 2 *OVERRUN* becomes = 1;
CAN message data are discarded.</td></tr>
<tr><td>5</td><td colspan="2">The value of MR 10503 *FIFO Occupancy* is increased by one.
This register shows whether new CAN messages have been received, as well as the number of messages.</td></tr>
<tr><td>6</td><td colspan="2">The message box number is entered into MR 10504 *FIFO data*.
This register shows which of the messages boxes has received a new CAN message.</td></tr>
<tr><td>7</td><td colspan="2">In MR 10500 *CAN-Prim Status*, bit 1 *NEW DAT* = 1.</td></tr>
</table>

Register description - CAN-Prim interface

Registers for configuring the JX2 system bus

The CAN-Prim interface is enabled in MR 2077 *JX2-system bus special functions*.

Register	Description
MR 2029	JX2 system bus - Baud rate
MR 2077	JX2 system bus special functions

Registers for configuring the CAN-Prim interface

Registers	Description
MR 10500	CAN-Prim status register
MR 10501	CAN-Prim command register
MR 10503	FIFO occupancy - Number of received messages
MR 10504	FIFO data - Numbers of message boxes which have received new messages
MR 10506	Global reception mask
MR 10507	Global reception ID

CANopen®-STX-API has not been implemented:

MR 2077

Enabling JX2 system bus special functions

The value of this register influences the behavior at initializing the JX2 system bus.

Meaning of the individual bits

Bit 2 **Activate CAN-Prim in addition to JX2 system bus**

1 = The CAN-Prim interface and the JX2 system bus are enabled following the next launch of the JX2 system bus. This requires a restart of the controller.

This function lets you connect JX2 expansion modules.

Bit 3 **Enable CAN-Prim only**

1 = Only the CAN-Prim interface is enabled following the next launch of the JX2 system bus. This requires a restart of the controller.

All node IDs can be used **without any** restrictions.

The controller does not initialize any JX2 expansion modules on the JX2 system bus. For this reason, JX2 expansion modules **cannot** be connected.

Bit 4 **CAN IDs 0x081 ... 9x09F for CAN-Prim**

1 = The CAN-Prim interface allows communication with the CAN IDs 0x081 ... 0x09F.

Generally, master-slave operations with JX2 slave modules and MC axes are executed via these CAN IDs, .

Module register properties	
Value after reset	Remanent; factory setting: 0
Takes effect	Next time when the controller is launched

CANopen®-STX-API has been implemented:

MR 2077

Enabling JX2 system bus special functions

The value of this register influences the behavior at initializing of the JX2 system bus (CAN 1).

Meaning of the individual bits		
Bit 3, bit 2	**Activate CAN-Prim in addition to JX2 system bus**	
	01 =	The CAN-Prim interface and the JX2 system bus are enabled following the next launch of the JX2 system bus. This requires a restart of the controller. This function lets you connect JX2 expansion modules.
Bit 3, bit 2	**Enable CAN-Prim and CANopen® STX API only.**	
	1x =	At the next restart, the JX2 system bus is not initialized. The CAN-Prim interface can be used. All node IDs can be used **without any** restrictions. The controller does not initialize any JX2 expansion modules on the JX2 system bus. For this reason, JX2 expansion modules **cannot** be connected. The CANopen® STX API can be used.
Bit 4	**CAN IDs 0x081 ... 9x09F for CAN-Prim**	
	1 =	The CAN-Prim interface allows communication with the CAN IDs 0x081 ... 0x09F. Generally, master-slave operations with JX2 slave modules and MC axes are executed via these CAN IDs, .

Module register properties	
Value after reset	Remanent; factory setting: 0
Takes effect	Next time when the controller is launched

MR 10500

CAN-Prim status register

Via MR 10500, the status of the CAN-Prim interface can be evaluated.

Meaning of the individual bits		
Bit 1	**NEW-DAT**	
	1 =	At least one message box has received a new CAN message.

Meaning of the individual bits		
Bit 2	**Length of CAN ID**	
	0 =	The length of sent/received CAN IDs is 11 bits
	1 =	The length of sent/received CAN IDs is 29 bits

Module register properties	
Type of access	Read access
Takes effect	When the CAN-Prim interface is enabled

MR 10501

CAN-Prim command register

Via MR 10501, certain commands are transmitted to the CAN-Prim interface.

These commands apply for both direct and indirect access to the CAN message box.

CAN-Prim interface - Commands	
7	**Clearing the FIFO buffer** This command is for clearing all entries in the FIFO buffer. Result: MR 10503 = 0
8	**Set the standard ID length to 11 bits** The ID length for all CAN messages is set to 11 bits. Result: Bit 2 = 0 in MR 10500 MR 10506 := 0 MR 10507 := 0 MR 10542 + message box number *20 := 0x7FF (in all message boxes)
9	**Set the standard ID length to 29 bits** The ID length for all CAN messages is set to 29 bits. Result: Bit 2 = 1 in MR 10500 MR 10506 := 0 MR 10507 := 0 MR 10542 + message box number*20 := 0x7FFFFFFF (in all message boxes)
10	**Check message boxes for receiving new messages** The CAN-Prim interface automatically checks the inbox for new CAN messages. Command 10 forces manual checking of pending messages. By now, issuing command 10 is not obligatory any more.

Module register properties	
Takes effect	When the CAN-Prim interface is enabled

MR 10503

FIFO buffer occupancy

MR 10503 shows if further CAN messages have been received, as well as the number of messages.

Subtracting the number read first from the number read next renders the number of new messages received.

Module register properties		
Values	Number of received messages:	0 ... 16
Type of access	Read access	
Takes effect	When the CAN-Prim interface is enabled	

MR 10504

FIFO data

MR 10504 shows which of the messages boxes has received the latest new CAN message. At reading MR 10504, the Fifo is cleared from the value just read. Accordingly, the value of MR 10503 is incremented by one.

Notice!

Each read access to this register, even via an active JetSym setup screen, decrements the number of received CAN messages.

Module register properties		
Values	No FIFO data available: Number of the message box containing new data:	-1 0 ... 15
Type of access	Read access removes characters	
Value after reset	-1	
Takes effect	When the CAN-Prim interface is enabled	

MR 10506

Global reception mask

The global receiving mask is for filtering the bits of the received CAN IDs. If the bit of the global reception mask is set, the received bit of the CAN ID is compared with the global reception ID as shown in MR 10507.

Module register properties		
Values	In the case of 11-bit CAN IDs	0 ... 0x7FF
	In the case of 29-bit CAN IDs	0 ... 0x1FFFFFFF
Bit = 0	The bit is not compared with MR 10507	
Bit = 1	The bit is compared with MR 10507	
Takes effect	When the CAN-Prim interface is enabled	

MR 10507

Global reception ID

By means of the global reception ID and MR 10506 *Global Reception Mask* a range of CAN IDs is set, which is transmitted to the CAN-Prim interface.

Module register properties		
Values	In the case of 11-bit CAN IDs	0 ... 0x7FF
	In the case of 29-bit CAN IDs	0 ... 0x1FFFFFFF
Takes effect	When the CAN-Prim interface is enabled	

CAN message box - Description of registers for direct access

Submodule JX6-SB(-I)

Direct access to the message box via the JX6-SB(-I) submodule is not possible.

Direct access

For programming purposes, always use registers for direct access to message boxes. 20 registers with identical functions are assigned to each message box. The registers of individual message boxes start from a certain basic register number.

Message box number	Module register number
0	MR 10530
1	MR 10550
2	MR 10570
3	MR 10590
4	MR 10610
5	MR 10630
6	MR 10650
7	MR 10670
8	MR 10690
9	MR 10710
10	MR 10730
11	MR 10750
12	MR 10770
13	MR 10790
14	MR 10810
15	MR 10830

Registers for message boxes of the CAN-Prim interface

20 registers with identical functions are assigned to each message box. The register number of individual message boxes is calculated from the basic register number and the message box number (0 ... 15).

Module register	Description
MR 10530 + message box number*20	Message box status register
MR 10531 + message box number*20	Message box configuration register
MR 10532 + message box number*20	CAN-ID
MR 10533 + message box number*20	Number of data bytes
MR 10534 + message box number*20	Data byte 0

Module register	Description
MR 10535 + message box number*20	Data byte 1
MR 10536 + message box number*20	Data byte 2
MR 10537 + message box number*20	Data byte 3
MR 10538 + message box number*20	Data byte 4
MR 10539 + message box number*20	Data byte 5
MR 10540 + message box number*20	Data byte 6
MR 10541 + message box number*20	Data byte 7
MR 10542 + message box number*20	CAN-ID mask
MR 10543 + message box number*20	Box command register
MR 10544 + message box number*20	Received CAN-ID
MR 10545 + message box number*20	Not used
MR 10546 + message box number*20	Not used
MR 10547 + message box number*20	Not used
MR 10548 + message box number*20	Not used
MR 10549 + message box number*20	Not used

MR 10530 + message box number*20

Message box status register

This register shows the status of the message box.

Meaning of the individual bits

Bit 0 **Valid**

1 = The message box is enabled

Bit 1 **NEW-DAT**

1 = The message box has received a CAN message. Reception of additional CAN messages is blocked.

Bit 2 **OVERRUN**

1 = A new CAN message for this message box was being received while bit 1 *NEW-DAT* was = 1.

The new message is discarded.

Meaning of the individual bits		
Bit 3	**Sending error**	
	1 =	An error has occurred when sending a CAN message from this message box.

Module register properties	
Type of access	Read access
Takes effect	When the CAN-Prim interface is enabled

MR 10543 + message box number*20

Box command register

R 200010543 + message box number *20 is used to transfer certain commands to the message box.

CAN-Prim interface - Commands	
1	**Enabling the message box** The message box is enabled. When enabling the message box, the system checks whether the CAN-ID of the message box has been reserved by the JX2 system bus or not. **Result, if the CAN-ID has not been reserved:** Bit 0 = 1 in MR 10530 + message box number*20
2	**Disabling the message box** The message box is disabled. **Result:** Bit 0 = 0 in MR 10530 + message box number*20
3	**Sending CAN messages** A CAN message is sent.
4	**Clearing the NEW DAT bit** Clears bit 1 *NEW-DAT* in MR 10530 + message box number*20. The message box is able to receive CAN messages again. **Result:** Bit 1 = 0 in MR 10530 + message box number*20 If for all message boxes the NEW DAT bit is 0, bit 1 = 0 in MR 10500.
5	**Clearing the OVERRUN bit** Clears bit 2 *OVERRUN* in MR 10530 + message box number*20 of the message box. **Result:** Bit 2 = 0 in MR 10530 + message box number*20
6	**Clearing the sending error bit** Clears bit 3 *Sending error* in MR 10530 + message box number*20 of the message box. **Result:** Bit 3 = 0 in MR 10530 + message box number*20

Module register properties	
Takes effect	When the CAN-Prim interface is enabled

MR 10531 + message box number*20

Message box configuration register

MR 10531 + message box number*20 is for configuring the message box.

Configuration values	
0	**Inbox** For configuring the message box as inbox
1	**Outbox** For configuring the message box as outbox for standard frames
2	**Outbox RTR** For configuring the message box as outbox for RTR frames

Module register properties	
Takes effect	When the CAN-Prim interface is enabled

MR 10532 + message box number*20

CAN-ID

In the case of an outbox, a CAN message is sent using the CAN-ID.

In the case of an inbox, CAN messages with this CAN-ID - which is masked by the CAN-ID mask - are received.

Module register properties		
Values	In the case of 11-bit CAN IDs	0 ... 0x7FF
	In the case of 29-bit CAN IDs	0 ... 0x1FFFFFFF
Takes effect	When the CAN-Prim interface is enabled and the message box is disabled, i.e. if bit 0 = 0 in MR 10530 + message box number*20	

MR 10542 + message box number*20

CAN-ID mask

The CAN-ID mask can be used to configure which bits of a received CAN-ID are compared with the configured CAN-ID of the message box.

Module register properties		
Values	Bit = 0	Bit is not compared with CAN-ID
	Bit = 1	Bit is compared with CAN-ID
Takes effect	When the CAN-Prim interface is enabled	

MR 10544 + message box number*20

Received CAN-ID

In the case of an inbox, the CAN-IDs of received CAN messages are entered here.

Module register properties		
Type of access	Read access	
Values	In the case of 11-bit CAN IDs	0 ... 0x7FF
	In the case of 29-bit CAN IDs	0 ... 0x1FFFFFFF
Takes effect	When the CAN-Prim interface is enabled	

MR 10533 + message box number*20

Number of data bytes

In the case of an outbox, a CAN message is sent with this number of data bytes.

In the case of an inbox, the number of received data bytes is entered.

Module register properties		
Values	Number of data bytes:	0 ... 8
Takes effect	When the CAN-Prim interface is enabled	

MR 10534 ... MR 10541 + message box number*20

Data bytes 0 through 7

In the case of an outbox, a CAN message is sent with these data bytes.
In the case of an inbox, the received data bytes are entered.

Module register properties		
Values	Data of data bytes:	0 ... 255
Takes effect	When the CAN-Prim interface is enabled	

CAN message box - Description of registers for indirect access

Indirect access

To get indirect access to message boxes of the CAN-Prim interface, always select the message box using MR 10502 *Message Box Number*.

To allow compatibility with previous OS versions the registers for indirect access are still supported. **Always use the registers for direct access when programming the CAN-Prim interface.**

MR 10501

CAN-Prim command register

Via MR 10501, certain commands are transmitted to the CAN-Prim interface.

CAN-Prim interface - Commands	
1	**Enabling the message box** The message box selected via MR 10502 is enabled. When enabling the message box, the system checks whether the CAN-ID of the message box has been reserved by the system bus or not. **Result:** Bit 0 = 1 in MR 10510
2	**Disabling the message box** The message box selected via MR 10502 is disabled. **Result:** Bit 0 = 0 in MR 10510
3	**Sending CAN messages** A CAN message is sent containing the data of the selected message box.
4	**Clearing the NEW DAT bit** This command is for clearing bit 1 *NEW-DAT* in MR 10510 which enables the selected message box to receive CAN messages again. **Result:** Bit 1 = 0 in MR 10510
5	**Clearing the OVERRUN bit** Clears bit 2 *OVERRUN* in MR 10510 of the selected message box. **Result:** Bit 2 = 0 in MR 10510
6	**Clearing the sending error bit** Clears bit 3 **Sending error** in MR 10510 of the selected message box. **Result:** Bit 3 = 0 in MR 10510
7	**Clearing the FIFO buffer** This command is for clearing all entries in the FIFO buffer. **Result:** MR 10503 = 0

CAN-Prim interface - Commands	
8	**Set the standard ID length to 11 bits** The ID length for all CAN messages is set to 11 bits. **Result:** Bit 2 = 0 in MR 10500 MR 10506 := 0 MR 10507 := 0
9	**Set the standard ID length to 29 bits** The ID length for all CAN messages is set to 29 bits. **Result:** Bit 2 = 1 in MR 10500 MR 10506 := 0 MR 10507 := 0
10	**Check message boxes for receiving new messages** The CAN-Prim interface automatically checks the inbox for new CAN messages. Command 10 forces manual checking of pending messages. By now, issuing command 10 is not obligatory any more.

Module register properties	
Takes effect	When the CAN-Prim interface is enabled

MR 10502

Message box number

Via MR 10502, a message box is selected. The data contained in the message box can then be accessed via MR 10510 to MR 10521.

Module register properties		
Values	Message box number:	0 ... 15
Takes effect	When the CAN-Prim interface is enabled	

MR 10510

Message box status register

This register shows the status of the message box.

Meaning of the individual bits		
Bit 0	**Valid**	
	1 =	The message box is enabled
Bit 1	**NEW-DAT**	
	1 =	The message box has received a CAN message. Reception of additional CAN messages is blocked.
Bit 2	**OVERRUN**	
	1 =	A new CAN message for this message box was being received while bit 1 *NEW-DAT* was = 1. The new message is discarded.
Bit 3	**Sending error**	
	1 =	An error has occurred when sending a CAN message from this message box.

Module register properties	
Type of access	Read access
Takes effect	When the CAN-Prim interface is enabled

MR 10511

Message box configuration register

MR 10511 is for configuring the message box.

Configuration values	
0	**Inbox**
	For configuring the message box as inbox
1	**Outbox**
	For configuring the message box as outbox for standard frames
2	**Outbox RTR**
	For configuring the message box as outbox for RTR frames

Module register properties	
Takes effect	When the CAN-Prim interface is enabled

MR 10512

CAN-ID

In the case of an outbox, a CAN message is sent using the CAN-ID.
In the case of an inbox, only CAN messages with this CAN-ID are received.

Module register properties		
Values	In the case of 11-bit CAN IDs	0 ... 0x7FF
	In the case of 29-bit CAN IDs	0 ... 0x1FFFFFFF
Takes effect	When a CAN-Prim interface is enabled and the message box is disabled, i.e. if bit 0 = 0 in MR 10510.	

MR 10513

Number of data bytes

In the case of an outbox, a CAN message is sent with this number of data bytes.
In the case of an inbox, the number of received data bytes is entered.

Module register properties		
Values	Number of data bytes:	0 ... 8
Takes effect	When the CAN-Prim interface is enabled	

MR 10514 ... MR 10521

Data bytes 0 through 7

In the case of an outbox, a CAN message is sent with these data bytes.
In the case of an inbox, the received data bytes are entered.

Module register properties		
Values	Data of data bytes:	0 ... 255
Takes effect	When the CAN-Prim interface is enabled	

Using the CAN-Prim interface (direct access)

Initialization

To initialize the CAN-Prim interface proceed as follows:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td><b>1</b></td><td colspan="2">Set bit 2 = 1 in MR 2077 <i>JX2 system bus special functions</i>.</td></tr>
<tr><td><b>2</b></td><td colspan="2">Start up the JX2 system bus.</td></tr>
<tr><td rowspan="4"><b>3</b></td><td colspan="2">Configure the CAN ID length for all message boxes.</td></tr>
<tr><th>If the CAN ID length...</th><th>... then ...</th></tr>
<tr><td>... is 11 bits,</td><td>... MR 10501 := 8;</td></tr>
<tr><td>... is 29 bits,</td><td>... MR 10501 := 9;</td></tr>
</table>

Configuring a message box as outbox

To configure a message box as outbox, proceed as follows:

Step	Action
1	Select a message box. In this manual, message box 1 is used (MR 10550).
2	Configure message box 1 as outbox: MR 10551 := 1;
3	Configure the CAN ID for sending messages MR 10552 := CAN ID;
4	Activate message box 1: MR 10563 := 1; **Result if configuration has been successful:** Bit 0 = 1 in MR 10550

Sending a CAN message

To send a CAN message proceed as follows:

Step	Action
1	Select a message box. In this manual message box 1 is used.
2	Enter the number of data bytes to be sent: MR 10553 := Number of bytes;

3	Enter the content into the data bytes to be sent:
	MR 10554 := Data byte 0;
	MR 10555 := Data byte 1;
	...
	MR 10561 := Data byte 7;

4	Start transmission of the CAN message: MR 10563 := 3; **Result if sending was successful:** Bit 3 = 0 in MR 10550

Configuring a message box as inbox

To configure a message box for receiving messages proceed as follows:

Step	Action
1	Select a message box. In this manual, message box 0 is used (MR 10530).
2	Configure message box 0 as inbox: MR 10531 := 0;
3	Configure the CAN ID for receiving messages MR 10532 := CAN ID;
4	Activate message box 1: MR 10543 := 1; **Result if configuration has been successful:** Bit 0 = 1 in MR 10530

Receiving a CAN message

To receive a CAN message in message box 0, proceed as follows:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td rowspan="3"><b>1</b></td><td colspan="2">Check bit 1 <i>NEW-DAT</i> in MR 10500</td></tr>
<tr><td><b>If ...</b></td><td><b>... then ...</b></td></tr>
<tr><td>... bit 1 <i>NEW-DAT</i> = 1 in MR 10500,</td><td>... a CAN message has been received.
Proceed with step 2.</td></tr>
<tr><td><b>2</b></td><td colspan="2">Read the number of the message box which has received a new CAN message.
Message box number := MR 10504;</td></tr>
<tr><td rowspan="3"><b>3</b></td><td colspan="2">Check the message box for overflow.</td></tr>
<tr><td><b>If ...</b></td><td><b>... then ...</b></td></tr>
<tr><td>... bit 2 <i>OVERRUN</i> = 1
in MR 10530,</td><td>... an overflow has occurred.</td></tr>
<tr><td><b>4</b></td><td colspan="2">Read the number of received bytes
Number of bytes := MR 10533;</td></tr>
<tr><td><b>5</b></td><td colspan="2">Read the received bytes.
Data byte 0 := MR 10534;
Data byte 1 := MR 10535;
...
Data byte 7 := MR 10541;</td></tr>
<tr><td><b>6</b></td><td colspan="2">Acknowledge reception
MR 10543 := 4;
<b>Result if message has been received successfully:</b>
Bit 1 = 0 in MR 10530</td></tr>
</table>

Using CAN-ID masks

Introduction

Usually the CAN-Prim interface receives only CAN messages with a CAN-ID which matches the configured CAN-ID of the message box.

You can use a mask to expand CAN-IDs of a message box which are to be received. Each message box has got a CAN-ID and a CAN-ID mask of its own.

Functioning principle

If ...	... then ...
... bit = 0 in MR 10542 + message box number*20,	... the bit of the CAN-ID received is not evaluated.
... bit = 1 in MR 10542 + message box number*20,	... the bit of the CAN-ID received must match the configured CAN-ID.

RTR frames via CAN-Prim interface

RTR frames

RTR (Remote Transmission Request) frames are a type of message specific to CAN. Using an RTR frame a CAN node A can prompt another CAN node B to transmit a message. An RTR frame cannot be used to transmit user data. Node B is prompted to transmit a frame of the same CAN-ID and the corresponding data.

Submodule JX6-SB-I

Transmitting RTR frames via the JX6-SB(-I) submodule is not possible.

Configuration for transmitting and receiving RTR frames

Step	Action
1	Select any message box for transmitting RTR frames and another message box for receiving them. In this manual message box 0 is used for transmitting and message box 1 for receiving RTR frames.
2	Configure message box 0 as outbox for RTR frames: MR 10531 := 2;
3	Configure the CAN-ID of the RTR frame: MR 10532 := CAN ID;
4	Activate message box 0: MR 10543 := 1; **Result:** Bit 0 = 1 in MR 10530
5	Configure message box 1 as inbox for replies to an RTR frame: MR 10551 := 0;
6	Configure the CAN-ID of the RTR frame: MR 10552 := CAN ID;
7	Activate message box 1: MR 10563 := 1; **Result:** Bit 0 = 1 in MR 10550

Transmitting and receiving RTR frames

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Prompt transmitting an RTR frame from message box 0:
MR 10543 := 3;</td></tr>
<tr><td rowspan="3">2</td><td colspan="2">Wait for a reply to the RTR frame in message box 1:</td></tr>
<tr><th>If ...</th><th>... then ...</th></tr>
<tr><td>... bit 1 <i>NEW-DAT</i> = 1 in MR 10550,</td><td>... the controller has received the reply to the RTR frame.
Proceed with step 3.</td></tr>
<tr><td>3</td><td colspan="2">Read the number of received bytes
Number of bytes := MR 10553;</td></tr>
<tr><td>4</td><td colspan="2">Read the received bytes
Data byte 0 := MR 10554;
Data byte 1 := MR 10555;
...
Data byte 7 := MR 10561;</td></tr>
<tr><td>5</td><td colspan="2">Acknowledge reception
MR 10563 := 4;</td></tr>
<tr><td>⇨</td><td colspan="2">The message box is again ready to receive.</td></tr>
</table>

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg | Germany

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

We automate your success.