

# X20(c)DS438A

Data sheet  
1.70 (August 2025)



## **Publishing information**

B&R Industrial Automation GmbH

B&R Strasse 1

5142 Eggelsberg

Austria

Telephone: +43 7748 6586-0

Fax: +43 7748 6586-26

[office@br-automation.com](mailto:office@br-automation.com)

## **Disclaimer**

All information in this document is current as of its creation. The contents of this document are subject to change without notice. B&R Industrial Automation GmbH assumes unlimited liability in particular for technical or editorial errors in this document only (i) in the event of gross negligence or (ii) for culpably inflicted personal injury. Beyond that, liability is excluded to the extent permitted by law. Liability in cases in which the law stipulates mandatory unlimited liability (such as product liability) remains unaffected. Liability for indirect damage, consequential damage, business interruption, loss of profit or loss of information and data is excluded, in particular for damage that is directly or indirectly attributable to the delivery, performance and use of this material.

B&R Industrial Automation GmbH notes that the software and hardware designations and brand names of the respective companies used in this document are subject to general trademark, brand or patent protection.

Hardware and software from third-party suppliers referenced in this document is subject exclusively to the respective terms of use of these third-party providers. B&R Industrial Automation GmbH assumes no liability in this regard. Any recommendations made by B&R Industrial Automation GmbH are not contractual content, but merely non-binding information for which no liability is assumed. When using hardware and software from third-party suppliers, the relevant user documentation of these third-party suppliers must additionally be consulted and, in particular, the safety guidelines and technical specifications contained therein must be observed. The compatibility of the products from B&R Industrial Automation GmbH described in this document with hardware and software from third-party suppliers is not contractual content unless this has been separately agreed in individual cases; in this respect, warranty for such compatibility is excluded in any case, and it is the sole responsibility of the customer to verify this compatibility in advance.

1395876371006-1.70

# 1 General information

## 1.1 Other applicable documents

For additional and supplementary information, see the following documents.

### Other applicable documents

Document name	Title
MAX20	<a href="#">X20 System user's manual</a>

## 1.2 Coated modules

Coated modules are X20 modules with a protective coating for the electronics component. This coating protects X20c modules from condensation and corrosive gases.

The modules' electronics are fully compatible with the corresponding X20 modules.



For simplification purposes, only images and module IDs of uncoated modules are used in this data sheet.

The coating has been certified according to the following standards:

- Condensation: BMW GS 95011-4, 2x 1 cycle
- Corrosive gas: EN 60068-2-60, method 4, exposure 21 days



## 1.3 Order data


Order number	Short description	Figure
	<b>IO-link modules</b>	
X20DS438A	X20 digital signal module, 4x IO-Link master V1.1, 4 digital channels configurable as inputs or outputs, 3-wire connections, NetTime function	
X20cDS438A	X20 digital signal module, coated, 4x IO-Link master V1.1, 4 digital channels configurable as inputs or outputs, 3-wire connections, NetTime function	
	<b>Required accessories</b>	
	<b>Bus modules</b>	
X20BM11	X20 bus module, 24 VDC keyed, internal I/O power supply connected through	
X20BM15	X20 bus module, with node number switch, 24 VDC keyed, internal I/O power supply connected through	
X20cBM11	X20 bus module, coated, 24 VDC keyed, internal I/O power supply connected through	
	<b>Terminal blocks</b>	
X20TB12	X20 terminal block, 12-pin, 24 VDC keyed	

Table 1: X20DS438A, X20cDS438A - Order data

### Optional accessories for sensors

Connectors and cables from the X67 system accessories can be used to connect standard IO-Link sensors and devices.

#### Sensors/Devices with M12 connector

<b>M12 connector</b>	
X67AC0C21	X67 female M12 connector, 5-pin, A-keyed, shielded, cage clamp connection
X67AC2C21	X67 female M12 connector, 5-pin, A-keyed, shielded, screw clamp connection

#### Sensors/Devices with M8 connector

<b>M8 connector</b>	
X67AC0P20	Female M8 connector, 4-pin, piercing connection
<b>M8 connector with cable, open on one side</b>	
X67CA0P20.xxxx	M8 attachment cable, 4-pin, straight
X67CA0P30.xxxx	M8 attachment cable, 4-pin, angled

## 1.4 Module description

The module is an IO-Link master that allows intelligent sensors and actuators to be connected to the X20 system in accordance with the IO-Link standard. The module can be used to operate up to 4 IO-Link devices. All IO-Link channels can be used either as standard digital inputs or outputs.

Functions:

- [IO-Link](#)
- [Parameter server](#)
- [Statistics counter](#)
- [NetTime Technology](#)
- [Flatstream communication](#)

### IO-Link

The module is an IO-Link master for controlling intelligent sensors and actuators per the IO-Link standard. Up to 4 IO-Link devices (IO-Link version 1.1) can be connected to the module.

### Parameter server

The parameter server permits the module to read configuration parameters of the connected IO-Link device. The data of the third-party device can thus be restored automatically, e.g. after replacing the IO-Link device.

### Statistics counter

Communication errors between individual IO-Link components can be easily recorded using the statistics counters.

### NetTime timestamps for IO-Link

Using these timestamps, applications can record value changes at on the IO-Link network and trigger events that have a higher resolution than the I/O cycle would allow.

### Flatstream communication

"Flatstream" was designed for X2X and POWERLINK networks and allows data transfer to be adapted to individual demands. This allows data to be transferred more efficiently than with standard cyclic polling.

## 2 Technical description

### 2.1 Technical data

Order number	X20DS438A	X20cDS438A
Short description		
I/O module	IO-Link master with 4 IO-Link interfaces	
General information		
B&R ID code	0xCAC0	0xEB57
Status indicators	IO-Link, operating state, module status	
Diagnostics		
Module run/error	Yes, using LED status indicator and software	
IO-Link operating state	Yes, using LED status indicator and software	
C/Q status	Yes, using LED status indicator and software	
Cable specification		
Cable type	3-pin standard sensor cable, unshielded	
Cable length	Max. 20 m	
Line capacitance	Max. 3 nF	
Loop resistance	Max. 6 Ω	
Power consumption		
Bus	0.01 W	
Internal I/O	0.71 W	
Additional power dissipation caused by actuators (resistive) [W]	-	
Certifications		
CE	Yes	
UKCA	Yes	
ATEX	Zone 2, II 3G Ex nA nC IIA T5 Gc IP20, Ta (see X20 user's manual) FTZÚ 09 ATEX 0083X	
UL	cULus E115267 Industrial control equipment	
HazLoc	cCSAus 244665 Process control equipment for hazardous locations Class I, Division 2, Groups ABCD, T5	
DNV	Temperature: <b>B</b> (0 to 55°C) Humidity: <b>B</b> (up to 100%) Vibration: <b>B</b> (4 g) EMC: <b>B</b> (bridge and open deck)	
CCS	Yes	
LR	ENV1	
KR	Yes	
ABS	Yes	
Sensor/Actuator power supply		
Voltage	I/O power supply minus voltage drop for short-circuit protection	
Voltage drop for short-circuit protection at 0.5 A	Max. 0.3 V	
Power consumption	Max. 12 W per IO-Link interface	
Short-circuit proof	Yes	
Overload protection		
Switch-off delay	Configurable using software	
Switch-off duration	Configurable using software	
IO-Link in master mode		
Transfer rates		
COM1	4.8 kbaud	
COM2	38.4 kbaud	
COM3	230.4 kbaud	
Limit values for COM3		
Max. connection capacity	22 nF (cable + IO-Link device)	
Max. load	96 Ω / 250 mA	
Data format	1 start bit, 8 data bits, 1 parity bit (even), 1 stop bit	
Bus level	24 VDC (active), 0 VDC (resting voltage)	
IO-Link in master mode or in SIO mode "digital output"		
Variant	Bipolar, positive and negative switching	
Peak short-circuit current	<1.3 A	
Residual voltage	<0.7 VDC at nominal current 0.25 A	
Switching voltage	I/O power supply minus voltage drop for short-circuit protection and semiconductor switch	
Voltage drop on semiconductor switch	Max. 0.5 VDC at 0.25 A	

Table 2: X20DS438A, X20cDS438A - Technical data


Order number	X20DS438A	X20cDS438A
Switching frequency	Typ. 25 kHz 300 kHz in IO-Link master mode	
Switching delay		
0 → 1	<10 µs	
1 → 0	<10 µs	
Switch-on in the event of overload shutdown or short-circuit shutdown	Configurable with software	
Insulation voltage between IO-Link and bus	500 V <sub>eff</sub>	
IO-Link in SIO mode "Digital output"		
Nominal voltage	24 VDC	
Nominal output current	0.25 A	
Total nominal current	Max. 1 A	
Output circuit	Sink or source	
Switching frequency (resistive load)	Max. 500 Hz	
Output protection <sup>1)</sup>	Thermal shutdown in the event of overcurrent or short circuit, integrated protection for switching inductive loads	
IO-Link in SIO mode "digital input"		
Nominal voltage	24 VDC	
Input characteristics per EN 61131-2	Type 1	
Input filter		
Hardware	300 ns	
Software	-	
Input circuit	Sink	
Input voltage	24 VDC -15% / +20%	
Input current at 24 VDC	Typ. 4 mA	
Input resistance	Typ. 6 kΩ	
Switching threshold		
Low	<5 VDC	
High	>15 VDC	
Insulation voltage between IO-Link and bus	500 V <sub>eff</sub>	
Electrical properties		
Electrical isolation	Bus isolated from IO-Link IO-Link not isolated from IO-Link	
Operating conditions		
Mounting orientation		
Horizontal	Yes	
Vertical	Yes	
Installation elevation above sea level		
0 to 2000 m	No limitation	
>2000 m	Reduction of ambient temperature by 0.5°C per 100 m	
Degree of protection per EN 60529	IP20	
Ambient conditions		
Temperature		
Operation		
Horizontal mounting orientation	-25 to 60°C	
Vertical mounting orientation	-25 to 50°C	
Derating	-	
Storage	-40 to 85°C	
Transport	-40 to 85°C	
Relative humidity		
Operation	5 to 95%, non-condensing	Up to 100%, condensing
Storage	5 to 95%, non-condensing	
Transport	5 to 95%, non-condensing	
Mechanical properties		
Note	Order 1x terminal block X20TB12 separately. Order 1x bus module X20BM11 separately.	Order 1x terminal block X20TB12 separately, order 1x bus module X20cBM11 separately.
Pitch	12.5 <sup>+0.2</sup> mm	

Table 2: X20DS438A, X20cDS438A - Technical data

1) Cutoff current on overload: Between 0.3 A and 0.8 A.

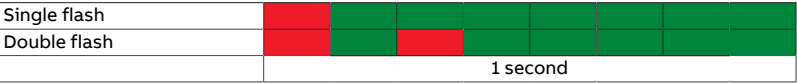
2.2 LED status indicators

For a description of the various operating modes, see section "Additional information - Diagnostic LEDs" in the X20 system user's manual.

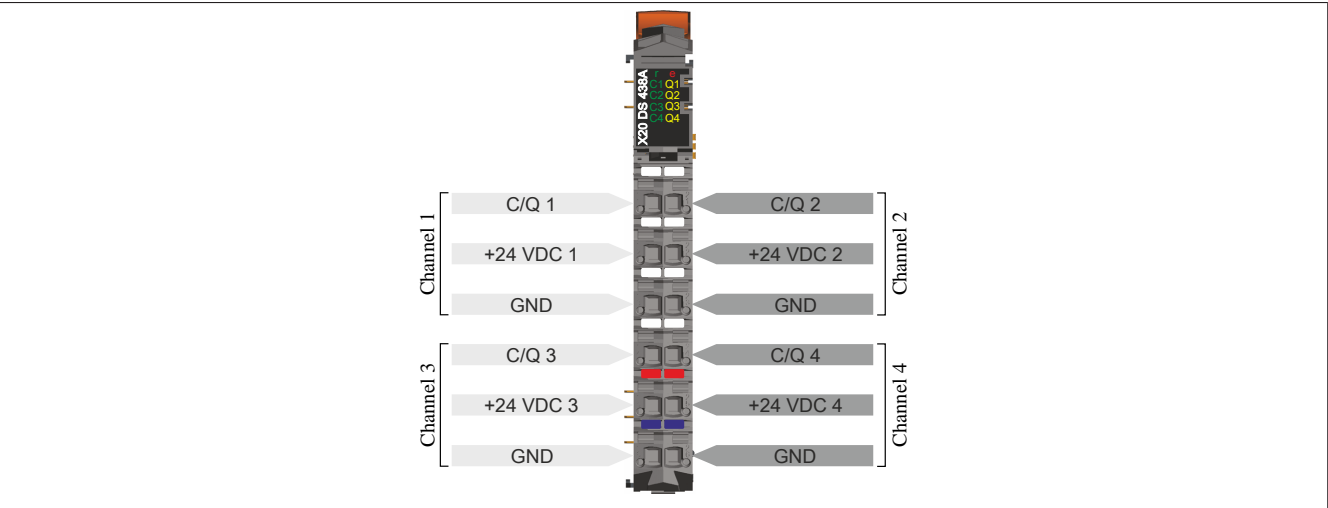
Figure	LED	Color	Status	Description
	r	Green	Off	No power to module
			Single flash	RESET mode
			Double flash	Mode BOOT (during firmware update) <sup>1)</sup>
			Blinking	PREOPERATIONAL mode
			On	OPERATE mode
	e	Red	Off	No power to module or everything OK
			On	Error or reset status
			Double flash	A module-internal error occurred. <sup>2)</sup>
	C1 - C4	Red	On	Overload on the supply or on the channel's C/Q line
		Green/Red	Off	Interface in SIO mode
			Single flash	Channel in OPERATE mode, no IO-Link communication
			Double flash	Channel in OPERATE mode, error on inspection level
		Green	On	Channel in OPERATE mode, IO-Link communication active
	Q1 - Q4	Orange		The LED lights up when there is data traffic on the C/Q line in SIO mode.

1) Depending on the configuration, a firmware update can take up to several minutes.  
2) Please contact B&R Support.

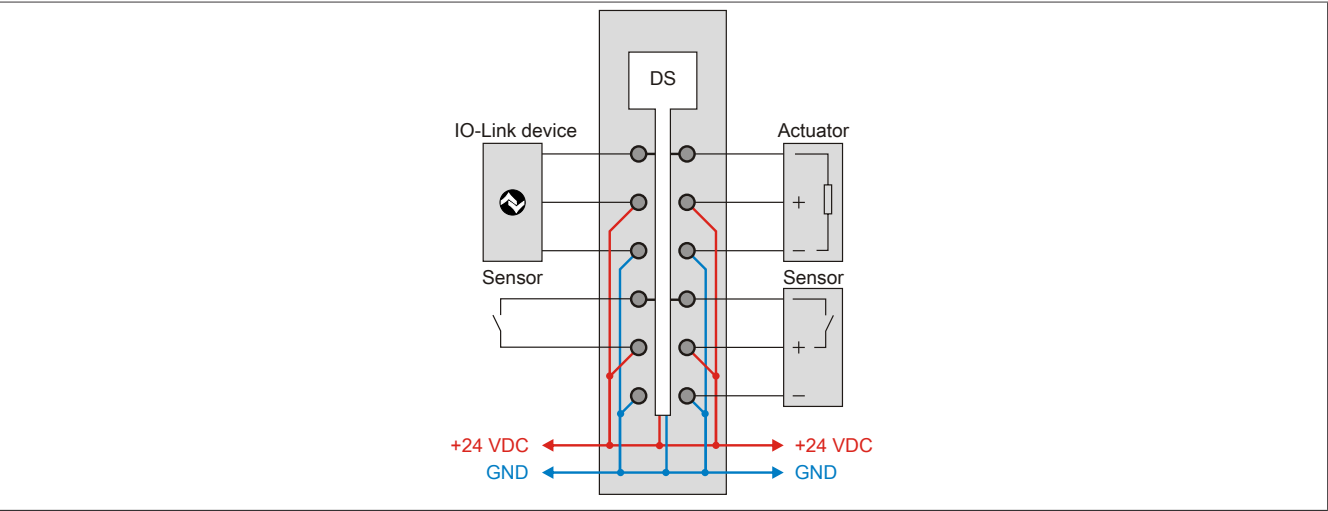
Blink times of LEDs C1 through C4 for single and double flash.



2.3 Pinout

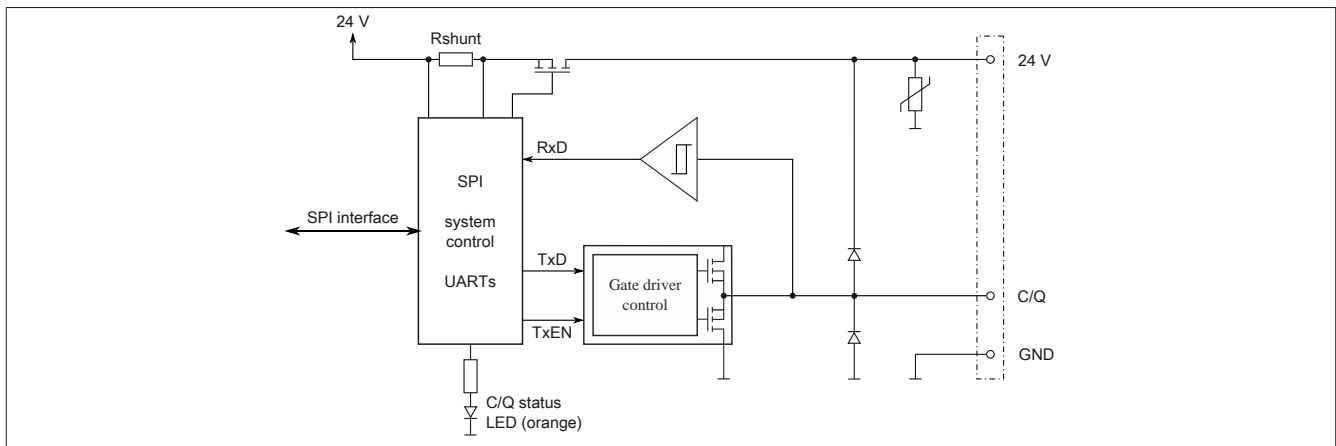


2.4 Connection example





## 2.5 Input/output circuit diagram



## 2.6 Overload protection

The module can be used to connect IO-Link devices (IO-Link version 1.1). The power supply for the IO-Link sensor/actuator is permitted to be obtained from the module. The power supply can be switched on or off individually for each IO-Link channel.

To avoid damage to the hardware, the module is equipped with overload protection.

## 2.7 Usage after the X20IF1091-1

If this module is operated after X2X Link module X20IF1091-1, delays may occur during the Flatstream transfer. For detailed information, see section "Data transfer on the Flatstream" in X20IF1091-1.

## 3 Function description

### 3.1 IO-Link

The module is an IO-Link master for controlling intelligent sensors and actuators per the IO-Link standard. Up to 4 IO-Link devices (IO-Link version 1.1) can be connected to the module.

#### 3.1.1 IO-Link communication

The module establishes communication with the IO-Link device if register "ChannelMode" on page 64 of the corresponding channel is configured.

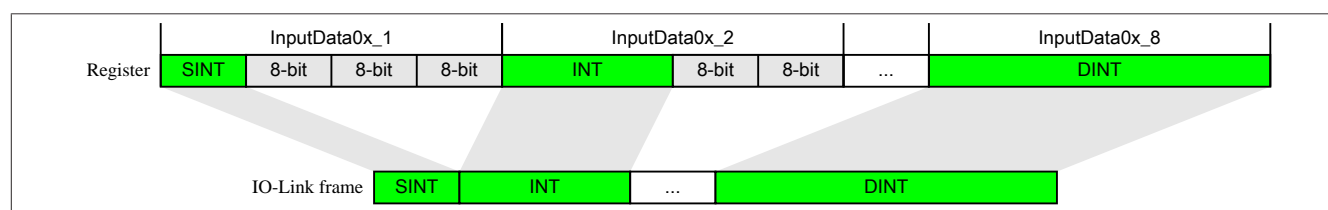
If the corresponding IO-Link channel of the module has been configured to "Operate" in Automation Studio, the IO-Link module attempts to exchange process data with the connected IO-Link device. For each active IO-Link channel, 8 [InputData0x\\_y](#) and 8 [OutputData0x\\_y](#) registers are allocated in the memory of the module.

Register	InputData0x_1				InputData0x_2				...	InputData0x_8			
	8-bit	8-bit	8-bit	8-bit	8-bit	8-bit	8-bit	8-bit		8-bit	8-bit	8-bit	8-bit

In order to define the actual IO-Link frame, how many of the maximum 8 registers are used must be defined as well as the data type of the IO-Link data.

Register	InputData0x_1				InputData0x_2				...	InputData0x_8			
	SINT	8-bit	8-bit	8-bit	INT	8-bit	8-bit			DINT			

The IO-Link frame for communication with the IO-Link device results from these initialized data points.



In order to transfer the IO-Link data to the PLC, the bandwidth of the X2X Link network must also be taken into account when defining the data type for the IO-Link communication. This limitation can be minimized if the "OCTET" data points or "multiplexed OCTET" data points are used instead of the standard data types.

#### "OCTET" byte arrays

8 registers with up to 32 bits are available per channel and direction. This way, 8 data points can be transferred. If this amount of data is insufficient, a byte array can be used to generate the IO-Link frame. The user must manage the distribution of IO-Link frames within the application and observe the byte order in the IO-Link device.

#### "Multiplexed OCTET" byte arrays

When transferring the IO-Link data via the X2X Link network, it is important to note that not all 32 bytes of the 4 IO-Link channels can be simultaneously registered on the cyclic section of the X2X Link network. In the input direction, the module has been extended in this respect. Time-multiplexed data transfer can take place in the background. Depending on the amount of data, several X2X cycles may be required to transfer new data between the module and the controller. This mode is not available in the output direction. Therefore, IO-Link output data can only be generated with a maximum length of 27 bytes.

#### SIO mode

"SIO" stands for "standard I/O" and defines an alternative use for the C/Qx connection. If a channel of the module is not required for IO-Link communication, the pin can be used as standard I/O. The user can decide whether to use the standard I/O as input or output. The IO-Link standard also permits IO-Link communication to be stopped and restarted. If IO-Link communication is stopped at runtime, the C/Qx connection can be used as a standard output.

## Process data

To transfer process data from the IO-Link device to the controller (application), the information is first read from the module and saved temporarily. Typically, 4 bytes are reserved for each piece for registered information. This configures how the incoming IO-Link process data stream (IO-Link frame) is divided. According to this configuration, the IO-Link process data is made available to the application via the corresponding InputData registers. The InputData registers are assigned to individual data points with the corresponding data type in the I/O mapping.

In order to transfer process data to the IO-Link device, it is necessary to configure which data type of the individual "OutputData" registers is used to combine the outgoing IO-Link process data stream. According to this configuration, OutputData registers are assigned to data points with the corresponding data types in Automation Studio (I/O mapping).

If a byte array is used, it is up to the user to assign the required data types to the corresponding bytes.

Data type	Values
USINT	0 to 255
SINT	-128 to 127
UINT	0 to 65535
INT	-32768 to 32767
UDINT	0 to 4,294,967,295
DINT	-2147483648 to 2147483647
REAL	-3.4E38 to 3.4E38



### Information:

The registers are described in ["IO-Link communication" on page 69](#).

### 3.1.2 Configuring IO-Link timing characteristics

The module must manage records from 2 different communication standards at runtime. For efficient communication on the X2X Link network, it must be ensured that the cycle time of all X2X modules corresponds to the bus cycle time.

#### IO-Link specified cycle times

The IO-Link specification defines that an IO-Link device must be queried at certain intervals. This cycle is referred to as the IO-Link cycle.

Valid IO-Link cycle times are in the range from 0.5 ms to 132.8 ms. A distinction is made between 3 different areas.

Area	Increment	Calculation	Valid cycle times
0.5 to 6.3 ms	0.1 ms	$\text{Cycle time} = 0.1 \text{ ms} * n + 0.4 \text{ ms}$	0.5, 0.6 to 6.2, 6.3 ms
6.4 to 32.6 ms	0.4 ms	$\text{Cycle time} = 0.4 \text{ ms} * n + 6.4 \text{ ms}$	6.4, 6.8, 7.2 to 32.2, 32.6 ms
32.0 to 132.8 ms	1.6 ms	$\text{Cycle time} = 1.6 \text{ ms} * n + 32.0 \text{ ms}$	32.0, 33.6, 35.2 to 131.2, 132.8 ms

#### Module timer

As a basis for synchronizing individual channels, the module is equipped with an internal module timer that applies globally to all channels. This defined time base can be used to synchronize X2X and IO-Link communication with each other. The period duration of the modulator can be specified in microseconds. To make communication as efficient and deterministic as possible, the module timer is configured by default in automatic mode with the same cycle time that is used to operate the X2X Link network. If the module timer should not be operated with the same cycle, the period duration of the module timer can be set. This allows channels to be synchronized with each other even if a very unusual X2X cycle time is used.

If necessary, startup of the module timer can be delayed using the "[timer offset](#)" on page 68.

The cycle of the module timer is automatically synchronized with the X2X cycle. Different ratios result between cycles depending on the ratio between the X2X and module timer cycle times.

#### Examples

1 to 1	(X2X cycle 1000, timer cycle 1000)	→ Always exactly one timer cycle per X2X cycle
2 to 1	(X2X cycle 2000, timer cycle 1000)	→ Always exactly two timer cycles per X2X cycle
1 to 2	(X2X cycle 1000, timer cycle 2000)	→ Always exactly one timer cycle per 2 X2X cycles
3 to 5	(X2X cycle 1500, timer cycle 2500)	→ Always exactly 3 timer cycles per 5 X2X cycles

#### Synchronous operation

In contrast to free-running operation, in synchronous operation the synchronization cycle time can be set individually for each channel.

Operating mode SYNCHRONIZED optimizes the interaction of X2X Link and IO-Link communication. The resources of the module were designed for this mode; this configuration should therefore be used for the channels of the module.

- In operating mode SYNCHRONIZED (automatic), the module calculates the required time parameters by itself. An IO-Link cycle is determined that corresponds to the IO-Link specification. The selected IO-Link cycle time corresponds to the smallest possible multiple of the module timer cycle time that meets the following conditions:
  - Valid IO-Link cycle time
  - Greater than or equal to the minimum cycle time of the device
- In operating mode SYNCHRONIZED (manual), the user can manually configure the timing characteristics of the module. The user can define both the synchronization cycle time and IO-Link cycle manually using a factor.
- CycleMultiple and CycleDivisor  
These registers can be used to manually set the synchronization cycle time and IO-Link cycle time of a channel.



#### Information:

**If the values of registers CycleMultiple and CycleDivisor are not defined for an IO-Link channel or are set to zero, the values are calculated automatically during module start-up.**

## Synchronization cycle time

$$\text{Synchronization cycle time} = \text{Timer cycle time} * \text{CfO\_ReqCycleMultiple0x}$$

The synchronization ensures that synchronization cycles run parallel with the same synchronization cycle time and are not offset by timer cycles.

## IO-Link cycle time

$$\text{IO-Link cycle time} = \text{Synchronization cycle time} / \text{CfO\_ReqCycleDivisor0x}$$

The IO-Link cycle is set individually for each channel. If necessary, the IO-Link cycle of the channel can be shifted using a channel-specific offset. Possible reasons for this:

- Channels should be aligned so that their requests end at the same time. With very short cycle times (<1 ms), it is possible that the data cannot be processed fast enough. The subsequent cycles are delayed in this case, which is indicated by a reset of the status bit for synchronization.
- All channels run with the same cycle time. All channels will be ready at the same time in this case, which may result in the module not processing all data in time. Offsets can be used to prevent such bottlenecks and distribute the data volume more evenly.

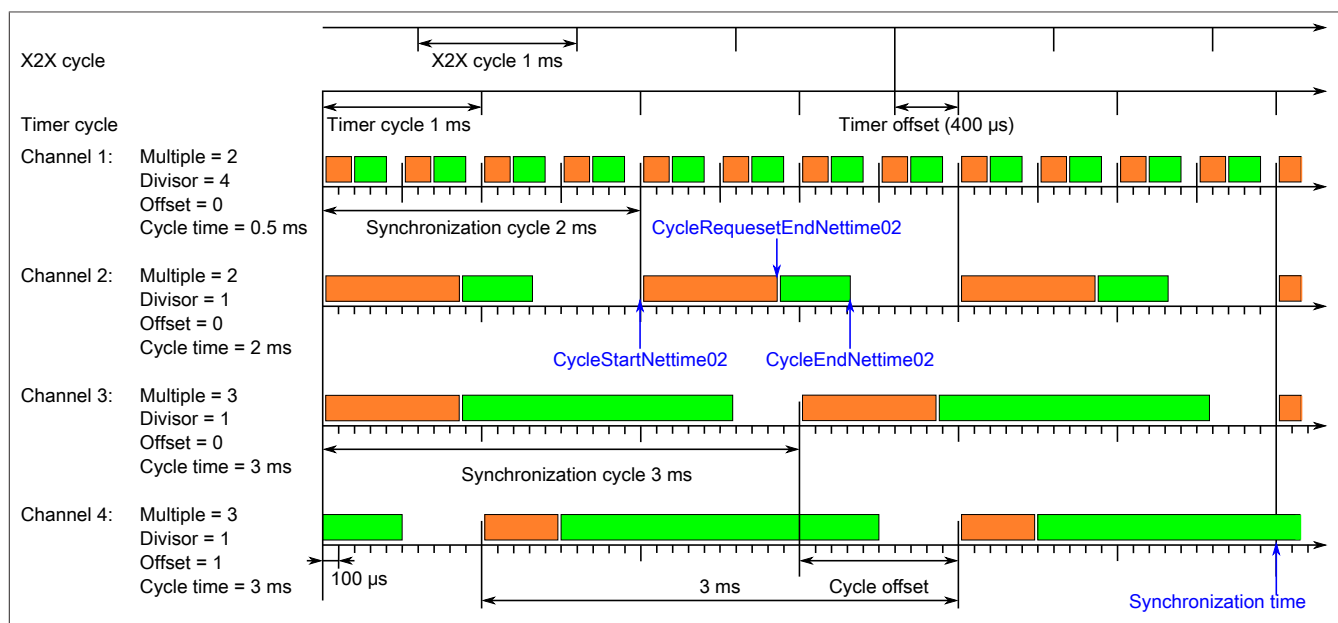


### Information:

If the IO-Link cycle is configured to be less than the minimum cycle time of the device, a cycle is automatically selected that meets the following conditions:

- Multiple of the module timer cycle
- Valid IO-Link cycle time
- Greater than or equal to the minimum cycle time of the device

## Configuration example



## Module timer in this example

- The period duration of the module timer was not explicitly defined. In this case, it corresponds to the cycle time of the X2X Link network.
- A timer offset of 400 μs was applied to the module timer; this means the module timer cycle starts 400 μs offset from the X2X cycle.

## IO-Link communication in this example

- The channel-specific cycle time for IO-Link communication results from parameters "Multiple" on page 68 and "Divisor" on page 68.
- Channels 1 and 2 have a common synchronization cycle of 2 ms. Channels 3 and 4 have a common synchronization cycle of 3 ms, shifted by the offset.
- Channels start the query simultaneously at the beginning of a common synchronization cycle.
- The IO-Link cycle of the fourth channel was delayed with an offset of 1 ms.
- All channels have a common synchronization cycle of 6 ms.

### Free-running (asynchronous) operation

If the IO-Link and X2X Link cycle times cannot be synchronized, then the IO-Link cycle time can be specified explicitly. IO-Link communication then runs independently of the module timer and X2X cycle. No other [NetTime](#) data points can be used except for "[CycleEndNettime](#)" on page 73. The cycle times of free-running IO-Link channels are defined directly via the corresponding registers. However, deviations may occur if the module's resources are exhausted.



#### Information:

- In free-running mode, no [NetTime](#) data points are permitted to be used except for "[CycleEndNettime](#)" on page 73.
- If the specified cycle time of the IO-Link communication undershoots the minimum cycle time of the device, the IO-Link data is queried with the minimum cycle time of the device.
- For efficient IO-Link communication, the set query cycle should correspond to the specified IO-Link cycle times. If the value is unsuitable, the next suitable cycle time is used automatically.



#### Information:

The registers are described in "[Configuring IO-Link timing characteristics](#)" on page 68.

### 3.1.3 IO-Link event interface

The event interface involves interrupt-controlled background communication. It enables the connected IO-Link devices to transmit special messages, or "event codes", to the master.

The module can receive up to 16 of these messages, buffer them and make them available for retrieval from the controller. Essentially, FIFO memory is used for this; this is managed independently of the cyclic communication.



#### Information:

If a message is received via the event interface and the FIFO memory is full, the oldest message in the buffer is overwritten. In rare cases, this can result in messages being lost before they have been evaluated.

#### Sequence for reading an event

- A new event was triggered by the device. This is indicated by an increase in "EventPortSeq" on page 74.
- Event data can be read out using registers "EventQualifier" on page 74 and "EventCode" on page 74.

Description	Information
<b>EventQualifier</b>	
Instance layer that generated the event	<ul style="list-style-type: none"> <li>• Instance layer unknown</li> <li>• Hardware</li> <li>• Data exchange layer of the IO-Link device</li> <li>• Application layer of the IO-Link device</li> <li>• Application</li> </ul>
Cause of the event	<ul style="list-style-type: none"> <li>• Device</li> <li>• Master</li> </ul>
Type of event	<ul style="list-style-type: none"> <li>• Information</li> <li>• Warning</li> <li>• Error</li> </ul>
Event mode	<ul style="list-style-type: none"> <li>• One-time event</li> <li>• Event no longer reported (e.g. voltage OK again)</li> <li>• Event reported (e.g. voltage too low)</li> </ul>
<b>EventCode</b>	
	The event codes can consist of vendor-specific event codes or event codes specified by the IO-Link specification.

- The event must be acknowledged. To do so, the sequence number from "EventPortSeq" on page 74 must be copied to the sequence number from "EventQuit" on page 75.
- The next event is specified only after the event is acknowledged.



#### Information:

The registers are described in "IO-Link event interface" on page 74.

### 3.1.4 IO-Link device IDs

IO-Link device IDs are defined by the manufacturer of the IO-Link device and cannot be modified by the user. They can be used to clearly identify a connected IO-Link device.

These include:

- Manufacturer ID
- Unique ID of the IO-Link device
- Function class of the device assigned by the manufacturer
- Currently applied IO-Link cycle time
- Currently applied multiplier
- Currently applied divisor
- Minimum IO-Link cycle time
- Specified size of the input process data
- Defined size of the output process data
- Specified baud rate
- IO-Link version



#### Information:

The registers are described in ["IO-Link device IDs" on page 79](#).

### 3.1.5 IO-Link status response

Status information provides information about the current situation between the module and IO-Link device. It can be retrieved from the controller and evaluated in the application task.

The following status information can be read out:

- Whether the last data transmitted to the IO-Link device was processed
- Whether the channel is synchronized without errors
- Whether an overload has occurred on the channel power supply or data line in the form of overcurrent or overtemperature
- Current status of the IO-Link channel
- Number of all received IO-Link frames. It is ensured that all frames are really detected, even if X2X cycles are lost or if the IO-Link cycle is faster than the X2X cycle.
- Start and end time of the last IO-Link cycle



#### Information:

The registers are described in ["IO-Link status response" on page 71](#).



### 3.1.6 IO-Link timestamp

The IO-Link timestamp registers allow the assignment of IO-Link timestamps to the NetTime of a controller, and vice versa.

This makes it possible for value changes of the IO-Link device to be assigned exactly to the NetTime of the controller, and vice versa. Events can be captured or triggered with a higher timing resolution than would be possible with the IO-Link cycle. This allows a highly precise timed response from the controller to signals from the sensor, and vice versa. The resolution depends on the devices being used.

For additional information about NetTime and timestamps, see "[NetTime Technology](#)" on page 26.

#### Examples

- For an input device, the timestamp is saved directly by the device when a certain event occurs (e.g. light barrier triggered) and then transferred via IO-Link. The IO-Link master converts this IO-Link-specific timestamp to a NetTime timestamp that can be used across the system.
- In the output direction, a converted timestamp is transferred to the device via IO-Link. The output device reacts at the corresponding time and performs the intended event (e.g. closing a switch).



#### Information:

- The timestamp function is device-specific and not supported by every IO-Link device.
- This function cannot be used if the channel is operated in free-running (asynchronous) mode.

#### Input timestamp

For input timestamps, associated status information can be retrieved.

Description	Information
Sequence number	The sequence number is incremented by 1 with each valid timestamp received. In the event that the sequence number has been increased by more than 1, an event has been lost.
Events triggered by the application	Signal state at occurrence of the timestamp <b>Example:</b> Signal state at occurrence of the timestamp <ul style="list-style-type: none"> <li>– Light barrier was interrupted → This bit = 0</li> <li>– Light barrier free → This bit = 1</li> </ul>
Timestamp error	No error → This bit = 0 An error has occurred on the IO-Link device → This bit = 1. Possible causes: <ul style="list-style-type: none"> <li>• More timestamps were generated than could be transferred.</li> <li>• The value of the IO-Link timestamp exceeded the permissible range of values.</li> </ul> In both cases, reducing the IO-Link cycle time can help.

#### Output timestamp

The NetTime for the output timestamp is automatically converted to an IO-Link timestamp. The event is triggered at the defined NetTime.



#### Information:

The NetTime must be at least 3 IO-Link cycles in the future; otherwise, a warning is set in `IoLinkTimestampOutStatus`.

The data type of the output timestamp must be identical to the data type defined in bit 26 of register "[ChannelMode](#)" on page 64.



#### Information:

The registers are described in "[IO-Link timestamp](#)" on page 78.

### 3.1.7 Saving IO-Link data

Some IO-Link devices must be instructed to save imported data storage parameters remanently after a download. In order to apply the data memory parameters for these devices to remanent memory, an specific [memory command](#) must be transmitted additionally (e.g. value 163 on index 2, subindex 0).

Since this type of memory command is not provided for in the specification, the option of configuring the device-specific memory command and associated index/subindex was implemented in the module. If a storage command is configured, it is transmitted automatically after a successful download of the data storage parameters or offline configuration.



#### Information:

The registers are described in ["Saving IO-Link data" on page 77](#).

### 3.1.8 Error codes

Requests can be made via registers or the Flatstream. If a request fails, the error bit is set and an error code is generated. In addition to the general error codes, vendor-specific error codes may also occur. For these, see the operating instructions for the corresponding IO-Link device.

#### Error display in the registers

- The error bit is set in ["ParameterCtrlIn" on page 84](#), and the length of the error code is displayed in the data length parameter.
- ["ParameterDataIn" on page 84](#) contains the error code.

#### Error display in the Flatstream

If the error bit is set, the Flatstream bytes are composed as follows:

- Bytes 1 to 3: Module-specific Flatstream array
- Byte 4: Error code. For error code 8 (error reported by the device), bytes 5 and 6 contain additional information.
- Bytes 5 and 6: Error code from the IO-Link device
- ...

#### Error codes

Code	Explanation
1	No device on this channel
2	IO-Link disabled
3	Communication error with device
4	Request buffer full
5	Event queue empty
6	Request not supported
7	Object access failed
8	Object access, error reported by device
9	Incorrect channel number
10	No write access possible
11	No input data available
12	Frame too short
13	One or more events discarded
14	Device has no input data.
15	Device has no output data.

## 3.2 Parameter server

The parameter server is a function that is defined by the IO-Link specification. This function is normally enabled in the module and can be managed with register "[CfO\\_DS\\_Config](#)" on page 76.

The parameter server permits the module to read configuration parameters of the connected IO-Link device. The data of the third-party device is stored in the EEPROM and can then be restored automatically, e.g. after replacing the IO-Link device.



### Information:

**The selection of the transferred configuration data depends on the connected IO-Link device. The module only functions as data storage. It requests the configuration data of the IO-Link device, stores the response and transmits the received information back to the connected IO-Link device, if required.**

**A change to the read parameter server data in the memory of the module is not foreseen.**

### Event code 0xFF91

The module is able to process the "data memory upload request" (event code 0xFF91) of the connected IO-Link device in order to automatically manage the memory of the parameter server in the module.

The standard does not specify exactly when the event code must be generated. Most IO-Link devices generate it as soon as the configuration parameters change. With some IO-Link devices, it can be advantageous to request the upload and download processes manually. For this purpose the module includes an option for adapting the transfer of the parameter server data to the individual application requirements.



### Information:

**Automatic management can be used if the connected third-party IO-Link device supports the parameter server function and can generate the event code.**

### The parameter server

If supported by the IO-Link device, the IO-Link parameter server can be used to read application-specific device configurations from the IO-Link master, for example. The module's parameter server is always enabled and can be used using a control register.

Which data storage parameters are transferred depends on the connected IO-Link device. The read information is stored in EEPROM on the module and can be fed back automatically after replacing the device, for example.

The module is able to process the data memory upload request (event code 0xFF91) of the IO-Link specification. The request is usually triggered when parameters are changed on the device. Depending on the configuration, an upload of the data memory data can be started in this case (default).

### Automatic management of data storage parameters

Automatic management has been designed according to IO-Link specification. Since the IO-Link standard exhibits a degree of tolerance here, it is possible that some IO-Link devices may have to be handled differently. This can be configured using the register.

An upload/download is performed under the following conditions:

- DsControl0x = 1
- While the device is starting up or if a data storage upload request has been received.

## Function description

### Offline configuration

With offline configuration, the configuration data set in Automation Studio for the device is saved in the project and automatically configured for the controller after the project is downloaded or after the memory card is created. Unlike the parameter server, where the values are read out from an existing device, the values are specified directly by the application in this case. The values are configured automatically only once after the download. The procedure is not repeated until a new parameter file is received from Automation Studio, the device has been replaced or the download is started manually by the library.

This function works independently of the parameter server. If the parameter server is enabled, however, it starts after the offline configured if required and saves the corresponding data. In this case, the data is loaded from the parameter server to the device when the device is replaced.

### Manual operation

The behavior of the parameter server can be set when the parameter server is operated manually. A corresponding reaction can be assigned to each trigger event here.

The following applies as the freely adjustable order of trigger events:

- The device ID of the connected device does not match the device ID stored together with the parameters.
- The device transmitted an upload request.
- A new parameter checksum was detected during device startup.
- The serial number of the connected device does not match the serial number stored together with the parameters.

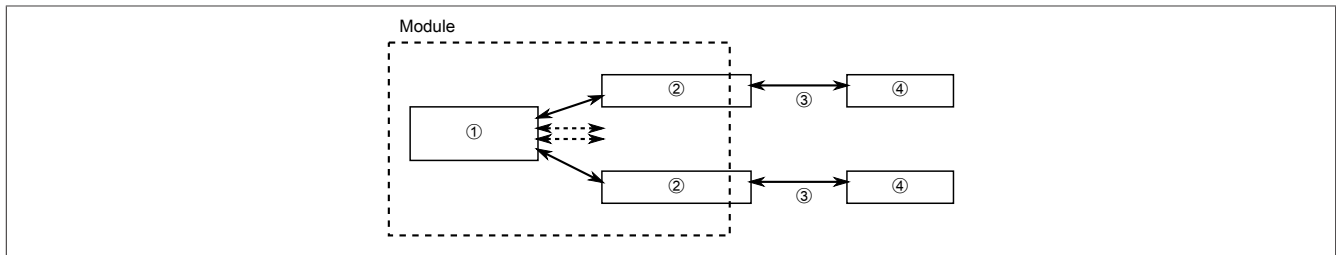


### Information:

The registers are described in "[IO-Link parameter server](#)" on page 75.

### 3.3 Statistics counter

Communication errors that have occurred between individual IO-Link components are mapped in the statistics counters.



#### Legend

- ① I/O processor
- ② Channel-specific IO-Link interface
- ③ IO-Link channel
- ④ IO-Link device

The following error messages are recorded:

- Number of command retries caused by communication errors between the I/O processor and IO-Link device.
- Number of checksum errors between the I/O processor and channel-specific IO-Link interface
- Number of communication errors between the I/O processor and channel-specific IO-Link interface
- Number of parity errors between the channel-specific IO-Link interface and IO-Link device
- Number of protocol errors between the channel-specific IO-Link interface and IO-Link device
- Number of bytes received with errors between the channel-specific IO-Link interface and IO-Link device.
- Number of checksum errors between the channel-specific IO-Link interface and IO-Link device
- Number of response errors. These occur if the IO-Link device does not respond in time to the request frame of the master or if the pause between the individual bytes in the response frame is too large.
- Number of cycle errors. These occur if an IO-Link cycle is started before the previous one could be completed and processed. These errors can be corrected by setting a lower cycle time.



#### Information:

The statistics counter registers are only available starting with firmware version 240.



#### Information:

The registers are described in "[Statistics counter](#)" on page 81.

### 3.4 Communication via the command interface

The command interface provides the possibility of accessing the object dictionary of the IO-Link device via index and subindex. Alternatively, access can also take place using library AsIoLink or the Flatstream.



#### Information:

A maximum of the first 4 bytes of an object can be read or written with this interface.

#### Procedure for write access:

- Set the object to be written using "ParameterIndexOut" on page 83 and "ParameterSubIndexOut" on page 83.
- Write the data to be written to "ParameterDataOut" on page 83.
- Set read/write, IF and the sequence number incremented by 1 in register "ParameterCtrlOut" on page 83.
- Wait until the sequence confirmation in "ParameterCtrlIn" on page 84 is equal to the sequence number.

#### Procedure for read access:

- Set the object to be read using "ParameterIndexOut" on page 83 and "ParameterSubIndexOut" on page 83.
- In parameter "ParameterCtrlOut" on page 83, delete bit 7, set the channel number and increase the sequence number.
- Wait until the sequence confirmation in "ParameterCtrlIn" on page 84 is equal to the sequence number.
- Read out the error state from "ParameterCtrlIn" on page 84. An error is indicated by a set error bit.
- Read the data from "ParameterCtrlIn" on page 84.

#### Behavior in the event of error

For details about the behavior when an error occurs, see "Error codes" on page 18.

#### Error display

- If the "error code" on page 18 is not equal to 8 (i.e. error reported by the device), then the LSB of register ParameterDataIn contains the error code.
- In the event of an error reported by the device, the error specified by the IO-Link device is also displayed.

UDINT			
MSB			LSB
Reserved	IO-Link error code	Additional IO-Link error code	8



#### Information:

The registers are described in "IO-Link communication via the command interface" on page 83.

### 3.5 Flatstream communication

The module enables the user to communicate with the connected IO-Link device using the Flatstream.

Communication is separated in time, i.e. output data is transferred completely from the controller to the module and checked. The module then initiates the actual communication with the IO-Link device.

The module behaves the same way in the input direction. Messages from the IO-Link device must be received in full in the X2X module before the Flatstream message is generated and transmitted to the controller.



#### Information:

The registers are described in ["Flatstream registers" on page 84](#).

#### 3.5.1 General handling of the Flatstream

Input/Output sequence	Rx/Tx bytes	
(Unchanged)	Control byte (unchanged)	Payload data array for Flatstream (IO-Link information)

The user has a choice when using the Flatstream.

- Using the Flatstream as described in ["Flatstream communication" on page 29](#).
- Using library "AsFltGen" to manage input/output sequences and the Flatstream control bytes automatically.

In both cases, a module-specific array with Flatstream payload data must be created in the application.

#### 3.5.2 IO-Link information for the Flatstream

An individual array must be defined in the application to be able to use IO-Link communication via the Flatstream.

The following must be defined for the request in the controller → module → IO-Link device direction.

- Channel number of the module
- Frame number for the request
- Type of request
- The corresponding IO-Link data must then be attached depending on the request.

The response consists of the following parts:

- The channel number, frame number, access type and type of request are repeated.
- The module generates the error bit and manages the confirmation bit.
- The successfully received IO-Link information or corresponding ["error code" on page 18](#) is then added.

#### Module-specific Flatstream array for IO-Link communication

Byte	Name	Value	Description
1	Channel number	1 to 4	
2	Frame number	0 to 255	This number is repeated in the module's response. In this way, the later response of the module can be clearly assigned to the request.
3	See <a href="#">Byte 3</a> .	x	
...	IO-Link data or <a href="#">error code</a>		Depends on byte 3

## Function description

### Byte 3

Bit	Description	Value	Information
0 - 2	Type of request	0	<a href="#">Access to the object dictionary</a>
		1	<a href="#">Access to the process data of the inputs</a>
		2	<a href="#">Access to the process data of the outputs</a>
		3	<a href="#">Read out single event</a>
		4	<a href="#">Read out multiple events</a>
		5	<a href="#">Enable event forwarding</a>
		6	<a href="#">Disable event forwarding</a>
		7	<a href="#">Announcement of an automatically forwarded event</a>
3 - 4	Reserved	-	
5	Confirmation	0	Message without request
		1	Response to request <sup>1)</sup>
6	Status bit (for response frame)	0	No error
		1	<a href="#">Error</a>
7	Access type	0	Read
		1	Write

- 1) This confirmation bit is also set for a response to a request. The response used to confirm a request often contains additional data that must be processed.

### 3.5.3 IO-Link data

Different IO-Link data must be attached to the Flatstream depending on the type of request.

#### 3.5.3.1 Access to the object dictionary

##### Request

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	Index number high	0 to 255	Index of the desired IO-Link parameter
5	Index number low	0 to 255	
6	Subindex number	0 to 255	Subindex of the IO-Link parameter
7 to ...	Data	0 to 255	Optional, for write access

##### Response

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4 to ...	Data / <a href="#">"Error code" on page 18</a>	0 to 255	Omitted if data was written successfully

#### 3.5.3.2 Access to process data

##### Request

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	Data	0 to 255	Optional, for write access

##### Response

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	Data / <a href="#">"Error code" on page 18</a>	0 to 255	Omitted if data was written successfully



### 3.5.3.3 Access to event data

#### Request

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		

#### Response

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	Event counter - Current	Bits 0 to 3	Number of attached events
	Event counter - Pending	Bits 4 to 7	Number of pending events
5	Event 0 - Event qualifier	0 to 255	See <a href="#">"EventQualifier" on page 74.</a>
6	Event 0 - Event data high	0 to 255	
7	Event 0 - Event data low	0 to 255	
8 - 10	Event 1		
x to (x + 2)	Event n <sup>1)</sup>		

1) Only applies if several events were queried with byte 3 (bits 0 to 2 = 4). Only 1 event occurs with byte 3 (bits 0 to 2 = 3).

or

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	<a href="#">"Error code" on page 18</a>	0 to 255	

### 3.5.3.4 Enabling or disabling event forwarding

#### Request

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		

#### Response

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		

or

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	<a href="#">"Error code" on page 18</a>	0 to 255	

### 3.5.3.5 Announcement of a forwarded event

After event forwarding has been enabled, events must no longer be queried cyclically. The module generates the event as soon as the corresponding event has occurred.

#### Message

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	Event counter - Current	Bits 0 to 3	Number of attached events
	Event counter - Pending	Bits 4 to 7	Number of pending events
5	Event 0 - Event qualifier	0 to 255	See <a href="#">"EventQualifier" on page 74.</a>
6	Event 0 - Event data high	0 to 255	
7	Event 0 - Event data low	0 to 255	
8 - 10	Event 1		
x to (x + 2)	Event n <sup>1)</sup>		

1) Only applies if several events were queried with byte 3 (bits 0 to 2 = 4). Only 1 event occurs with byte 3 (bits 0 to 2 = 3).

or

Byte	Name	Value	Description
1 to 3	Module-specific Flatstream array for IO-Link communication		
4	<a href="#">"Error code" on page 18</a>	0 to 255	

### 3.6 NetTime Technology

NetTime refers to the ability to precisely synchronize and transfer system times between individual components of the controller or network (controller, I/O modules, X2X Link, POWERLINK, etc.).

This allows the moment that events occur to be determined system-wide with microsecond precision. Upcoming events can also be executed precisely at a specified moment.



#### 3.6.1 Time information

Various time information is available in the controller or on the network:

- System time (on the PLC, Automation PC, etc.)
- X2X Link time (for each X2X Link network)
- POWERLINK time (for each POWERLINK network)
- Time data points of I/O modules

The NetTime is based on 32-bit counters, which are increased with microsecond resolution. The sign of the time information changes after 35 min, 47 s, 483 ms and 648  $\mu$ s; an overflow occurs after 71 min, 34 s, 967 ms and 296  $\mu$ s.

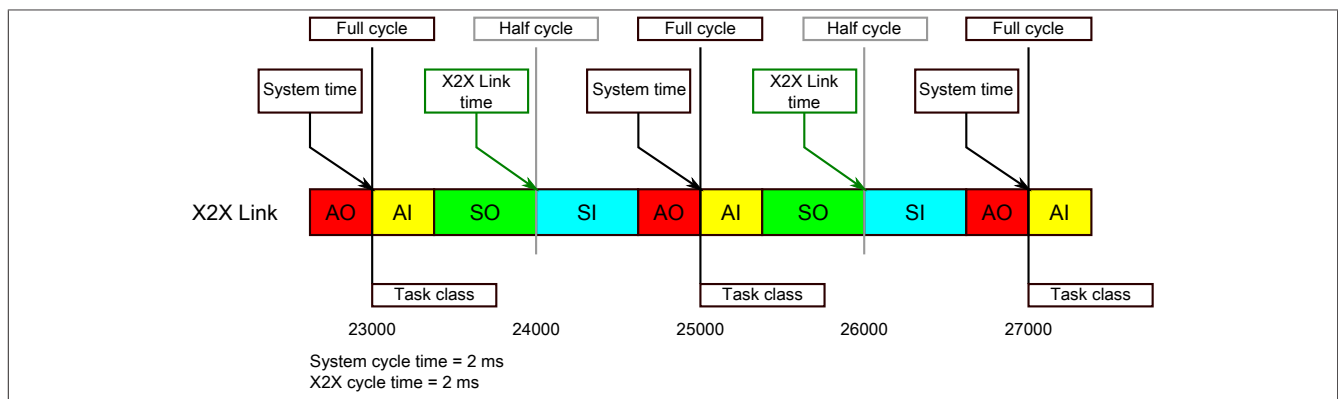
The initialization of the times is based on the system time during the startup of the X2X Link, the I/O modules or the POWERLINK interface.

Current time information in the application can also be determined via library AslOTime.

##### 3.6.1.1 Controller data points

The NetTime I/O data points of the controller are latched to each system clock and made available.

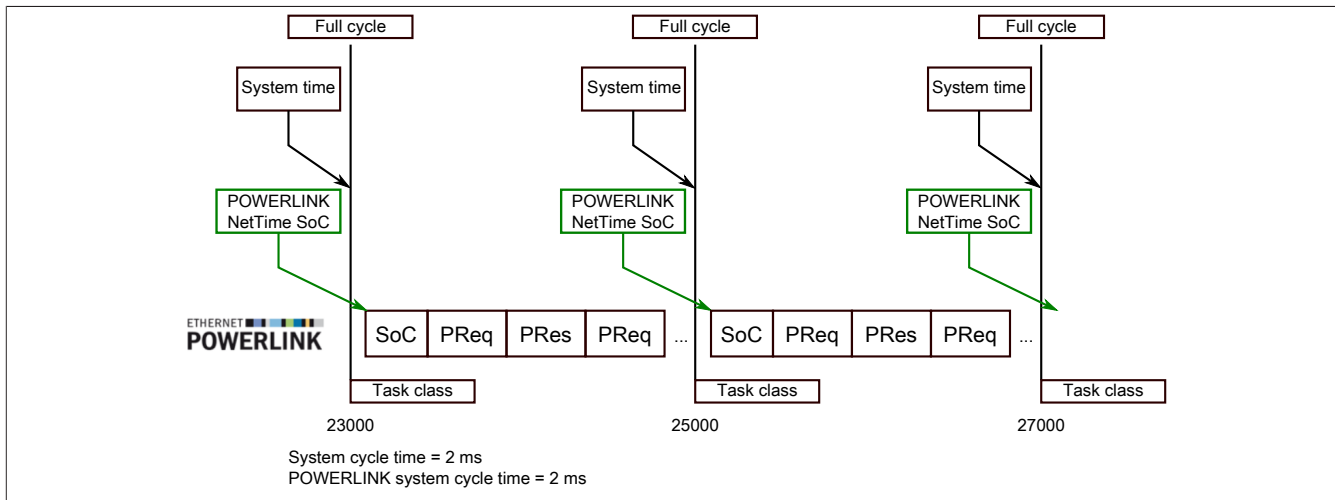
##### 3.6.1.2 X2X Link - Reference time point



The reference time point on the X2X Link network is always calculated at the half cycle of the X2X Link cycle. This results in a difference between the system time and the X2X Link reference time point when the reference time is read out.

In the example above, this results in a difference of 1 ms, i.e. if the system time and X2X Link reference time are compared at time 25000 in the task, then the system time returns the value 25000 and the X2X Link reference time returns the value 24000.

### 3.6.1.3 POWERLINK - Reference time point

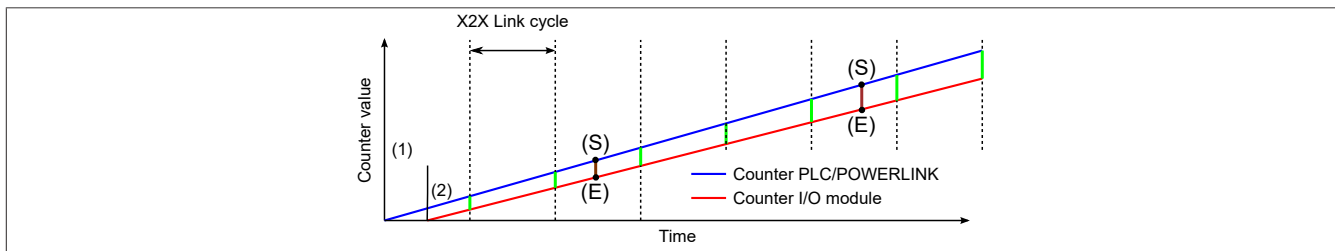


The POWERLINK reference time point is always calculated at the start of cycle (SoC) of the POWERLINK network. The SoC starts 20  $\mu$ s after the system clock due to the system. This results in the following difference between the system time and the POWERLINK reference time:

POWERLINK reference time = System time - POWERLINK cycle time + 20  $\mu$ s

In the example above, this means a difference of 1980  $\mu$ s, i.e. if the system time and POWERLINK reference time are compared at time 25000 in the task, then the system time returns the value 25000 and the POWERLINK reference time returns the value 23020.

### 3.6.1.4 Synchronization of system time/POWERLINK time and I/O module



At startup, the internal counters for the controller/POWERLINK (1) and the I/O module (2) start at different times and increase the values with microsecond resolution.

At the beginning of each X2X Link cycle, the controller or POWERLINK network sends time information to the I/O module. The I/O module compares this time information with the module's internal time and forms a difference (green line) between the two times and stores it.

When a NetTime event (E) occurs, the internal module time is read out and corrected with the stored difference value (brown line). This means that the exact system moment (S) of an event can always be determined, even if the counters are not absolutely synchronous.

#### Note

The deviation from the clock signal is strongly exaggerated in the picture as a red line.

### 3.6.2 Timestamp functions

NetTime-capable modules provide various timestamp functions depending on the scope of functions. If a timestamp event occurs, the module immediately saves the current NetTime. After the respective data is transferred to the controller, including this precise moment, the controller can then evaluate the data using its own NetTime (or system time), if necessary.  
For details, see the respective module documentation.

#### 3.6.2.1 Time-based inputs

NetTime Technology can be used to determine the exact moment of a rising edge at an input. The rising and falling edges can also be detected and the duration between 2 events can be determined.



**Information:**

**The determined moment always lies in the past.**

#### 3.6.2.2 Time-based outputs

NetTime Technology can be used to specify the exact moment of a rising edge on an output. The rising and falling edges can also be specified and a pulse pattern generated from them.



**Information:**

**The specified time must always be in the future, and the set X2X Link cycle time must be taken into account for the definition of the moment.**

#### 3.6.2.3 Time-based measurements

NetTime Technology can be used to determine the exact moment of a measurement that has taken place. Both the starting and end moment of the measurement can be transmitted.

## 3.7 Flatstream communication

### 3.7.1 Introduction

B&R offers an additional communication method for some modules. "Flatstream" was designed for X2X and POWERLINK networks and allows data transfer to be adapted to individual demands. Although this method is not 100% real-time capable, it still allows data transfer to be handled more efficiently than with standard cyclic polling.

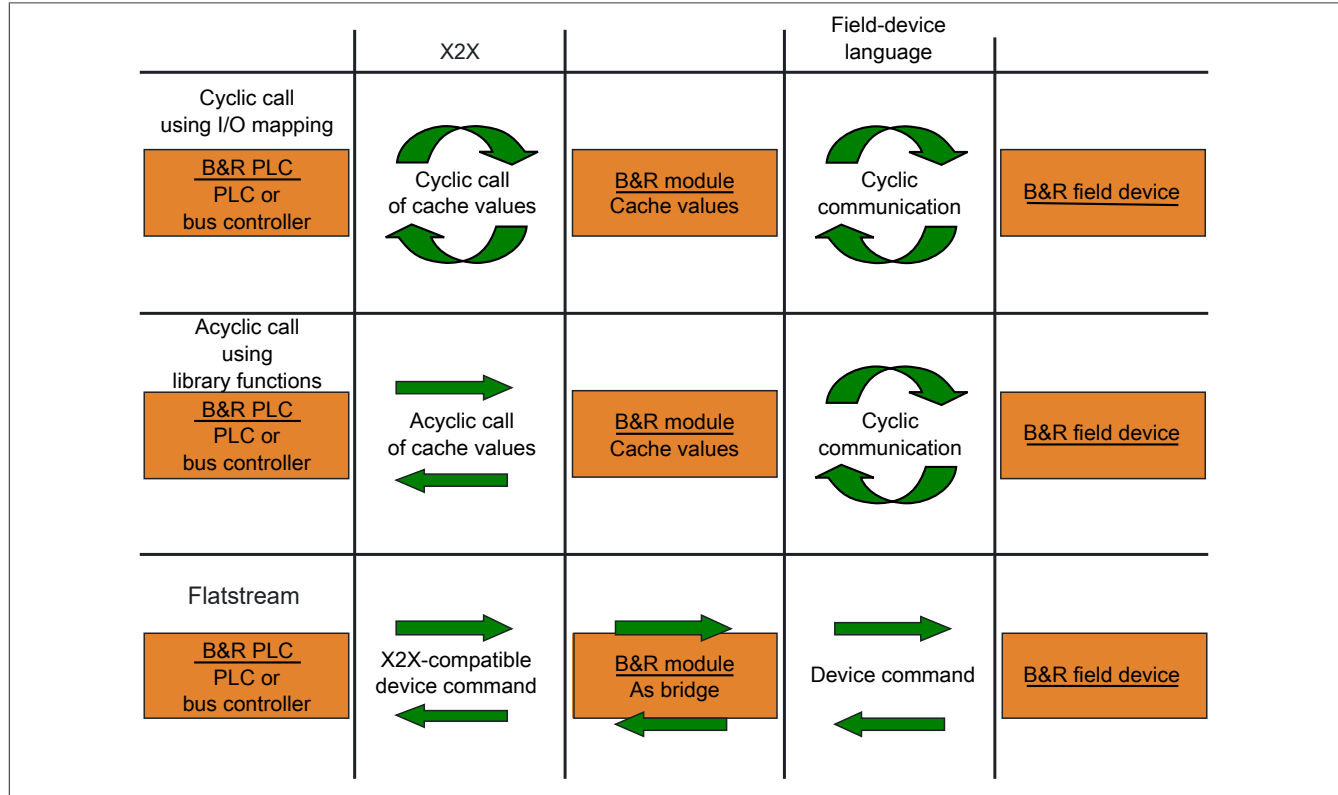


Figure 1: 3 types of communication

Flatstream extends cyclic and acyclic data queries. With Flatstream communication, the module acts as a bridge. The module is used to pass controller requests directly on to the field device.

### 3.7.2 Message, segment, sequence, MTU

The physical properties of the bus system limit the amount of data that can be transmitted during one bus cycle. With Flatstream communication, all messages are viewed as part of a continuous data stream. Long data streams must be broken down into several fragments that are sent one after the other. To understand how the receiver puts these fragments back together to get the original information, it is important to understand the difference between a message, a segment, a sequence and an MTU.

#### Message

A message refers to information exchanged between 2 communicating partner stations. The length of a message is not restricted by the Flatstream communication method. Nevertheless, module-specific limitations must be considered.

#### Segment (logical division of a message):

A segment has a finite size and can be understood as a section of a message. The number of segments per message is arbitrary. So that the recipient can correctly reassemble the transferred segments, each segment is preceded by a byte with additional information. This control byte contains information such as the length of a segment and whether the approaching segment completes the message. This makes it possible for the receiving station to interpret the incoming data stream correctly.

#### Sequence (how a segment must be arranged physically):

The maximum size of a sequence corresponds to the number of enabled Rx or Tx bytes (later: "MTU"). The transmitting station splits the transmit array into valid sequences. These sequences are then written successively to the MTU and transferred to the receiving station where they are lined up together again. The receiver stores the incoming sequences in a receive array, obtaining an image of the data stream in the process.

With Flatstream communication, the number of sequences sent are counted. Successfully transferred sequences must be acknowledged by the receiving station to ensure the integrity of the transfer.

#### MTU (Maximum Transmission Unit) - Physical transport:

MTU refers to the enabled USINT registers used with Flatstream. These registers can accept at least one sequence and transfer it to the receiving station. A separate MTU is defined for each direction of communication. OutputMTU defines the number of Flatstream Tx bytes, and InputMTU specifies the number of Flatstream Rx bytes. The MTUs are transported cyclically via the X2X Link network, increasing the load with each additional enabled USINT register.

#### Properties

Flatstream messages are not transferred cyclically or in 100% real time. Many bus cycles may be needed to transfer a particular message. Although the Rx and Tx registers are exchanged between the transmitter and the receiver cyclically, they are only processed further if explicitly accepted by register "InputSequence" or "OutputSequence".

#### Behavior in the event of an error (brief summary)

The protocol for X2X and POWERLINK networks specifies that the last valid values should be retained when disturbances occur. With conventional communication (cyclic/acyclic data queries), this type of error can generally be ignored.

In order for communication to also take place without errors using Flatstream, all of the sequences issued by the receiver must be acknowledged. If Forward functionality is not used, then subsequent communication is delayed for the length of the disturbance.

If Forward functionality is being used, the receiving station receives a transmission counter that is incremented twice. The receiver stops, i.e. it no longer returns any acknowledgments. The transmitting station uses SequenceAck to determine that the transfer was faulty and that all affected sequences must be repeated.

### 3.7.3 The Flatstream principle

#### Requirements

Before Flatstream can be used, the respective communication direction must be synchronized, i.e. both communication partners cyclically query the sequence counter on the remote station. This checks to see if there is new data that should be accepted.

#### Communication

If a communication partner wants to transmit a message to its remote station, it should first create a transmit array that corresponds to Flatstream conventions. This allows the Flatstream data to be organized very efficiently without having to block other important resources.

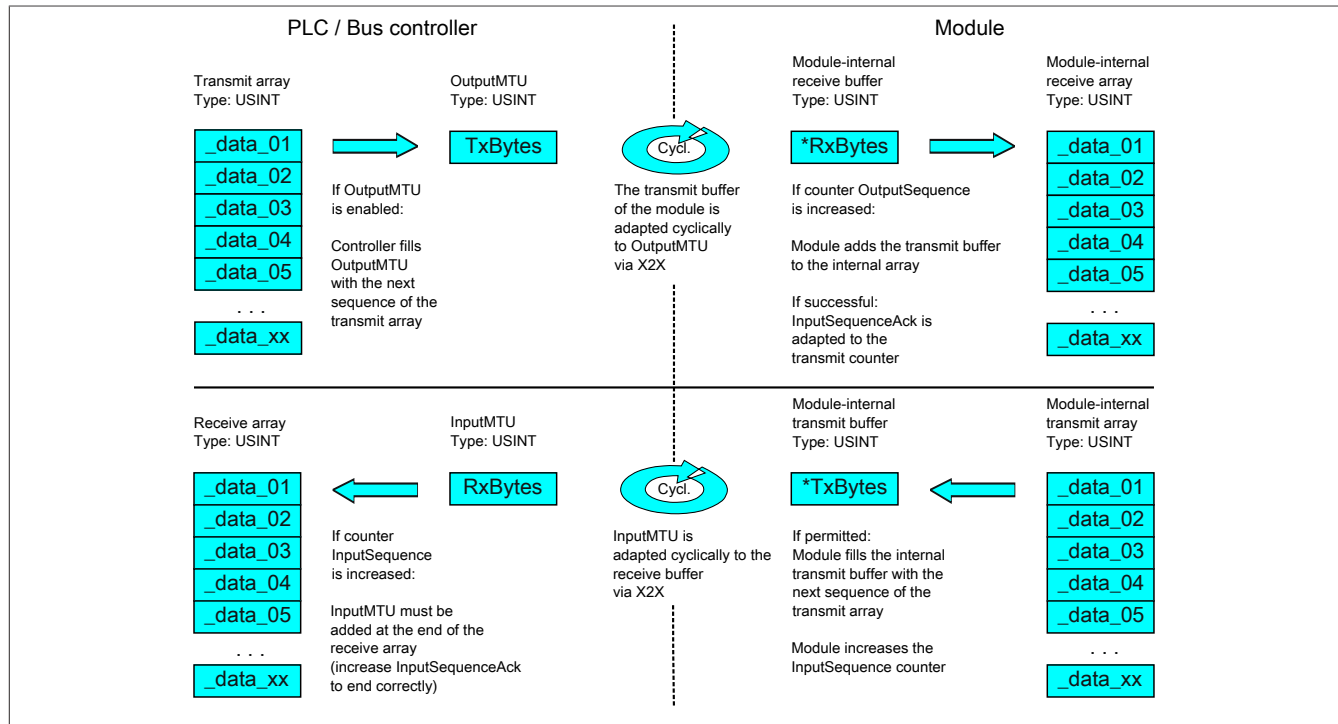


Figure 2: Flatstream communication

#### Procedure

The first thing that happens is that the message is broken into valid segments of up to 63 bytes, and the corresponding control bytes are created. The data is formed into a data stream made up of one control bytes per associated segment. This data stream can be written to the transmit array. The maximum size of each array element matches that of the enabled MTU so that one element corresponds to one sequence. If the array has been completely created, the transmitter checks whether the MTU is permitted to be re-filled. It then copies the first element of the array or the first sequence to the Tx byte registers. The MTU is transported to the receiver station via X2X Link and stored in the corresponding Rx byte registers. To signal that the data should be accepted by the receiver, the transmitter increases its SequenceCounter. If the communication direction is synchronized, the remote station detects the incremented SequenceCounter. The current sequence is appended to the receive array and acknowledged by SequenceAck. This acknowledgment signals to the transmitter that the MTU can now be refilled.

If the transfer is successful, the data in the receive array will correspond 100% to the data in the transmit array. During the transfer, the receiving station must detect and evaluate the incoming control bytes. A separate receive array should be created for each message. This allows the receiver to immediately begin further processing of messages that are completely transferred.

### 3.7.4 Registers for Flatstream mode

5 registers are available for configuring Flatstream. The default configuration can be used to transmit small amounts of data relatively easily.



#### Information:

The controller communicates directly with the field device via registers "OutputSequence" and "InputSequence" as well as the enabled Tx and RxBytes bytes. For this reason, the user must have sufficient knowledge of the communication protocol being used on the field device.

#### 3.7.4.1 Flatstream configuration

To use Flatstream, the program sequence must first be expanded. The cycle time of the Flatstream routines must be set to a multiple of the bus cycle. Other program routines should be implemented in Cyclic #1 to ensure data consistency.

At the absolute minimum, registers "InputMTU" and "OutputMTU" must be set. All other registers are filled in with default values at the beginning and can be used immediately. These registers are used for additional options, e.g. to transfer data in a more compact way or to increase the efficiency of the general procedure.

The Forward registers extend the functionality of the Flatstream protocol. This functionality is useful for substantially increasing the Flatstream data rate, but it also requires quite a bit of extra work when creating the program sequence.



#### Information:

In the rest of this description, the names "OutputMTU" and "InputMTU" do not refer to the registers names. Instead, they are used as synonyms for the currently enabled Tx or Rx bytes.



#### Information:

The registers are described in "[Flatstream registers](#)" on page 84.

Registers are described in section "Flatstream communication" in the respective data sheets.

#### 3.7.4.2 Flatstream operation

When using Flatstream, the communication direction is very important. For transmitting data to a module (output direction), Tx bytes are used. For receiving data from a module (input direction), Rx bytes are used. Registers "OutputSequence" and "InputSequence" are used to control or secure communication, i.e. the transmitter uses them to give instructions to apply data and the receiver confirms a successfully transferred sequence.



#### Information:

The registers are described in "[Flatstream registers](#)" on page 84.

Registers are described in section "Flatstream communication" in the respective data sheets.

##### 3.7.4.2.1 Format of input and output bytes

Name:

"Format of Flatstream" in Automation Studio

On some modules, this function can be used to set how the Flatstream input and output bytes (Tx or Rx bytes) are transferred.

- **Packed:** Data is transferred as an array.
- **Byte-by-byte:** Data is transferred as individual bytes.



### 3.7.4.2.2 Transporting payload data and control bytes

The Tx and Rx bytes are cyclic registers used to transport the payload data and the necessary control bytes. The number of active Tx and Rx bytes is taken from the configuration of registers "OutputMTU" and "InputMTU", respectively.

In the user program, only the Tx and Rx bytes from the controller can be used. The corresponding counterparts are located in the module and are not accessible to the user. For this reason, the names were chosen from the point of view of the controller.

- "T" - "Transmit" → Controller transmits data to the module.
- "R" - "Receive" → Controller receives data from the module.

#### 3.7.4.2.2.1 Control bytes

In addition to the payload data, the Tx and Rx bytes also transfer the necessary control bytes. These control bytes contain additional information about the data stream so that the receiver can reconstruct the original message from the transferred segments.

##### Bit structure of a control byte

Bit	Name	Value	Information
0 - 5	SegmentLength	0 - 63	Size of the subsequent segment in bytes (default: Max. MTU size - 1)
6	nextCBPos	0	Next control byte at the beginning of the next MTU
		1	Next control byte directly after the end of the current segment
7	MessageEndBit	0	Message continues after the subsequent segment
		1	Message ended by the subsequent segment

##### SegmentLength

The segment length lets the receiver know the length of the coming segment. If the set segment length is insufficient for a message, then the information must be distributed over several segments. In these cases, the actual end of the message is detected using bit 7 (control byte).



#### Information:

**The control byte is not included in the calculation to determine the segment length. The segment length is only derived from the bytes of payload data.**

##### nextCBPos

This bit indicates the position where the next control byte is expected. This information is especially important when using option "MultiSegmentMTU".

When using Flatstream communication with MultiSegmentMTUs, the next control byte is no longer expected in the first Rx byte of the subsequent MTU, but transferred directly after the current segment.

##### MessageEndBit

"MessageEndBit" is set if the subsequent segment completes a message. The message has then been completely transferred and is ready for further processing.



#### Information:

**In the output direction, this bit must also be set if one individual segment is enough to hold the entire message. The module will only process a message internally if this identifier is detected.**

**The size of the message being transferred can be calculated by adding all of the message's segment lengths together.**

Flatstream formula for calculating message length:

Message [bytes] = Segment lengths (all CBs without ME) + Segment length (of the first CB with ME)	CB	Control byte
	ME	MessageEndBit

Function description

3.7.4.2.3 Communication status

The communication status is determined via registers "OutputSequence" and "InputSequence".

- OutputSequence contains information about the communication status of the controller. It is written by the controller and read by the module.
- InputSequence contains information about the communication status of the module. It is written by the module and should only be read by the controller.

3.7.4.2.3.1 Relationship between OutputSequence and InputSequence

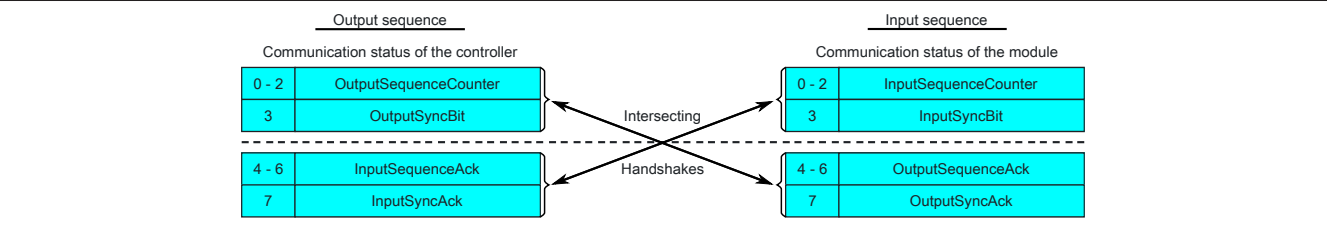


Figure 3: Relationship between OutputSequence and InputSequence

Registers OutputSequence and InputSequence are logically composed of 2 half-bytes. The low part indicates to the remote station whether a channel should be opened or whether data should be accepted. The high part is to acknowledge that the requested action was carried out.

SyncBit and SyncAck

If SyncBit and SyncAck are set in one communication direction, then the channel is considered "synchronized", i.e. it is possible to send messages in this direction. The status bit of the remote station must be checked cyclically. If SyncAck has been reset, then SyncBit on that station must be adjusted. Before new data can be transferred, the channel must be resynchronized.

SequenceCounter and SequenceAck

The communication partners cyclically check whether the low nibble on the remote station changes. When one of the communication partners finishes writing a new sequence to the MTU, it increments its SequenceCounter. The current sequence is then transmitted to the receiver, which acknowledges its receipt with SequenceAck. In this way, a "handshake" is initiated.



Information:

If communication is interrupted, segments from the unfinished message are discarded. All messages that were transferred completely are processed.

### 3.7.4.3 Synchronization

During synchronization, a communication channel is opened. It is important to make sure that a module is present and that the current value of SequenceCounter is stored on the station receiving the message. Flatstream can handle full-duplex communication. This means that both channels / communication directions can be handled separately. They must be synchronized independently so that simplex communication can theoretically be carried out as well.

#### Synchronization in the output direction (controller as the transmitter):

The corresponding synchronization bits (OutputSyncBit and OutputSyncAck) are reset. Because of this, Flatstream cannot be used at this point in time to transfer messages from the controller to the module.

#### Algorithm

1) The controller must write 000 to OutputSequenceCounter and reset OutputSyncBit. The controller must cyclically query the high nibble of register "InputSequence" (checks for 000 in OutputSequenceAck and 0 in OutputSyncAck). The module does not accept the current contents of InputMTU since the channel is not yet synchronized. The module matches OutputSequenceAck and OutputSyncAck to the values of OutputSequenceCounter and OutputSyncBit.
2) If the controller registers the expected values in OutputSequenceAck and OutputSyncAck, it is permitted to increment OutputSequenceCounter. The controller continues cyclically querying the high nibble of register "OutputSequence" (checks for 001 in OutputSequenceAck and 0 in InputSyncAck). The module does not accept the current contents of InputMTU since the channel is not yet synchronized. The module matches OutputSequenceAck and OutputSyncAck to the values of OutputSequenceCounter and OutputSyncBit.
3) If the controller registers the expected values in OutputSequenceAck and OutputSyncAck, it is permitted to increment OutputSequenceCounter. The controller continues cyclically querying the high nibble of register "OutputSequence" (checks for 001 in OutputSequenceAck and 1 in InputSyncAck).  <b>Note:</b> Theoretically, data can be transferred from this point forward. However, it is still recommended to wait until the output direction is completely synchronized before transferring data. The module sets OutputSyncAck. The output direction is synchronized, and the controller can transmit data to the module.

#### Synchronization in the input direction (controller as the receiver):

The corresponding synchronization bits (InputSyncBit and InputSyncAck) are reset. Because of this, Flatstream cannot be used at this point in time to transfer messages from the module to the controller.

#### Algorithm

The module writes 000 to InputSequenceCounter and resets InputSyncBit. The module monitors the high nibble of register "OutputSequence" and expects 000 in InputSequenceAck and 0 in InputSyncAck.
1) The controller is not permitted to accept the current contents of InputMTU since the channel is not yet synchronized. The controller must match InputSequenceAck and InputSyncAck to the values of InputSequenceCounter and InputSyncBit. If the module registers the expected values in InputSequenceAck and InputSyncAck, it increments InputSequenceCounter. The module monitors the high nibble of register "OutputSequence" and expects 001 in InputSequenceAck and 0 in InputSyncAck.
2) The controller is not permitted to accept the current contents of InputMTU since the channel is not yet synchronized. The controller must match InputSequenceAck and InputSyncAck to the values of InputSequenceCounter and InputSyncBit. If the module registers the expected values in InputSequenceAck and InputSyncAck, it sets InputSyncBit. The module monitors the high nibble of register "OutputSequence" and expects 1 in InputSyncAck.
3) The controller is permitted to set InputSyncAck.  <b>Note:</b> Theoretically, data could already be transferred in this cycle. If InputSyncBit is set and InputSequenceCounter has been increased by 1, the values in the enabled Rx bytes must be accepted and acknowledged (see also "Communication in the input direction"). The input direction is synchronized, and the module can transmit data to the controller.

Function description

3.7.4.4 Transmitting and receiving

If a channel is synchronized, then the remote station is ready to receive messages from the transmitter. Before the transmitter can send data, it must first create a transmit array in order to meet Flatstream requirements.

The transmitting station must also generate a control byte for each segment created. This control byte contains information about how the subsequent part of the data being transferred should be processed. The position of the next control byte in the data stream can vary. For this reason, it must be clearly defined at all times when a new control byte is being transmitted. The first control byte is always in the first byte of the first sequence. All subsequent positions are determined recursively.

Flatstream formula for calculating the position of the next control byte:

Position (of the next control byte) = Current position + 1 + Segment length

Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The rest of the configuration corresponds to the default settings.

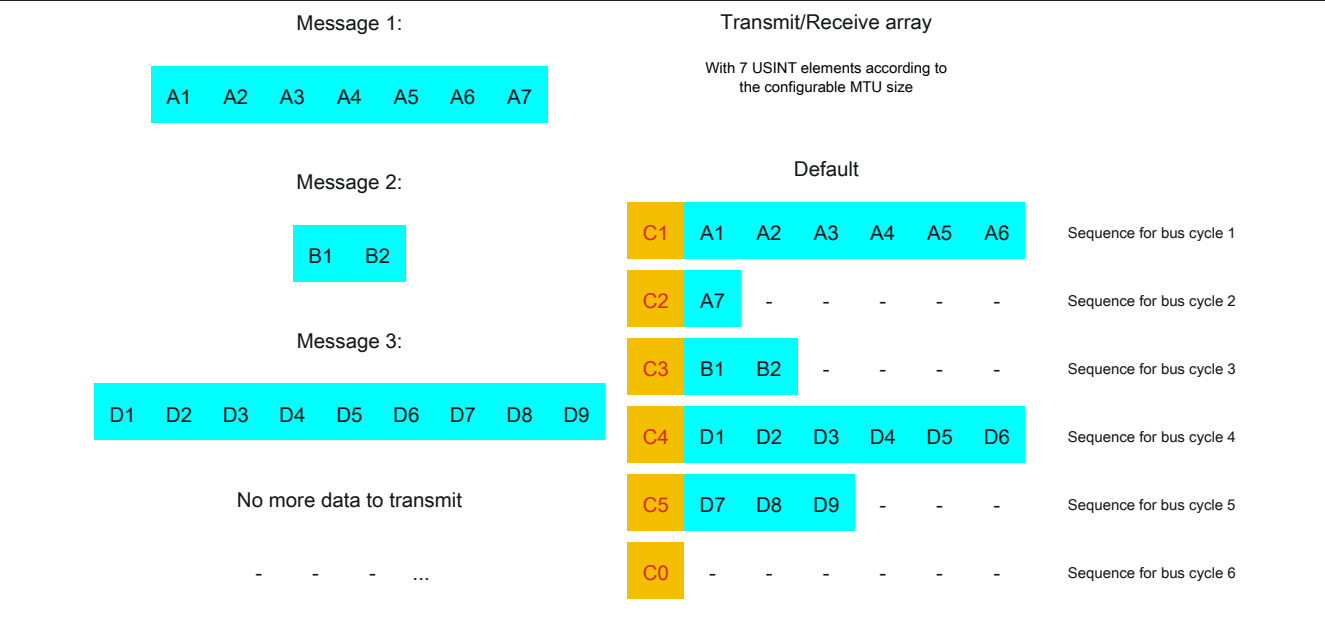


Figure 4: Transmit/Receive array (default)

The messages must first be split into segments. In the default configuration, it is important to ensure that each sequence can hold an entire segment, including the associated control byte. The sequence is limited to the size of the enable MTU. In other words, a segment must be at least 1 byte smaller than the MTU.

MTU = 7 bytes → Max. segment length = 6 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 6 bytes of data
  - ⇒ Second segment = Control byte + 1 data byte
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 6 bytes of data
  - ⇒ Second segment = Control byte + 3 data bytes
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C0 (control byte 0)			C1 (control byte 1)			C2 (control byte 2)		
- SegmentLength (0)	=	0	- SegmentLength (6)	=	6	- SegmentLength (1)	=	1
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (0)	=	0	- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	128
Control byte	Σ	0	Control byte	Σ	6	Control byte	Σ	129

Table 3: Flatstream determination of the control bytes for the default configuration example (part 1)

C3 (control byte 3)			C4 (control byte 4)			C5 (control byte 5)		
- SegmentLength (2)	=	2	- SegmentLength (6)	=	6	- SegmentLength (3)	=	3
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (1)	=	128	- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	128
Control byte	Σ	130	Control byte	Σ	6	Control byte	Σ	131

Table 4: Flatstream determination of the control bytes for the default configuration example (part 2)

### 3.7.4.4.1 Transmitting data to a module (output)

When transmitting data, the transmit array must be generated in the application program. Sequences are then transferred one by one using Flatstream and received by the module.



#### Information:

Although all B&R modules with Flatstream communication always support the most compact transfers in the output direction, it is recommended to use the same design for the transfer arrays in both communication directions.

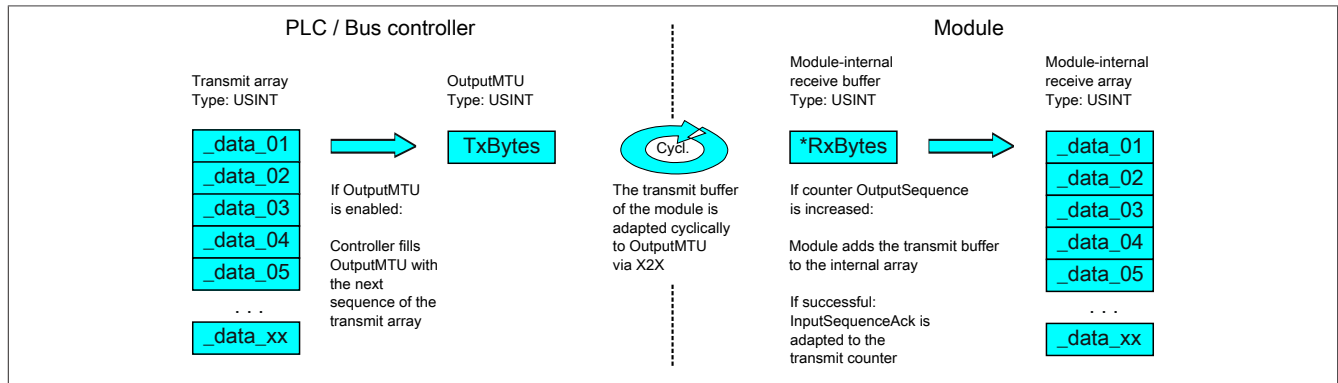


Figure 5: Flatstream communication (output)

#### Message smaller than OutputMTU

The length of the message is initially smaller than OutputMTU. In this case, one sequence would be sufficient to transfer the entire message and the necessary control byte.

#### Algorithm

Cyclic status query: - The module monitors OutputSequenceCounter.
0) Cyclic checks: - The controller must check OutputSyncAck. → If OutputSyncAck = 0: Reset OutputSyncBit and resynchronize the channel. - The controller must check whether OutputMTU is enabled. → If OutputSequenceCounter > InputSequenceAck: MTU is not enabled because the last sequence has not yet been acknowledged.
1) Preparation (create transmit array): - The controller must split up the message into valid segments and create the necessary control bytes. - The controller must add the segments and control bytes to the transmit array.
2) Transmit: - The controller transfers the current element of the transmit array to OutputMTU. → OutputMTU is transferred cyclically to the module's transmit buffer but not processed further. - The controller must increase OutputSequenceCounter.
Reaction: - The module accepts the bytes from the internal receive buffer and adds them to the internal receive array. - The module transmits acknowledgment and writes the value of OutputSequenceCounter to OutputSequenceAck.
3) Completion: - The controller must monitor OutputSequenceAck. → A sequence is only considered to have been transferred successfully if it has been acknowledged via OutputSequenceAck. In order to detect potential transfer errors in the last sequence as well, it is important to make sure that the length of the Completion phase is run through long enough.
<b>Note:</b> To monitor communication times exactly, the task cycles that have passed since the last increase of OutputSequenceCounter should be counted. In this way, the number of previous bus cycles necessary for the transfer can be measured. If the monitoring counter exceeds a predefined threshold, then the sequence can be considered lost. (The relationship of bus to task cycle can be influenced by the user so that the threshold value must be determined individually.) - Subsequent sequences are only permitted to be transmitted in the next bus cycle after the completion check has been carried out successfully.

### Message larger than OutputMTU

The transmit array, which must be created in the program sequence, consists of several elements. The user must arrange the control and data bytes correctly and transfer the array elements one after the other. The transfer algorithm remains the same and is repeated starting at the point *Cyclic checks*.

#### General flowchart

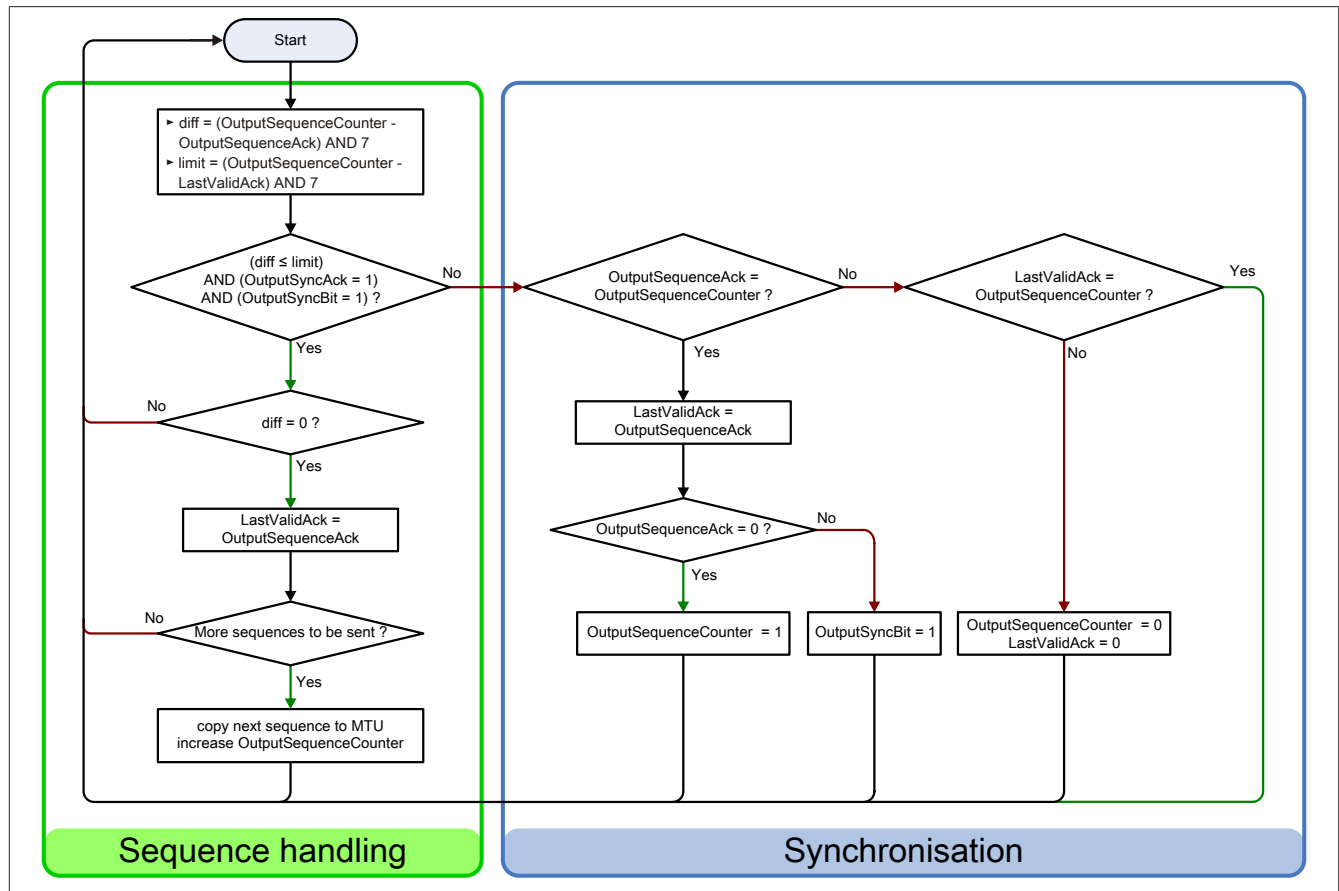


Figure 6: Flowchart for the output direction

Function description

3.7.4.4.2 Receiving data from a module (input)

When receiving data, the transmit array is generated by the module, transferred via Flatstream and must then be reproduced in the receive array. The structure of the incoming data stream can be set with the mode register. The algorithm for receiving the data remains unchanged in this regard.

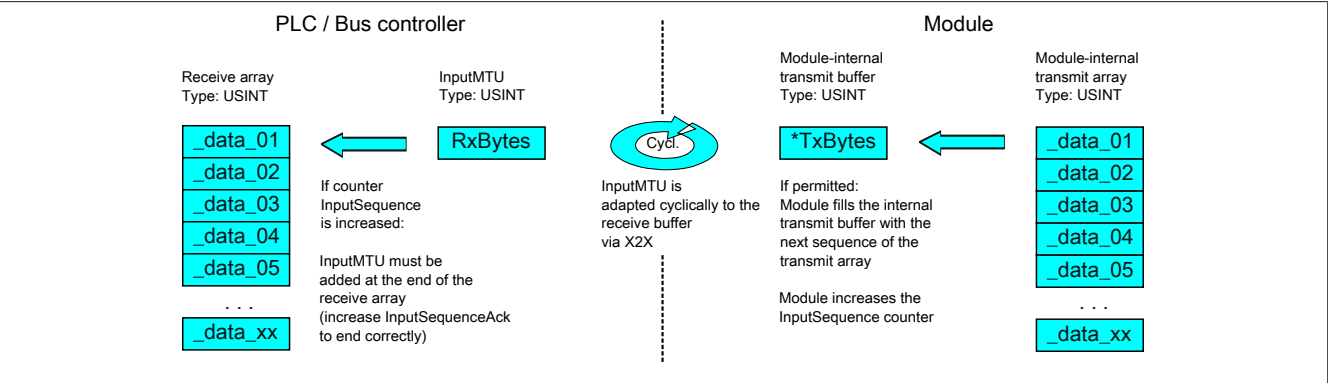


Figure 7: Flatstream communication (input)

Algorithm

0) Cyclic status query: - The controller must monitor InputSequenceCounter.
Cyclic checks: - The module checks InputSyncAck. - The module checks InputSequenceAck.
Preparation: - The module forms the segments and control bytes and creates the transmit array.
Action: - The module transfers the current element of the internal transmit array to the internal transmit buffer. - The module increases InputSequenceCounter.
1) Receiving (as soon as InputSequenceCounter is increased): - The controller must apply data from InputMTU and append it to the end of the receive array. - The controller must match InputSequenceAck to InputSequenceCounter of the sequence currently being processed.
Completion: - The module monitors InputSequenceAck. → A sequence is only considered to have been transferred successfully if it has been acknowledged via InputSequenceAck. - Subsequent sequences are only transmitted in the next bus cycle after the completion check has been carried out successfully.



## General flowchart

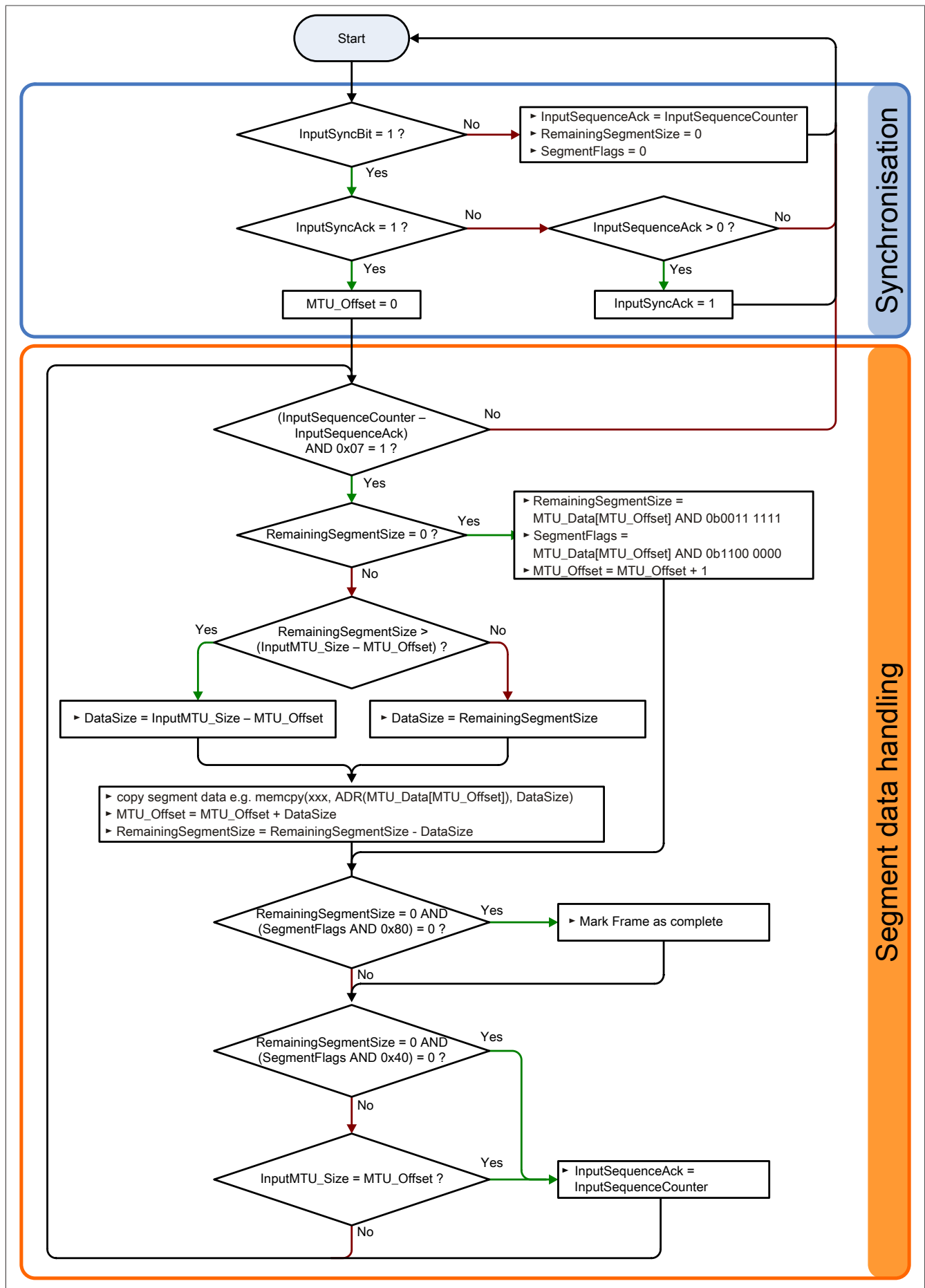


Figure 8: Flowchart for the input direction

### 3.7.4.4.3 Details

**It is recommended to store transferred messages in separate receive arrays.**

After a set MessageEndBit is transmitted, the subsequent segment should be added to the receive array. The message is then complete and can be passed on internally for further processing. A new/separate array should be created for the next message.



**Information:**

When transferring with MultiSegmentMTUs, it is possible for several small messages to be part of one sequence. In the program, it is important to make sure that a sufficient number of receive arrays can be managed. The acknowledge register is only permitted to be adjusted after the entire sequence has been applied.

**If SequenceCounter is incremented by more than one counter, an error is present.**

In this case, the receiver stops. All additional incoming sequences are ignored until the transmission with the correct SequenceCounter is retried. This response prevents the transmitter from receiving any more acknowledgments for transmitted sequences. The transmitter can identify the last successfully transferred sequence from the remote station's SequenceAck and continue the transfer from this point.



**Information:**

This situation is very unlikely when operating without "Forward" functionality.

**Acknowledgments must be checked for validity.**

If the receiver has successfully accepted a sequence, it must be acknowledged. The receiver takes on the value of SequenceCounter sent along with the transmission and matches SequenceAck to it. The transmitter reads SequenceAck and registers the successful transmission. If the transmitter acknowledges a sequence that has not yet been dispatched, then the transfer must be interrupted and the channel resynchronized. The synchronization bits are reset and the current/incomplete message is discarded. It must be sent again after the channel has been resynchronized.

### 3.7.4.5 Flatstream mode

In the input direction, the transmit array is generated automatically. Flatstream mode offers several options to the user that allow an incoming data stream to have a more compact arrangement. These include:

- [Standard](#)
- [MultiSegmentMTU allowed](#)
- [Large segments allowed:](#)

Once enabled, the program code for evaluation must be adapted accordingly.



#### Information:

**All B&R modules that offer Flatstream mode support options "Large segments" and "MultiSegmentMTU" in the output direction. Compact transfer must be explicitly allowed only in the input direction.**

#### Standard

By default, both options relating to compact transfer in the input direction are disabled.

1. The module only forms segments that are at least one byte smaller than the enabled MTU. Each sequence begins with a control byte so that the data stream is clearly structured and relatively easy to evaluate.
2. Since a Flatstream message is permitted to be any length, the last segment of the message frequently does not fill up all of the MTU's space. By default, the remaining bytes during this type of transfer cycle are not used.

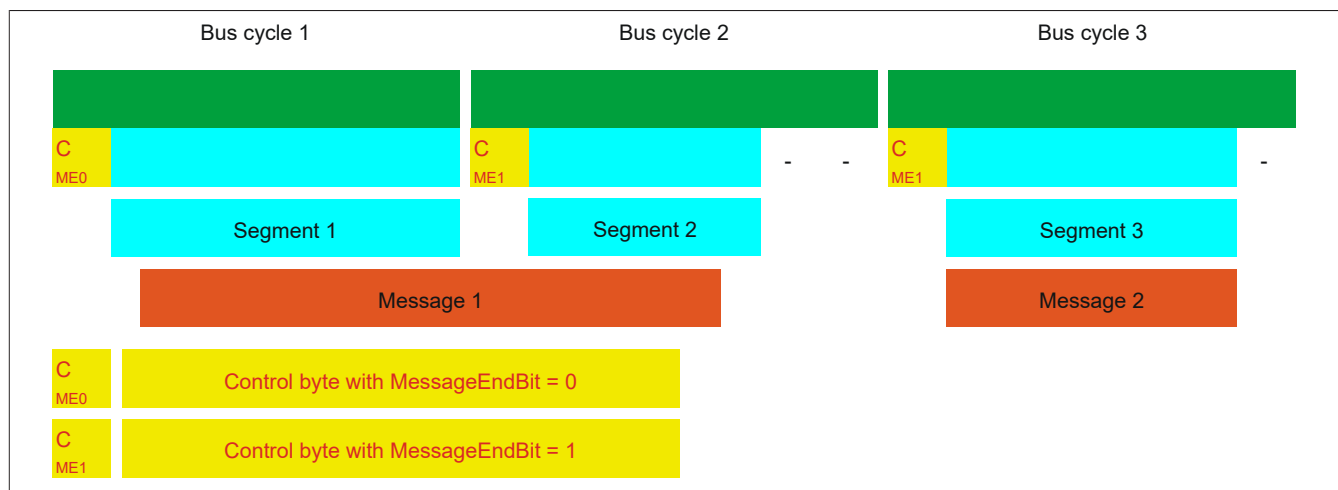


Figure 9: Message arrangement in the MTU (default)

Function description

MultiSegmentMTU allowed

With this option, InputMTU is completely filled (if enough data is pending). The previously unfilled Rx bytes transfer the next control bytes and their segments. This allows the enabled Rx bytes to be used more efficiently.

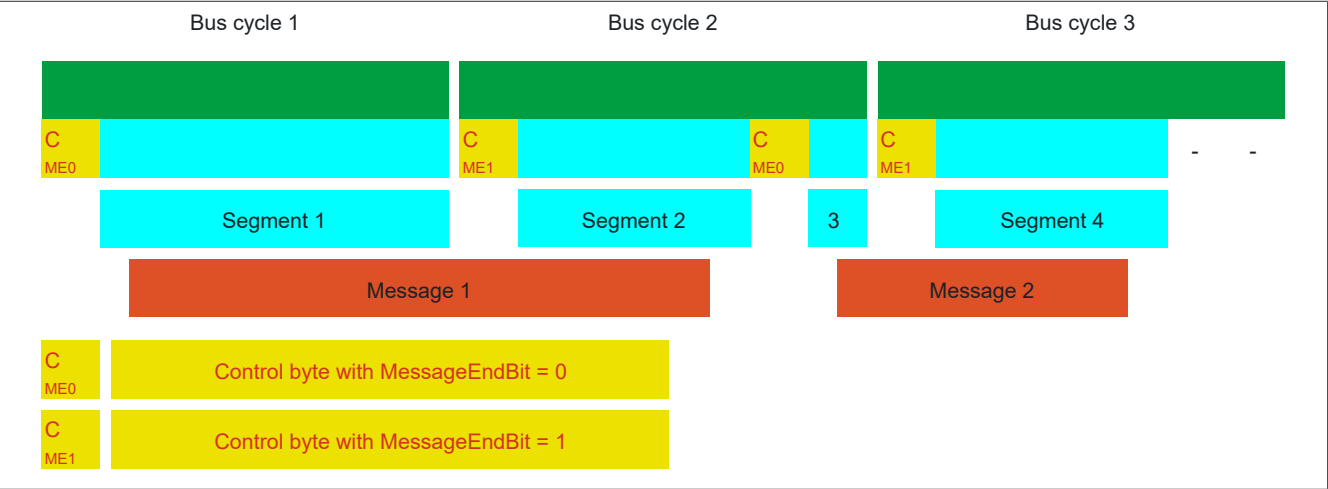


Figure 10: Arrangement of messages in the MTU (MultiSegmentMTU)

Large segments allowed:

When transferring very long messages or when enabling only very few Rx bytes, then a great many segments must be created by default. The bus system is more stressed than necessary since an additional control byte must be created and transferred for each segment. With option "Large segments", the segment length is limited to 63 bytes independently of InputMTU. One segment is permitted to stretch across several sequences, i.e. it is possible for "pure" sequences to occur without a control byte.

**Information:**

It is still possible to split up a message into several segments, however. If this option is used and messages with more than 63 bytes occur, for example, then messages can still be split up among several segments.

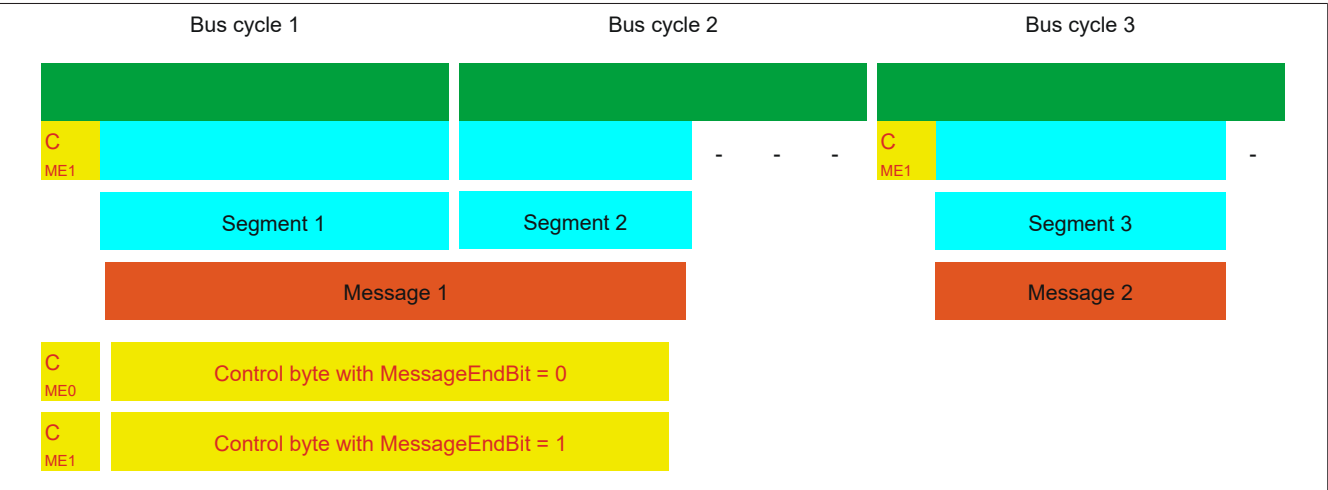


Figure 11: Arrangement of messages in the MTU (large segments)

### Using both options

Using both options at the same time is also permitted.

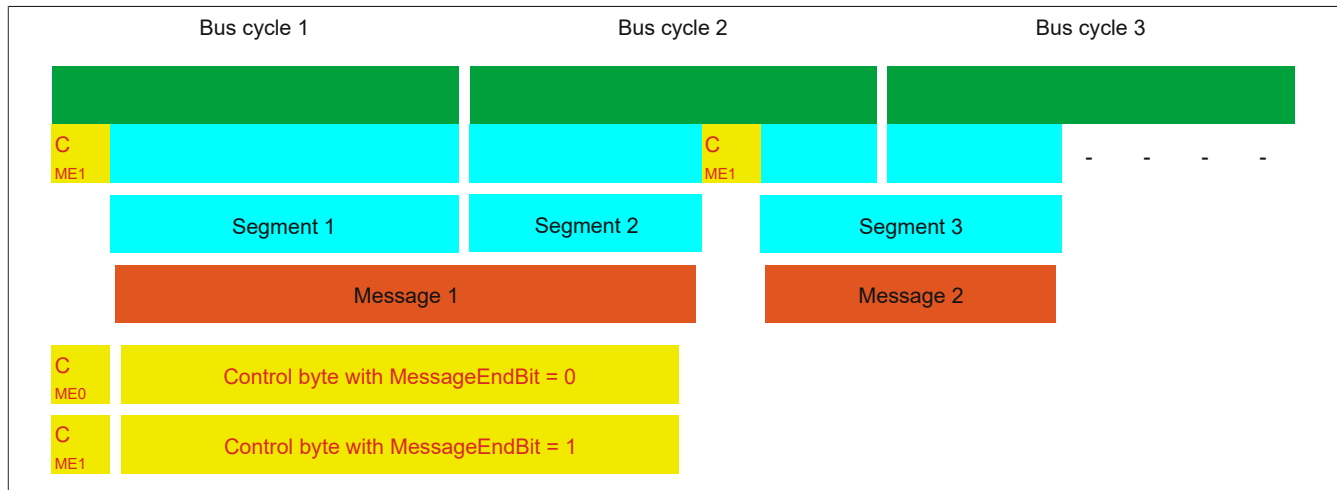


Figure 12: Arrangement of messages in the MTU (large segments and MultiSegmentMTU)

### 3.7.4.6 Adjusting the Flatstream

If the way messages are structured is changed, then the way data in the transmit/receive array is arranged is also different. The following changes apply to the example given earlier.

#### MultiSegmentMTU

If MultiSegmentMTUs are allowed, then "open positions" in an MTU can be used. These "open positions" occur if the last segment in a message does not fully use the entire MTU. MultiSegmentMTUs allow these bits to be used to transfer the subsequent control bytes and segments. In the program sequence, the "nextCB-Pos" bit in the control byte is set so that the receiver can correctly identify the next control byte.

#### Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The configuration allows the transfer of MultiSegmentMTUs.

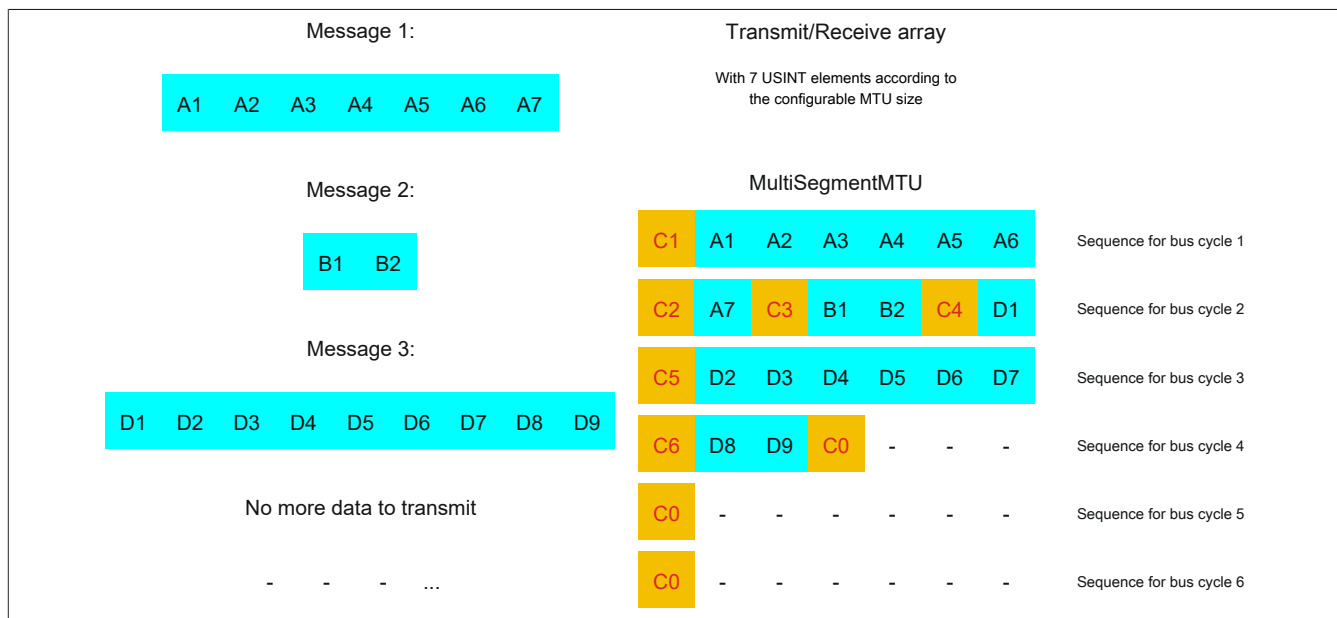


Figure 13: Transmit/Receive array (MultiSegmentMTU)

## Function description

The messages must first be split into segments. As in the default configuration, it is important for each sequence to begin with a control byte. The free bits in the MTU at the end of a message are filled with data from the following message, however. With this option, the "nextCBPos" bit is always set if payload data is transferred after the control byte.

MTU = 7 bytes → Max. segment length = 6 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 6 bytes of data (MTU full)
  - ⇒ Second segment = Control byte + 1 byte of data (MTU still has 5 open bytes)
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data (MTU still has 2 open bytes)
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 1 byte of data (MTU full)
  - ⇒ Second segment = Control byte + 6 bytes of data (MTU full)
  - ⇒ Third segment = Control byte + 2 bytes of data (MTU still has 4 open bytes)
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C1 (control byte 1)			C2 (control byte 2)			C3 (control byte 3)		
- SegmentLength (6)	=	6	- SegmentLength (1)	=	1	- SegmentLength (2)	=	2
- nextCBPos (1)	=	64	- nextCBPos (1)	=	64	- nextCBPos (1)	=	64
- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128
Control byte	Σ	70	Control byte	Σ	193	Control byte	Σ	194

Table 5: Flatstream determination of the control bytes for the MultiSegmentMTU example (part 1)



### Warning!

The second sequence is only permitted to be acknowledged via SequenceAck if it has been completely processed. In this example, there are 3 different segments within the second sequence, i.e. the program must include enough receive arrays to handle this situation.



### Mise en garde !

La deuxième séquence ne peut être acquittée via SequenceAck que si elle a été entièrement traitée. Dans cet exemple, il y a 3 segments différents dans la deuxième séquence, c'est-à-dire que le programme doit inclure suffisamment de tableaux de réception pour gérer cette situation.

C4 (control byte 4)			C5 (control byte 5)			C6 (control byte 6)		
- SegmentLength (1)	=	1	- SegmentLength (6)	=	6	- SegmentLength (2)	=	2
- nextCBPos (6)	=	6	- nextCBPos (1)	=	64	- nextCBPos (1)	=	64
- MessageEndBit (0)	=	0	- MessageEndBit (1)	=	0	- MessageEndBit (1)	=	128
Control byte	Σ	7	Control byte	Σ	70	Control byte	Σ	194

Table 6: Flatstream determination of the control bytes for the MultiSegmentMTU example (part 2)

## Large segments

Segments are limited to a maximum of 63 bytes. This means they can be larger than the active MTU. These large segments are divided among several sequences when transferred. It is possible for sequences to be completely filled with payload data and not have a control byte.



### Information:

**It is still possible to subdivide a message into several segments so that the size of a data packet does not also have to be limited to 63 bytes.**

### Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The configuration allows the transfer of large segments.

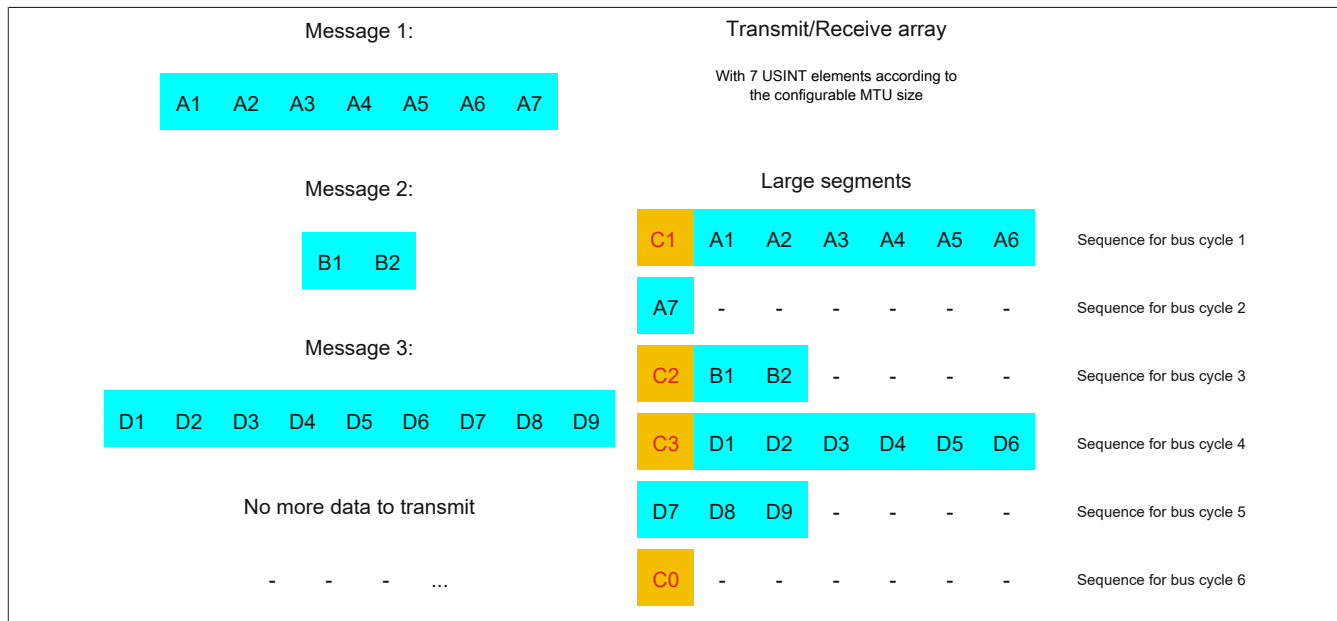


Figure 14: Transmit/receive array (large segments)

The messages must first be split into segments. The ability to form large segments means that messages are split up less frequently, which results in fewer control bytes generated.

Large segments allowed → Max. segment length = 63 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 7 bytes of data
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 9 bytes of data
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C1 (control byte 1)			C2 (control byte 2)			C3 (control byte 3)		
- SegmentLength (7)	=	7	- SegmentLength (2)	=	2	- SegmentLength (9)	=	9
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128
Control byte	Σ	135	Control byte	Σ	130	Control byte	Σ	137

Table 7: Flatstream determination of the control bytes for the large segment example

## Function description

### Large segments and MultiSegmentMTU

#### Example

3 autonomous messages (7 bytes, 2 bytes and 9 bytes) are being transmitted using an MTU with a width of 7 bytes. The configuration allows transfer of large segments as well as MultiSegmentMTUs.

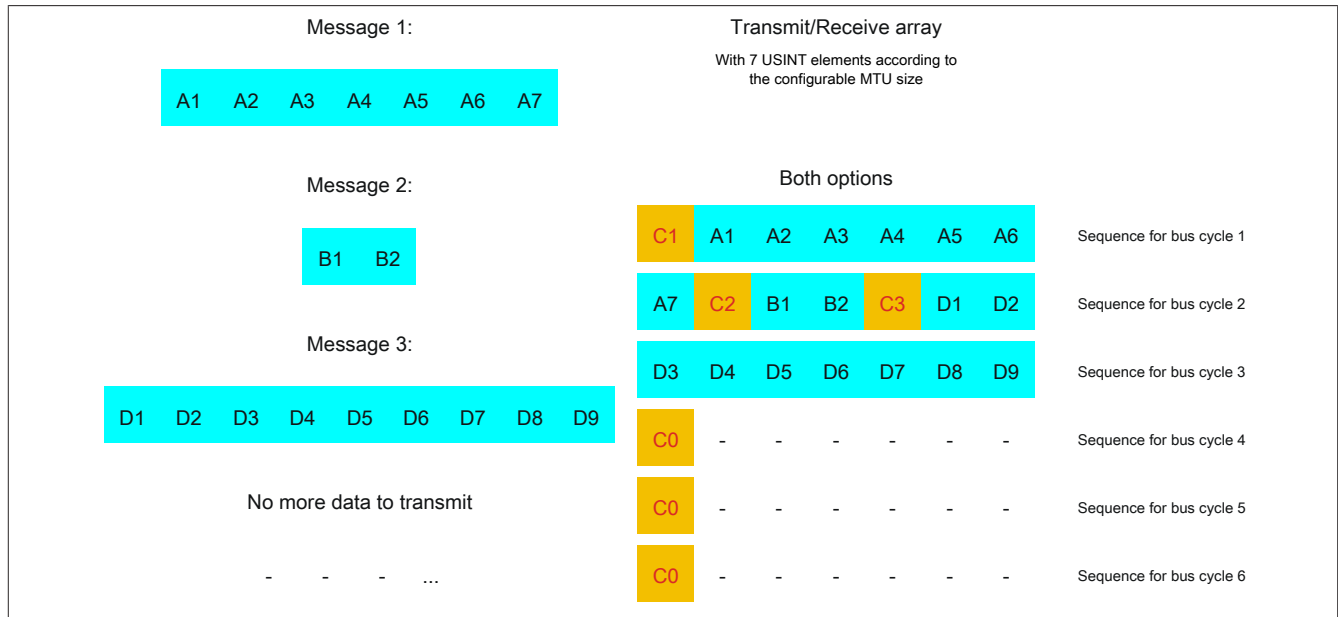


Figure 15: Transmit/Receive array (large segments and MultiSegmentMTU)

The messages must first be split into segments. If the last segment of a message does not completely fill the MTU, it is permitted to be used for other data in the data stream. Bit "nextCBPos" must always be set if the control byte belongs to a segment with payload data.

The ability to form large segments means that messages are split up less frequently, which results in fewer control bytes generated. Control bytes are generated in the same way as with option "Large segments".

Large segments allowed → Max. segment length = 63 bytes

- Message 1 (7 bytes)
  - ⇒ First segment = Control byte + 7 bytes of data
- Message 2 (2 bytes)
  - ⇒ First segment = Control byte + 2 bytes of data
- Message 3 (9 bytes)
  - ⇒ First segment = Control byte + 9 bytes of data
- No more messages
  - ⇒ C0 control byte

A unique control byte must be generated for each segment. In addition, the C0 control byte is generated to keep communication on standby.

C1 (control byte 1)			C2 (control byte 2)			C3 (control byte 3)		
- SegmentLength (7)	=	7	- SegmentLength (2)	=	2	- SegmentLength (9)	=	9
- nextCBPos (0)	=	0	- nextCBPos (0)	=	0	- nextCBPos (0)	=	0
- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128	- MessageEndBit (1)	=	128
Control byte	Σ	135	Control byte	Σ	130	Control byte	Σ	137

Table 8: Flatstream determination of the control bytes for the large segment and MultiSegmentMTU example



### 3.7.5 Example of function "Forward" with X2X Link

Function "Forward" is a method that can be used to substantially increase the Flatstream data rate. The basic principle is also used in other technical areas such as "pipelining" for microprocessors.

#### 3.7.5.1 Function principle

X2X Link communication cycles through 5 different steps to transfer a Flatstream sequence. At least 5 bus cycles are therefore required to successfully transfer the sequence.

	Step I	Step II	Step III	Step IV	Step V
<b>Actions</b>	Transfer sequence from transmit array, increase Sequence-Counter	Cyclic synchronization of MTU and module buffer	Append sequence to receive array, adjust SequenceAck	Cyclic synchronization MTU and module buffer	Check SequenceAck
<b>Resource</b>	Transmitter (task to transmit)	Bus system (direction 1)	Recipients (task to receive)	Bus system (direction 2)	Transmitter (task for Ack checking)

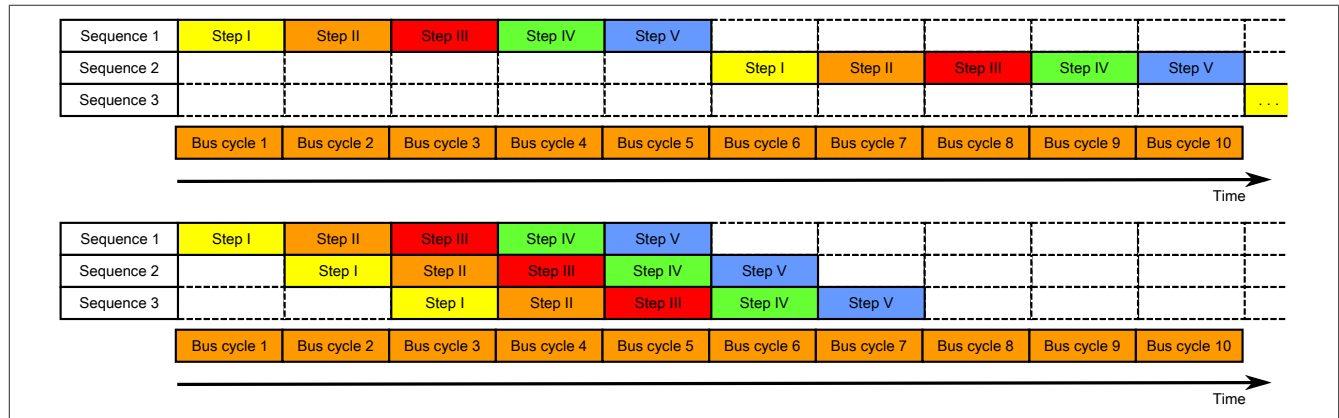



Figure 16: Comparison of transfer without/with Forward

Each of the 5 steps (tasks) requires different resources. If Forward functionality is not used, the sequences are executed one after the other. Each resource is then only active if it is needed for the current sub-action. With Forward, a resource that has executed its task can already be used for the next message. The condition for enabling the MTU is changed to allow for this. Sequences are then passed to the MTU according to the timing. The transmitting station no longer waits for an acknowledgment from SequenceAck, which means that the available bandwidth can be used much more efficiently.

In the most ideal situation, all resources are working during each bus cycle. The receiver must still acknowledge every sequence received. Only when SequenceAck has been changed and checked by the transmitter is the sequence considered as having been transferred successfully.

3.7.5.2 Configuration

The Forward function must only be enabled for the input direction. Flatstream modules have been optimized in such a way that they support this function. In the output direction, the Forward function can be used as soon as the size of OutputMTU is specified.



**Information:**

The registers are described in ["Flatstream registers" on page 84](#).

Registers are described in section ["Flatstream communication"](#) in the respective data sheets.

3.7.5.2.1 Delay time

The delay time is specified in microseconds. This is the amount of time the module must wait after sending a sequence until it is permitted to write new data to the MTU in the following bus cycle. The program routine for receiving sequences from a module can therefore be run in a task class whose cycle time is slower than the bus cycle.

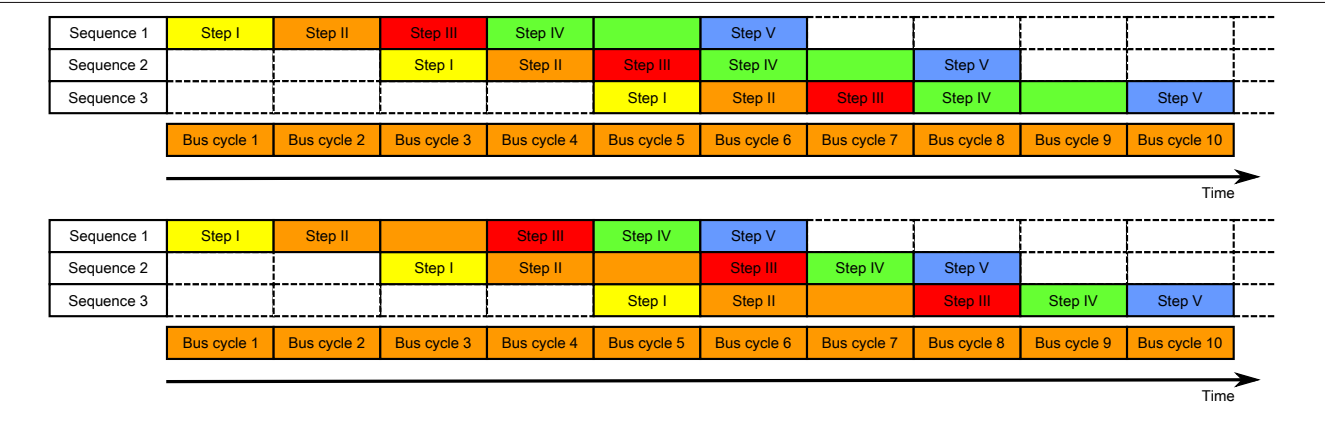


Figure 17: Effect of ForwardDelay when using Flatstream communication with the Forward function

In the program, it is important to make sure that the controller is processing all of the incoming InputSequences and InputMTUs. The ForwardDelay value causes delayed acknowledgment in the output direction and delayed reception in the input direction. In this way, the controller has more time to process the incoming InputSequence or InputMTU.

### 3.7.5.3 Transmitting and receiving with Forward

The basic algorithm for transmitting and receiving data remains the same. With the Forward function, up to 7 unacknowledged sequences can be transmitted. Sequences can be transmitted without having to wait for the previous message to be acknowledged. Since the delay between writing and response is eliminated, a considerable amount of additional data can be transferred in the same time window.

#### Algorithm for transmitting

Cyclic status query: - The module monitors OutputSequenceCounter.
0) Cyclic checks: - The controller must check OutputSyncAck. → If OutputSyncAck = 0: Reset OutputSyncBit and resynchronize the channel. - The controller must check whether OutputMTU is enabled. → If OutputSequenceCounter > OutputSequenceAck + 7, then it is not enabled because the last sequence has not yet been acknowledged.
1) Preparation (create transmit array): - The controller must split up the message into valid segments and create the necessary control bytes. - The controller must add the segments and control bytes to the transmit array.
2) Transmit: - The controller must transfer the current part of the transmit array to OutputMTU. - The controller must increase OutputSequenceCounter for the sequence to be accepted by the module. - The controller is then permitted to transmit in the next bus cycle if the MTU has been enabled.
The module responds since OutputSequenceCounter > OutputSequenceAck: - The module accepts data from the internal receive buffer and appends it to the end of the internal receive array. - The module is acknowledged and the currently received value of OutputSequenceCounter is transferred to OutputSequenceAck. - The module queries the status cyclically again.
3) Completion (acknowledgment): - The controller must check OutputSequenceAck cyclically. → A sequence is only considered to have been transferred successfully if it has been acknowledged via OutputSequenceAck. In order to detect potential transfer errors in the last sequence as well, it is important to make sure that the algorithm is run through long enough.  <b>Note:</b> To monitor communication times exactly, the task cycles that have passed since the last increase of OutputSequenceCounter should be counted. In this way, the number of previous bus cycles necessary for the transfer can be measured. If the monitoring counter exceeds a predefined threshold, then the sequence can be considered lost (the relationship of bus to task cycle can be influenced by the user so that the threshold value must be determined individually).

#### Algorithm for receiving

0) Cyclic status query: - The controller must monitor InputSequenceCounter.
Cyclic checks: - The module checks InputSyncAck. - The module checks if InputMTU for enabling. → Enabling criteria: InputSequenceCounter > InputSequenceAck + Forward
Preparation: - The module forms the control bytes / segments and creates the transmit array.
Action: - The module transfers the current part of the transmit array to the receive buffer. - The module increases InputSequenceCounter. - The module waits for a new bus cycle after time from ForwardDelay has expired. - The module repeats the action if InputMTU is enabled.
1) Receiving (InputSequenceCounter > InputSequenceAck): - The controller must apply data from InputMTU and append it to the end of the receive array. - The controller must match InputSequenceAck to InputSequenceCounter of the sequence currently being processed.
Completion: - The module monitors InputSequenceAck. → A sequence is only considered to have been transferred successfully if it has been acknowledged via InputSequenceAck.

### Details/Background

1. Illegal SequenceCounter size (counter offset)

Error situation: MTU not enabled

If the difference between SequenceCounter and SequenceAck during transmission is larger than permitted, a transfer error occurs. In this case, all unacknowledged sequences must be repeated with the old SequenceCounter value.

2. Checking an acknowledgment

After an acknowledgment has been received, a check must verify whether the acknowledged sequence has been transmitted and had not yet been unacknowledged. If a sequence is acknowledged multiple times, a severe error occurs. The channel must be closed and resynchronized (same behavior as when not using Forward).



#### Information:

**In exceptional cases, the module can increment OutputSequenceAck by more than 1 when using Forward.**

**An error does not occur in this case. The controller is permitted to consider all sequences up to the one being acknowledged as having been transferred successfully.**

3. Transmit and receive arrays

The Forward function has no effect on the structure of the transmit and receive arrays. They are created and must be evaluated in the same way.

### 3.7.5.4 Errors when using Forward

In industrial environments, it is often the case that many different devices from various manufacturers are being used side by side. The electrical and/or electromagnetic properties of these technical devices can sometimes cause them to interfere with one another. These kinds of situations can be reproduced and protected against in laboratory conditions only to a certain point.

Precautions have been taken for transfer via X2X Link in case such interference should occur. For example, if an invalid checksum occurs, the I/O system will ignore the data from this bus cycle and the receiver receives the last valid data once more. With conventional (cyclic) data points, this error can often be ignored. In the following cycle, the same data point is again retrieved, adjusted and transferred.

Using Forward functionality with Flatstream communication makes this situation more complex. The receiver receives the old data again in this situation as well, i.e. the previous values for SequenceAck/SequenceCounter and the old MTU.

#### Loss of acknowledgment (SequenceAck)

If a SequenceAck value is lost, then the MTU was already transferred properly. For this reason, the receiver is permitted to continue processing with the next sequence. The SequenceAck is aligned with the associated SequenceCounter and sent back to the transmitter. Checking the incoming acknowledgments shows that all sequences up to the last one acknowledged have been transferred successfully (see sequences 1 and 2 in the image).

**Loss of transmission (SequenceCounter, MTU):**

If a bus cycle drops out and causes the value of SequenceCounter and/or the filled MTU to be lost, then no data reaches the receiver. At this point, the transmission routine is not yet affected by the error. The time-controlled MTU is released again and can be rewritten to.

The receiver receives SequenceCounter values that have been incremented several times. For the receive array to be put together correctly, the receiver is only permitted to process transmissions whose SequenceCounter has been increased by one. The incoming sequences must be ignored, i.e. the receiver stops and no longer transmits back any acknowledgments.

If the maximum number of unacknowledged sequences has been sent and no acknowledgments are returned, the transmitter must repeat the affected SequenceCounter and associated MTUs (see sequence 3 and 4 in the image).

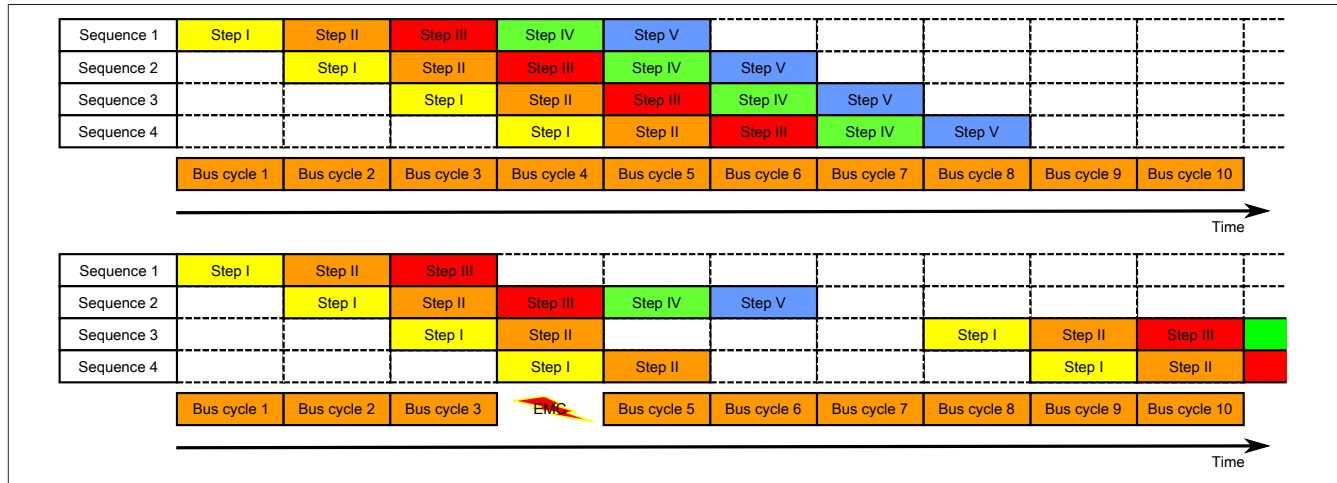


Figure 18: Effect of a lost bus cycle

**Loss of acknowledgment**

In sequence 1, the acknowledgment is lost due to disturbance. Sequences 1 and 2 are therefore acknowledged in Step V of sequence 2.

**Loss of transmission**

In sequence 3, the entire transmission is lost due to disturbance. The receiver stops and no longer sends back any acknowledgments.

The transmitting station continues transmitting until it has issued the maximum permissible number of unacknowledged transmissions.

5 bus cycles later at the earliest (depending on the configuration), it begins resending the unsuccessfully sent transmissions.

## 4 Commissioning

---

### 4.1 SG3 support

This module does not support SG3 target systems.

### 4.2 Using the module on the bus controller

Function model 254 "Bus controller" is used by default only by non-configurable bus controllers. All other bus controllers can use other registers and functions depending on the fieldbus used.

For detailed information, see section "Additional information - Using I/O modules on the bus controller" in the X20 user's manual (version 3.50 or later).

#### 4.2.1 CAN I/O bus controller

The module occupies 3 analog logical slots with CAN I/O.

### 4.3 Configuring the IO-Link device

The following options are available for configuring an IO-Link device:

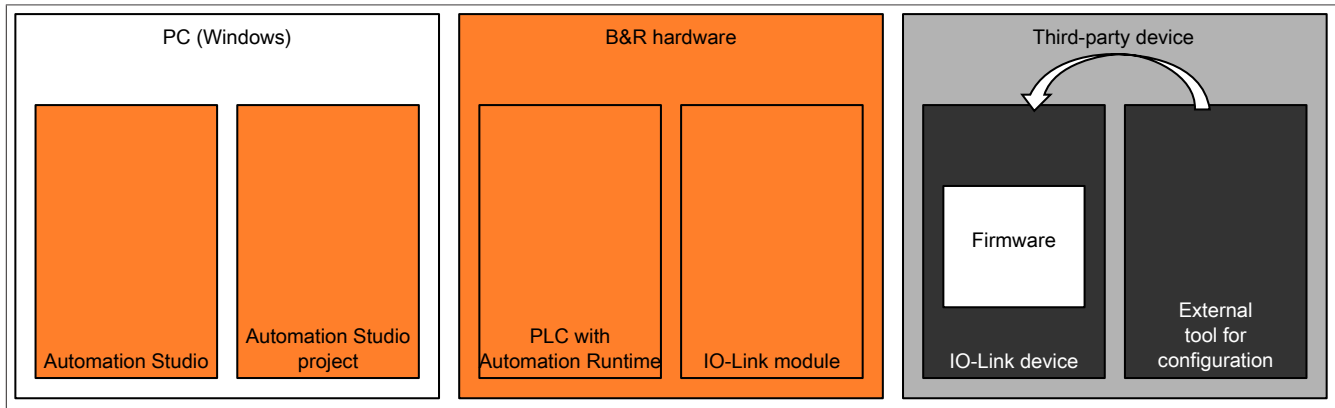
- Direct configuration
- Configuration via IODD/DTM support.  
A corresponding IODD or DTM file must be provided by the vendor for this.
- Restoring a configuration using the parameter server.  
For this, the IO-Link device must support the "parameter server" function per version 1.1 of the IO-Link specification.

**Information:**

Library "AsIoLink" offers another option for configuring the IO-Link device. This library is not part of this description.

### 4.3.1 Direct configuration

Direct configuration takes place independently of the B&R hardware and software used. The parameters can be entered via an additional configuration device, integrated display or other operating elements on the IO-Link device, for example.



#### Advantage

Advantageous for individual devices since the IO-Link device can be commissioned using the manufacturer's tools.

If problems occur during configuration of the IO-Link device, it is not necessary to check which software component is causing the malfunction.

#### Disadvantage

Each IO-Link device must be individually preconfigured manually.

Several development environments may have to be used on the user's computer.

### 4.3.2 IODD/DTM support

IO-Link devices can be configured using Automation Studio and the integrated FDT container. IODD/DTM support for IO-Link devices can be provided both online and offline.



#### Information:

To use Automation Studio to configure IO-Link devices, a corresponding hardware description file (IODD or DTM) must be downloaded and installed.

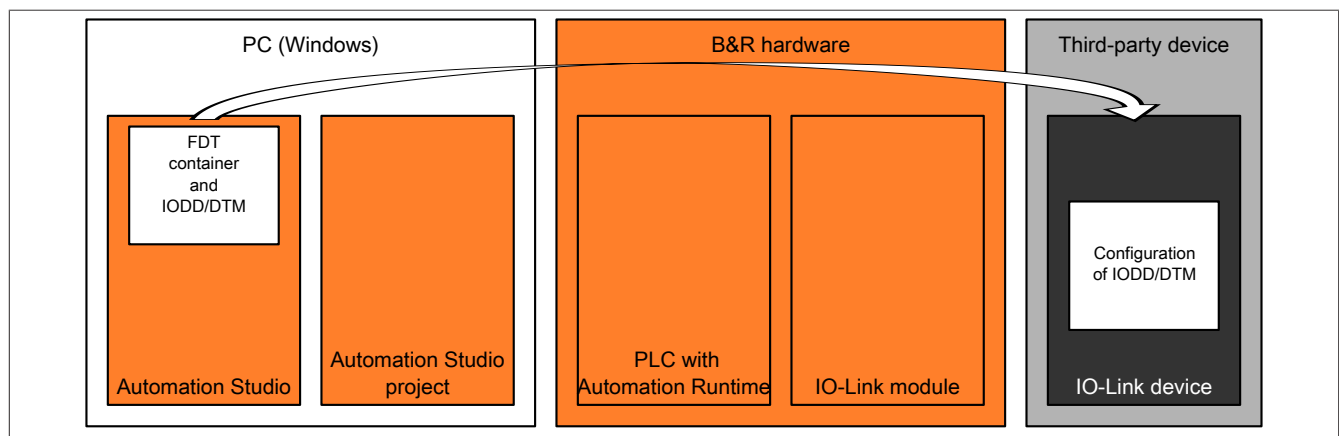


#### Information:

This function is available in Automation Runtime V4.08 and later.

#### 4.3.2.1 IODD/DTM (online)

During online configuration, the Automation Studio FDT container communicates directly with the IO-Link device. After the connection has been established, the configuration parameters can be adjusted as desired.



#### Advantage

No additional devices are generally required to configure the IO-Link device. All settings can be made by the user in a single development environment.

#### Disadvantage

Each IO-Link device must be configured individually.

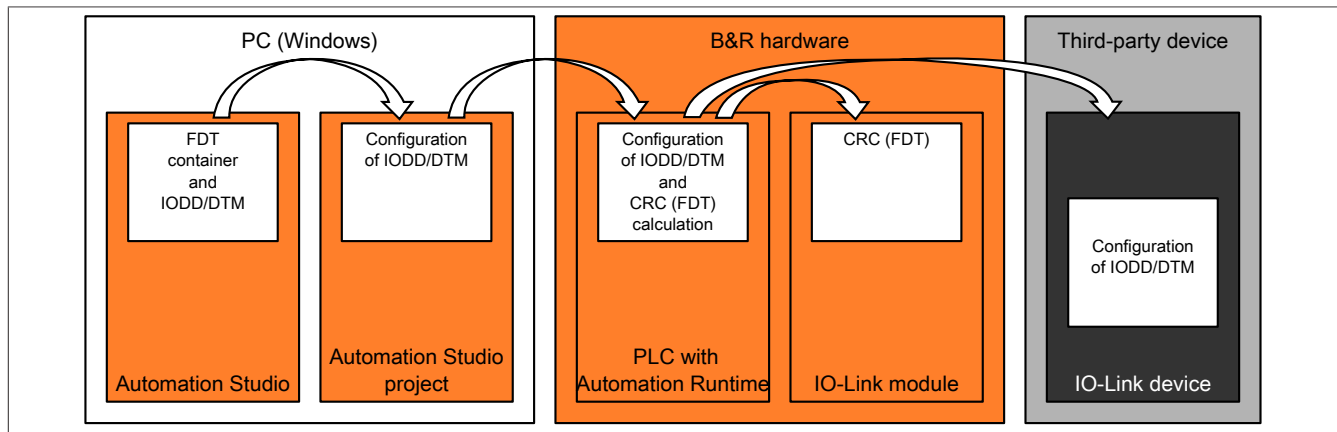


#### 4.3.2.2 IODD/DTM (offline)

With offline configuration, the parameter set that can be entered via the IODD or DTM file is stored in the Automation Studio project. During the download, the parameter set for the IO-Link device is transferred to the controller and from there imported into the IO-Link device via the module.

##### Procedure

- 1) When the IO-Link module is started, the checksum ( $CRC_{FDT}$ ) is calculated for the current parameter set.
- 2) If the previously stored checksum differs from the currently calculated checksum, the parameter set is transferred to the IO-Link device.
- 3) After the parameter set is transferred, the associated checksum ( $CRC_{FDT}$ ) is saved on the IO-Link module and can be used for future comparisons.
- 4) If the parameter set changes, a new checksum ( $CRC_{FDT}$ ) is generated the next time the controller is restarted and steps 2 and 3 are repeated.



##### Advantage

The configuration parameters of the IO-Link device are stored as part of the Automation Studio project. The user can work with one development environment and define all settings.

With series-produced machines, the IO-Link devices used later do not have to be preconfigured individually.

##### Disadvantage

The configuration options for the IO-Link device depend on the scope of the IODD or DTM file.



##### Information:

**Before the parameter set is transferred to the IO-Link device, the controller checks whether the connected device has the correct device ID. If the device ID is not correct, the procedure is aborted. The parameter set is not transferred, and the checksum is not saved.**

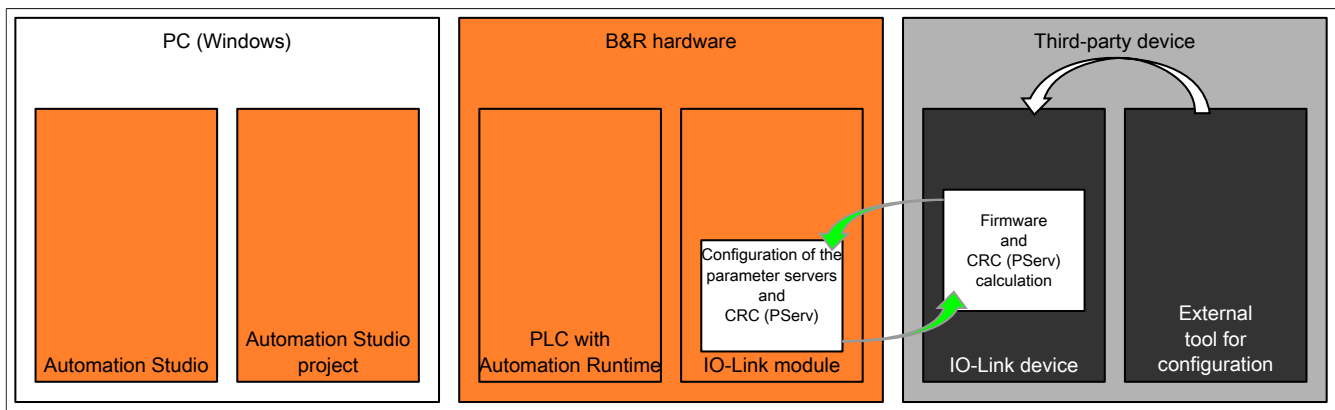
### 4.3.3 Parameter server

The "parameter server" function is defined in the IO-Link specification version 1.1 and later. This function makes it possible to replace an IO-Link device without requiring special knowledge from maintenance personnel.

The configuration saved on the IO-Link device is stored on the IO-Link module for this purpose. In addition, a checksum ( $CRC_{PServ}$ ) is calculated to enable simple comparison of the parameter sets.

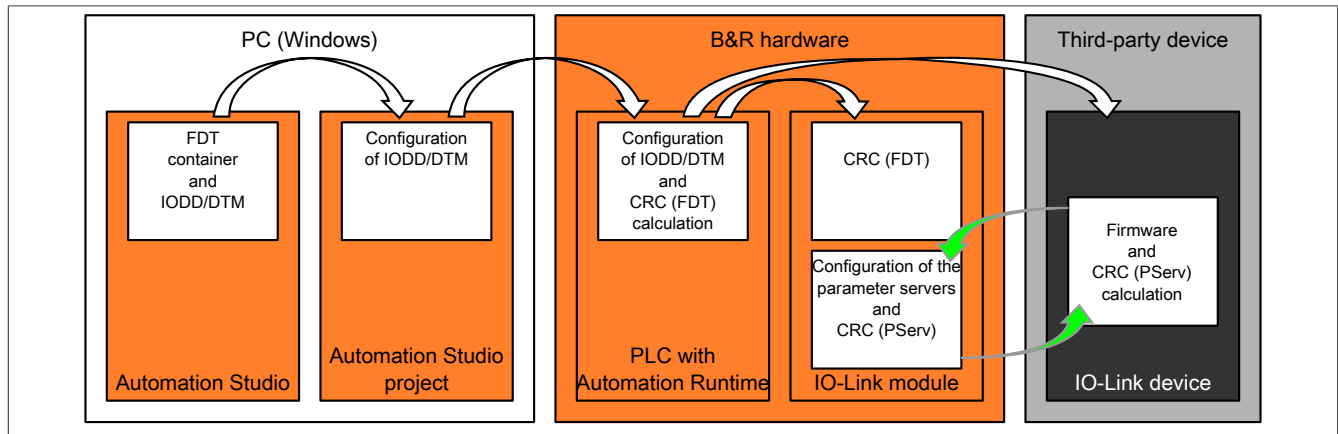
#### Procedure

- 1) If the IO-Link device supports the "parameter server" function, it calculates the checksum ( $CRC_{PServ}$ ) for its current parameter set during startup.
- 2) If the currently calculated checksum ( $CRC_{PServ}$ ) differs from the one previously stored on the IO-Link module, the parameter set of the IO-Link device differs from the one currently stored on the module.
- 3) The values of the device ID and serial number of the IO-Link device are evaluated to decide whether the parameter set must be downloaded from the device or from the IO-Link module.
  - a) If the device ID has changed, a different device type has been recognized. In this case, the parameter set of the IO-Link device must be read out and saved on the IO-Link module. The current checksum ( $CRC_{PServ}$ ) is also stored on the IO-Link module.
  - b) If the device ID is unchanged but the serial number has changed, it is assumed that the IO-Link device has been replaced with a device of the same type. In this case, the parameter set stored in the IO-Link module is downloaded to the IO-Link device.
  - c) If the device ID and serial number are unchanged, it is assumed that the IO-Link device has received a new configuration. In this case, the new parameter set of the IO-Link device is read out and saved on the IO-Link module. The current checksum ( $CRC_{PServ}$ ) is also stored on the IO-Link module.



#### 4.3.4 Using IODD/DTM and the parameter server together

IODD/DTM support and the parameter server can be used together. The two functions work independently but influence each other.



##### 4.3.4.1 Changing the configuration using IODD/DTM support

If the IO-Link device is reconfigured using an FDT container (IODD/DTM), the IO-Link device then calculates the new checksum ( $CRC_{PServ}$ ). The changed data is then read back from the parameter server of the IO-Link module.

##### 4.3.4.2 Replacing the IO-Link device

If the IO-Link device is replaced, the system only checks the checksum ( $CRC_{PServ}$ ). The parameter set of the FDT container remains disregarded since the checksum ( $CRC_{FDT}$ ) in the project on the controller still matches the checksum ( $CRC_{FDT}$ ) stored on the IO-Link module (see ["Parameter server" on page 58](#) for the procedure).

## 5 Register description

### 5.1 General data points

In addition to the registers described in the register description, the module has additional general data points. These are not module-specific but contain general information such as serial number and hardware variant.

General data points are described in section "Additional information - General data points" in the X20 System user's manual.

### 5.2 Function model 0 - Standard

Register	Name	Data type	Read		Write	
			Cyclic	Acyclic	Cyclic	Acyclic
Module configuration						
3073 + N * 1024	CfO_OperatingMode0N (index N = 1 to 4)	USINT				•
IO-Link configuration						
3076 + N * 1024	CfO_ChannelMode0N (index N = 1 to 4)	UDINT				•
3099 + N * 1024	CfO_IdentificationRevisionId0N (index N = 1 to 4)	USINT				•
3102 + N * 1024	CfO_IdentificationVendorId0N (index N = 1 to 4)	UINT				•
3108 + N * 1024	CfO_IdentificationDeviceId0N (index N = 1 to 4)	UDINT				•
3116 + N * 1024	CfO_PDI_TypeInfo0N (index N = 1 to 4)	UDINT				•
3124 + N * 1024	CfO_PDO_TypeInfo0N (index N = 1 to 4)	UDINT				•
15372	CfO_TimerCycle	UDINT				•
15366	CfO_TimerOffset	INT				•
3086 + N * 1024	CfO_ReqCycleMultiple0N (index N = 1 to 4)	UINT				•
3090 + N * 1024	CfO_ReqCycleDivisor0N (index N = 1 to 4)	UINT				•
3094 + N * 1024	CfO_ReqCycleOffset0N (index N = 1 to 4)	UINT				•
3082 + N * 1024	CfO_ReqCycleTime0N (index N = 1 to 4)	UINT				•
IO-Link communication						
4473 + N * 8	OutputData01_N (index N = 1 to 8)	(U)SINT			•	
4474 + N * 8	OutputData01_N (index N = 1 to 8)	(U)SINT				
4476 + N * 8	OutputData01_N (index N = 1 to 8)	(U)DINT REAL				
5497 + N * 8	OutputData02_N (index N = 1 to 8)	(U)SINT			•	
5498 + N * 8	OutputData02_N (index N = 1 to 8)	(U)SINT				
5500 + N * 8	OutputData02_N (index N = 1 to 8)	(U)DINT REAL				
6521 + N * 8	OutputData03_N (index N = 1 to 8)	(U)SINT			•	
6522 + N * 8	OutputData03_N (index N = 1 to 8)	(U)SINT				
6524 + N * 8	OutputData03_N (index N = 1 to 8)	(U)DINT REAL				
7545 + N * 8	OutputData04_N (index N = 1 to 8)	(U)SINT			•	
7546 + N * 8	OutputData04_N (index N = 1 to 8)	(U)SINT				
7548 + N * 8	OutputData04_N (index N = 1 to 8)	(U)DINT REAL				
7	SIO or digital outputs	USINT			•	
	DigitalOutput01	Bit 0				
	...	...				
	DigitalOutput04	Bit 3				
	DisablePowerSupply01	Bit 4				
	...	...				
	DisablePowerSupply04	Bit 7				
4345 + N * 8	InputData01_N (index N = 1 to 8)	(U)SINT	•			
4346 + N * 8	InputData01_N (index N = 1 to 8)	(U)INT				
4348 + N * 8	InputData01_N (index N = 1 to 8)	(U)DINT REAL				
5369 + N * 8	InputData02_N (index N = 1 to 8)	(U)SINT	•			
5370 + N * 8	InputData02_N (index N = 1 to 8)	(U)INT				
5372 + N * 8	InputData02_N (index N = 1 to 8)	(U)DINT REAL				
6393 + N * 8	InputData03_N (index N = 1 to 8)	(U)SINT	•			
6394 + N * 8	InputData03_N (index N = 1 to 8)	(U)INT				
6396 + N * 8	InputData03_N (index N = 1 to 8)	(U)DINT REAL				
7417 + N * 8	InputData04_N (index N = 1 to 8)	(U)SINT	•			
7418 + N * 8	InputData04_N (index N = 1 to 8)	(U)INT				
7420 + N * 8	InputData04_N (index N = 1 to 8)	(U)DINT REAL				
1	SIO or digital inputs	USINT	•			
	DigitalInput01	Bit 0				
	...	...				
	DigitalInput04	Bit 3				

Register	Name	Data type	Read		Write	
			Cyclic	Acyclic	Cyclic	Acyclic
IO-Link status response						
3	Sync (status byte)	USINT	•			
	Synchronized01	Bit 0				
	...	...				
	Synchronized04	Bit 3				
	CycleEnd01	Bit 4				
	...	...				
	CycleEnd04	Bit 7				
5	Overload (status byte)	USINT	•			
	Overload01	Bit 0				
	...	...				
	Overload04	Bit 3				
17 + N * 16	ChannelStatus0N (index N = 1 to 4)	USINT	•			
22 + N * 16	FrameCount0N (index N = 1 to 4)	SINT	•			
3586 + N * 1024	CycleStartNettime0N (index N = 1 to 4)	INT	•			
3588 + N * 1024	CycleStartNettime0N (index N = 1 to 4)	DINT				
3594 + N * 1024	CycleEndNettime0N (index N = 1 to 4)	INT	•			
3596 + N * 1024	CycleEndNettime0N (index N = 1 to 4)	DINT				
IO-Link event interface						
113	EventPortSeq	USINT	•	•		
115	EventQualifier	USINT	•	•		
118	EventCode	UINT	•	•		
121	EventsLeft	USINT		•		
123	EventQuit	USINT			•	•
123	EventQuitReadBack	USINT		•		
IO-Link parameter server						
19 + N * 16	DsControl0N (index N = 1 to 4)	USINT			•	•
3140 + N * 1024	Cfo_DS_Config0N (index N = 1 to 4)	UDINT				•
3241 + N * 1024	DsProgress0N (index N = 1 to 4)	USINT		•		
3148 + N * 1024	Cfo_DS_SaveCtrl0N (index N = 1 to 4)	UDINT				•
3156 + N * 1024	Cfo_DS_SaveData0N (index N = 1 to 4)	UDINT				•
IO-Link timestamp						
3610 + N * 1024	IoLinkTimestampIn0N (index N = 1 to 4)	INT	•			
3612 + N * 1024	IoLinkTimestampIn0N (index N = 1 to 4)	DINT				
3617 + N * 1024	IoLinkTimestampInStatusSeq0N (index N = 1 to 4)	USINT	•			
3714 + N * 1024	IoLinkTimestampOut0N (index N = 1 to 4)	INT			•	
3716 + N * 1024	IoLinkTimestampOut0N (index N = 1 to 4)	DINT				
3721 + N * 1024	IoLinkTimestampOutCtrlSeq0N (index N = 1 to 4)	USINT			•	
3619 + N * 1024	IoLinkTimestampOutStatus0N (index N = 1 to 4)	USINT	•			
IO-Link device IDs						
3202 + N * 1024	VendorId0N (index N = 1 to 4)	UINT	•	•		
3212 + N * 1024	DeviceId0N (index N = 1 to 4)	UDINT	•	•		
3206 + N * 1024	FunctionId0N (index N = 1 to 4)	UINT	•	•		
3218 + N * 1024	CycleTime0N (index N = 1 to 4)	UINT	•	•		
3222 + N * 1024	CycleMultiple0N (index N = 1 to 4)	UINT		•		
3226 + N * 1024	CycleDivisor0N (index N = 1 to 4)	UINT		•		
3230 + N * 1024	MinCycleTime0N (index N = 1 to 4)	UINT		•		
3233 + N * 1024	PDI_Size0N (index N = 1 to 4)	USINT		•		
3235 + N * 1024	PDO_Size0N (index N = 1 to 4)	USINT		•		
3237 + N * 1024	Baudrate0N (index N = 1 to 4)	USINT		•		
3239 + N * 1024	IoLinkVersionID0N (index N = 1 to 4)	USINT		•		
Statistics counter registers						
Command interface						
98	ParameterIndexOut	UINT			•	•
101	ParameterSubIndexOut	USINT			•	•
103	ParameterCtrlOut	USINT			•	•
108	ParameterDataOut_0	UDINT			•	•
103	ParameterCtrlIn	USINT	•	•		
108	ParameterDataIn_0	UDINT	•	•		
Flatstream						
193	Cfo_OutputMTU	USINT				•
195	Cfo_InputMTU	USINT				•
197	Cfo_FlatStreamMode	USINT				•
199	Cfo_Forward	USINT				•
204	Cfo_ForwardDelay	UDINT				•
129	InputSequence	USINT	•			
129 + N * 2	RxByteN (index N = 1 to 27)	USINT	•			
129	OutputSequence	USINT			•	
129 + N * 2	TxByteN (index N = 1 to 27)	USINT			•	

## 5.3 Function model 254 - Bus controller

Register	Offset <sup>1)</sup>	Name	Data type	Read		Write	
				Cyclic	Acyclic	Cyclic	Acyclic
Module properties							
3073 + N * 1024	-	CfO_OperatingMode0N (Index N = 1 to 4)	USINT				•
IO-Link configuration							
3076 + N * 1024	-	CfO_ChannelMode0N (Index N = 1 to 4)	UDINT				•
3099 + N * 1024	-	CfO_IdentificationRevisionId0N (index N = 1 to 4)	USINT				•
3102 + N * 1024	-	CfO_IdentificationVendorId0N (Index N = 1 to 4)	UINT				•
3108 + N * 1024	-	CfO_IdentificationDeviceId0N (Index N = 1 to 4)	UDINT				•
3116 + N * 1024	-	CfO_PDI_TypeInfo0N (Index N = 1 to 4)	UDINT				•
3124 + N * 1024	-	CfO_PDO_TypeInfo0N (Index N = 1 to 4)	UDINT				•
15372	-	CfO_TimerCycle	UDINT				•
15366	-	CfO_TimerOffset	INT				•
3086 + N * 1024	-	CfO_ReqCycleMultiple0N (Index N = 1 to 4)	UINT				•
3090 + N * 1024	-	CfO_ReqCycleDivisor0N (Index N = 1 to 4)	UINT				•
3094 + N * 1024	-	CfO_ReqCycleOffset0N (Index N = 1 to 4)	UINT				•
3082 + N * 1024	-	CfO_ReqCycleTime0N (Index N = 1 to 4)	UINT				•
IO-Link communication							
4473 + N * 8	(N-1)	OutputData01_N (index N = 1 to 4)	USINT			•	
5497 + N * 8	4 + (N-1)	OutputData02_N (index N = 1 to 4)	USINT			•	
6521 + N * 8	8 + (N-1)	OutputData03_N (index N = 1 to 4)	USINT			•	
7545 + N * 8	12 + (N-1)	OutputData04_N (index N = 1 to 4)	USINT			•	
7	-	SIO or digital outputs	USINT				•
		DigitalOutput01	Bit 0				
		...	...				
		DigitalOutput04	Bit 3				
		DisablePowerSupply01	Bit 4				
		...	...				
4345 + N * 8	(N-1)	InputData01_N (index N = 1 to 4)	USINT	•			
5369 + N * 8	4 + (N-1)	InputData02_N (index N = 1 to 4)	USINT	•			
6393 + N * 8	8 + (N-1)	InputData03_N (index N = 1 to 4)	USINT	•			
7417 + N * 8	12 + (N-1)	InputData04_N (index N = 1 to 4)	USINT	•			
1	20	SIO or digital inputs	USINT	•			
		DigitalInput01	Bit 0				
		...	...				
		DigitalInput04	Bit 3				
IO-Link status response							
3	21	Sync (status byte)	USINT	•			
		Synchronized01	Bit 0				
		...	...				
		Synchronized04	Bit 3				
		CycleEnd01	Bit 4				
		...	...				
5	22	Overload (status byte)	USINT	•			
Overload01		Bit 0					
...		...					
Overload04		Bit 3					
17 + N * 16	15 + N	ChannelStatus0N (Index N = 1 to 4)	USINT	•			
22 + N * 16	-	FrameCount0N (Index N = 1 to 4)	SINT		•		
3586 + N * 1024	-	CycleStartNettime0N (Index N = 1 to 4)	INT		•		
3588 + N * 1024	-	CycleStartNettime0N (Index N = 1 to 4)	DINT				
3594 + N * 1024	-	CycleEndNettime0N (Index N = 1 to 4)	INT		•		
3596 + N * 1024	-	CycleEndNettime0N (Index N = 1 to 4)	DINT				
IO-Link event interface							
113	-	EventPortSeq	USINT		•		
115	-	EventQualifier	USINT		•		
118	-	EventCode	UINT		•		
121	-	EventsLeft	USINT		•		
123	-	EventQuit	USINT				•
123	-	EventQuitReadBack	USINT		•		
IO-Link parameter server							
19 + N * 16	-	DsContol0N (Index N = 1 to 4)	USINT				•
3140 + N * 1024	-	CfO_DS_Config0N (Index N = 1 to 4)	UDINT				•
3241 + N * 1024	-	DsProgress0N (Index N = 1 to 4)	USINT		•		
3148 + N * 1024	-	CfO_DS_SaveCtrl0N (Index N = 1 to 4)	UDINT				•
3156 + N * 1024	-	CfO_DS_SaveData0N (Index N = 1 to 4)	UDINT				•

Register	Offset <sup>1)</sup>	Name	Data type	Read		Write	
				Cyclic	Acyclic	Cyclic	Acyclic
IO-Link timestamp							
3610 + N * 1024	-	IoLinkTimestampIn0N (Index N = 1 to 4)	INT		•		
3612 + N * 1024	-	IoLinkTimestampIn0N (Index N = 1 to 4)	DINT				
3617 + N * 1024	-	IoLinkTimestampInStatusSeq0N (Index N = 1 to 4)	USINT		•		
3714 + N * 1024	-	IoLinkTimestampOut0N (Index N = 1 to 4)	INT				•
3716 + N * 1024	-	IoLinkTimestampOut0N (Index N = 1 to 4)	DINT				
3721 + N * 1024	-	IoLinkTimestampOutCtrlSeq0N (Index N = 1 to 4)	USINT				•
3619 + N * 1024	-	IoLinkTimestampOutStatus0N (Index N = 1 to 4)	USINT		•		
IO-Link device IDs							
3202 + N * 1024	-	VendorId0N (Index N = 1 to 4)	UINT		•		
3212 + N * 1024	-	DeviceId0N (Index N = 1 to 4)	UDINT		•		
3206 + N * 1024	-	FunctionId0N (Index N = 1 to 4)	UINT		•		
3218 + N * 1024	-	CycleTime0N (Index N = 1 to 4)	UINT		•		
3222 + N * 1024	-	CycleMultible0N (Index N = 1 to 4)	UINT		•		
3226 + N * 1024	-	CycleDivisor0N (Index N = 1 to 4)	UINT		•		
3230 + N * 1024	-	MinCycleTime0N (Index N = 1 to 4)	UINT		•		
3233 + N * 1024	-	PDI_Size0N (Index N = 1 to 4)	USINT		•		
3235 + N * 1024	-	PDO_Size0N (Index N = 1 to 4)	USINT		•		
3237 + N * 1024	-	Baudrate0N (Index N = 1 to 4)	USINT		•		
3239 + N * 1024	-	IoLinkVersionID0N (Index N = 1 to 4)	USINT		•		
Command interface							
98	-	ParameterIndexOut	UINT				•
101	-	ParameterSubIndexOut	USINT				•
103	-	ParameterCtrlOut	USINT				•
108	-	ParameterDataOut_0	UDINT				•
103	-	ParameterCtrlIn	USINT		•		
108	-	ParameterDataIn_0	UDINT		•		

1) The offset specifies the position of the register within the CAN object.

## 5.4 Module configuration

### 5.4.1 OperatingMode

Name:

CfO\_OperatingMode0X

This register is the same as the first byte of register "[ChannelMode](#)" on page 64 in the IO-Link configuration. It contains all settings of an IO-Link channel that are permitted to be changed at runtime.

Data type	Values	Bus controller default setting
USINT	See the bit structure.	0

Bit structure:

Bit	Description	Values	Information
0 - 1	Channel mode	00	Mode: Inactive (bus controller default setting)
		01	Mode: SIO output The channel's C/Q connector is configured as a digital output.
		10	Mode: SIO input The channel's C/Q connector is configured as a digital input.
		11	Mode: Operate The channel's C/Q connector is configured for IO-Link data transfer.
2 - 7	Reserved	-	

## 5.5 IO-Link configuration

The module establishes communication with the IO-Link device if register "[ChannelMode](#)" on page 64 of the corresponding channel is configured. The other registers in this section can be used to adapt the data exchange to the application requirements.

## Register description

### 5.5.1 ChannelMode

Name:

CfO\_ChannelMode0X

The user has the option of setting all channel-specific settings via this register.

Data type	Values	Bus controller default setting
UDINT	See the bit structure.	0

Bit structure:

Bit	Description	Values	Information
0 - 1	Channel mode	00	Mode: Inactive (bus controller default setting)
		01	Mode: SIO output The channel's C/Q connector is configured as a digital output.
		10	Mode: SIO input The channel's C/Q connector is configured as a digital input.
		11	Mode: Operate The channel's C/Q connector is configured for IO-Link data transfer.
2 - 7	Reserved	-	
8 - 9	Threshold value for overcurrent on the channel <sup>1)</sup> (OverCurrentThreshold in Automation Studio configuration)	00	Up to revision D0: 250 mA (bus controller default setting) Revision E0 and later: 500 mA
		01	Up to revision D0: 125 mA Revision E0 and later: 500 mA
		10	Up to revision D0: 75 mA Revision E0 and later: 500 mA
		11	Up to revision D0: 50 mA Revision E0 and later: 500 mA
10 - 11	Reserved	-	
12 - 13	Switch-off duration after overload <sup>1)</sup> (OverloadOffTime in Automation Studio configuration)	00	Up to revision D0: 20 ms (bus controller default setting) Revision E0 and later: 50 ms (setting "Low off time" in Automation Studio)
		01	Up to revision D0: 12 ms Revision E0 and later: 50 ms (setting "Medium off time" in Automation Studio)
		10	Up to revision D0: 6.4 ms Revision E0 and later: 50 ms (setting "Extreme low off time" in Automation Studio)
		11	Up to revision D0: 32 ms Revision E0 and later: 50 ms (setting "High off time" in Automation Studio)
14 - 15	Reserved	-	
16 - 17	Mode for synchronization	00	Free-running (asynchronous) (bus controller default setting)
		01	Synchronous (manual)
		10	Synchronous (automatic)
		11	Impermissible
18 - 19	Reserved	-	
20 - 23	Inspection level	0	Tests disabled (bus controller default setting)
		1	Testing <a href="#">VendorID</a> and <a href="#">DeviceID</a>
24 - 25	IO-Link timestamp	00	No timestamp (bus controller default setting)
		01	Input timestamp
		10	Output timestamp
		11	Input and output timestamps
26	Format of the IO-Link output timestamp <sup>2)</sup>	0	32-bit (DINT) (bus controller default setting)
		1	16-bit (INT)
27 - 32	Reserved	-	

1) This is overload protection for the C/Q connector of the IO-Link channels (IO-Link data line or SIO output) as opposed to overload protection of the IO-Link power supply.

2) This bit informs the module of the format used for the [IoLinkTimestampOut](#) output timestamp. In Automation Studio, this setting is made implicitly in the I/O configuration together with the selection of the data type for the IO-Link timestamp.




### 5.5.2 IdentificationRevisionID

Name:

CfO\_IdentificationRevisionId0X

If the identifiers (IDs) of the connected device should be verified during startup, the IO-Link revision with which the check takes place can be disclosed in this register.

Data type	Values	Information
USINT	0	The revision read from the device is used.
	16	The connected device is checked per revision V1.0.
	17	The connected device is checked per revision V1.1.
 <p>If the device does not support this standard, error code 41 is output in register "<a href="#">ChannelStatus</a>" on <a href="#">page 72</a>.</p>		

### 5.5.3 IdentificationVendorID

Name:

CfO\_IdentificationVendorId0X

If the vendor ID should be verified during startup, the expected vendor ID must be entered into this register. The check can be enabled by setting the inspection level in the register "[ChannelMode](#)" on [page 64](#).



#### Information:

If the expected ID does not match the actual ID of the connected IO-Link device, communication for this channel is not started.

Data type	Values	Information
UINT	0 to 65535	Bus controller default setting: 0

### 5.5.4 IdentificationDeviceID

Name:

CfO\_IdentificationDeviceId0X

If the device ID should be verified during startup, the expected ID of the IO-Link device must be entered in this register. The check can be enabled by setting the inspection level in the register "[ChannelMode](#)" on [page 64](#).



#### Information:

If the expected ID does not match the actual ID of the connected IO-Link device, communication for this channel is not started.

Data type	Values	Information
UDINT	0 to 4,294,967,295	Bus controller default setting: 0

## 5.5.5 PDI\_TypeInfo

Name: CfO\_PDI\_TypeInfo0X

To transfer process data from the IO-Link device to the controller (application), the information is first read from the module and saved temporarily. Typically, 4 bytes are reserved for each piece for registered information (see "[IO-Link communication](#)" on page 10).

This register is used to configure how the incoming IO-Link process data stream (IO-Link frame) is divided. According to this configuration, the IO-Link process data is made available to the application via the corresponding [InputData](#) registers. The InputData registers are assigned to individual data points with the corresponding data type in the I/O mapping.

Data type	Values	Bus controller default setting
UDINT	See the bit structure.	0

Bit structure:

Bit	Description	Value	Information
0 to 3	IO-Link information 1	0000	Array[4] of Bytes (bus controller default setting)
		0001	USINT
		0010	SINT
		0011	UINT
		0100	INT
		0101	UDINT
		0110	DINT
		0111	REAL
		1000 - 1111	Reserved
4 - 7	IO-Link information 2		Possible values are identical with IO-Link information 1.
8 - 11	IO-Link information 3		
12 - 15	IO-Link information 4		
16 - 19	IO-Link information 5		
20 - 23	IO-Link information 6		
24 - 27	IO-Link information 7		
28 - 31	IO-Link information 8		



### Information:

**With setting 0 (array[4] of bytes), the bytes from the IO-Link data stream are unchanged when copied. The byte order is changed in all other modes (from big-endian to little-endian).**

## 5.5.6 PDO\_TypeInfo

Name: CfO\_PDO\_TypeInfo0X

In order to transfer process data to the IO-Link device, this register is used to configure which data type of the individual "OutputData" on page 69 registers are used to merge the outgoing IO-Link process data stream (IO-Link frame, see "IO-Link communication" on page 10). According to this configuration, OutputData registers are assigned to data points with the corresponding data type in Automation Studio (I/O mapping).

Data type	Values	Bus controller default setting
UDINT	See the bit structure.	0

Bit structure:

Bit	Description	Value	Information
0 to 3	IO-Link information 1	0000	Array[4] of Bytes (bus controller default setting)
		0001	USINT
		0010	SINT
		0011	UINT
		0100	INT
		0101	UDINT
		0110	DINT
		0111	REAL
		1000 - 1111	Reserved
4 - 7	IO-Link information 2		Possible values are identical with IO-Link information 1.
8 - 11	IO-Link information 3		
12 - 15	IO-Link information 4		
16 - 19	IO-Link information 5		
20 - 23	IO-Link information 6		
24 - 27	IO-Link information 7		
28 - 31	IO-Link information 8		



### Information:

With setting 0 (array[4] of bytes), the bytes from the IO-Link data stream are unchanged when copied. The byte order is changed in all other modes (from big-endian to little-endian).

## 5.6 Configuring IO-Link timing characteristics

The module must manage records from 2 different communication standards at runtime. For efficient communication on the X2X Link network, it must be ensured that the cycle time of all X2X modules corresponds to the bus cycle time.

### 5.6.1 TimerCycle

Name:

CfO\_TimerCycle

This register can be used to configure synchronous IO-Link communication. If the module timer should not be operated with the same cycle, the period duration of the module timer can be set in this register in microseconds. This allows channels to be synchronized with each other even if a very unusual X2X cycle time is used.

Data type	Values	Information
UDINT	0 to 4,294,967,295	Bus controller default setting: Current X2X cycle time

### 5.6.2 TimerOffset

Name:

CfO\_TimerOffset

This register can be used to configure synchronous IO-Link communication. If the module timer should run with a time delay to the X2X Link network, this register can be used to specify by how many microseconds before or after the module timer should be offset.

Data type	Values	Information
INT	-32768 to 32767	Bus controller default setting: 0

### 5.6.3 ReqCycleMultiple

Name:

CfO\_ReqCycleMultiple0X

The synchronization cycle time of a channel can be manually set with this register. This cycle time can be used together with register "[ReqCycleDivisor](#)" on page 68 to define the IO-Link cycle time. See "[Synchronous operation](#)" on page 13 for an example.



#### Information:

**If this register is not defined for an IO-Link channel or specified with zero, the values of registers CycleMultiple and CycleDivisor are calculated automatically during module startup.**

Data type	Values	Information
UINT	0 to 65535	Bus controller default setting: 0

### 5.6.4 ReqCycleDivisor

Name:

CfO\_ReqCycleDivisor0X

This register can be used together with "[ReqCycleMultiple](#)" on page 68 to define the IO-Link cycle time. See "[Synchronous operation](#)" on page 13 for an example.



#### Information:

**If this register is not defined for an IO-Link channel or specified with zero, the values of registers CycleMultiple and CycleDivisor are calculated automatically during module startup.**

Data type	Values	Information
UINT	0 to 65535	Bus controller default setting: 0

## 5.6.5 ReqCycleOffset

Name:

CfO\_ReqCycleOffset0X

This register can be used to offset the IO-Link cycle of a channel to the synchronization cycle.

This offset can be useful if all channels operate with the same cycle time. All channels will be ready at the same time in this case, which may result in the module not processing all data in time. Offsets can be used to prevent such bottlenecks and distribute the data volume more evenly.

Data type	Values	Information
UINT	0 to 65535	Configured in timer cycles. Bus controller default setting: 0

## 5.6.6 ReqCycleTime

Name:

CfO\_ReqCycleTime0X

This register is used for free-running (asynchronous) IO-Link communication. It contains the cycle time directly specified for the IO-Link query in microseconds.



### Information:

- In free-running mode, no **NetTime** data points are permitted to be used except for "**CycleEndNettime**" on page 73.
- If the specified cycle time of the IO-Link communication undershoots the minimum cycle time of the device, the IO-Link data is queried with the minimum cycle time of the device.
- For efficient IO-Link communication, the set query cycle should correspond to the specified IO-Link cycle times. If the value is unsuitable, the next suitable cycle time is used automatically.

Data type	Values	Information
UINT	0 to 65535	In 100 µs increments. Bus controller default setting: 0

## 5.7 IO-Link communication

### 5.7.1 OutputData

Name:

OutputDataXX\_1 to OutputDataXX\_8

Output data from the IO-Link device in IO-Link communication mode. Alternatively, a byte array can be used. The user must then manually allocate the bytes to the required data types.

Register "**PDO\_TypeInfo**" on page 67 can be used to configure how many bytes from the output registers should be applied to the IO-Link frame.

Data type	Values
USINT	0 to 255
SINT	-128 to 127
UINT	0 to 65535
INT	-32768 to 32767
UDINT	0 to 4,294,967,295
DINT	-2147483648 to 2147483647
REAL	-3.4E38 to 3.4E38

## Register description

### 5.7.2 Digital SIO outputs

Name:

DigitalOutput0X

DisablePowerSupply0X

If a channel is operated in SIO mode (SIO output), the SIO output of the IO-Link channel can be controlled via this register. In addition, the power supply of each IO-Link channel can be switched on or off individually.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0	DigitalOutput01	0	Reset digital SIO output 01
		1	Set digital SIO output 01
...		...	
3	DigitalOutput04	0	Reset digital SIO output 04
		1	Set digital SIO output 04
4	DisablePowerSupply01	0	Switch power supply for IO-Link channel 01 on
		1	Switch power supply for IO-Link channel 01 off
...		...	
7	DisablePowerSupply04	0	Switch power supply for IO-Link channel 04 on
		1	Switch power supply for IO-Link channel 04 off

### 5.7.3 InputData

Name:

InputDataXX\_1 to InputDataXX\_8

Input data from the IO-Link device in IO-Link communication mode. Alternatively, a byte array can be used. The user must then manually allocate the bytes to the required data types.

Data type	Values
USINT	0 to 255
SINT	-128 to 127
UINT	0 to 65535
INT	-32768 to 32767
UDINT	0 to 4,294,967,295
DINT	-2147483648 to 2147483647
REAL	-3.4E38 to 3.4E38

### 5.7.4 Digital SIO inputs

Name:

DigitalInput0X

If a channel is operated in SIO mode (SIO input), the input state of the channel can be read in via this register.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0	DigitalInput01	0	Digital SIO input 01 reset
		1	Digital SIO input 01 set
...		...	
3	DigitalInput04	0	Digital SIO input 04 reset
		1	Digital SIO input 04 set
4 - 7	Reserved	-	

## 5.8 IO-Link status response

The status registers for IO-Link communication are explained in the following chapter. The status information provides information about the current situation between the module and IO-Link device. It can be retrieved from the controller and evaluated in the application task.

### 5.8.1 Sync (status byte)

Name:

Synchronized0X

CycleEnd0X

The module uses this status register to report whether error-free communication with the device was possible during the last module cycle.

- The CycleEnd bits indicate whether the last data sent to the IO-Link device has been processed. The CycleEnd bits are reset after each X2X cycle.
- The Synchronized bits indicate that the channel is synchronized without errors.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0	Synchronized01	0	Synchronization for channel 1 not OK
		1	Synchronization for channel 1 OK
...	...	...	...
3	Synchronized04	0	Synchronization for channel 4 not OK
		1	Synchronization for channel 4 OK
4	CycleEnd01	0	I/O cycle end: No new IO-Link data
		1	I/O cycle end: New data transmitted and received
...	...	...	...
7	CycleEnd04	0	I/O cycle end: No new IO-Link data
		1	I/O cycle end: New data transmitted and received

### 5.8.2 Overload (status byte)

Name:

Overload0X

The module uses this status register to report whether an overload in the form of overcurrent or overtemperature has occurred on the channel power supply or data line.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0	Overload01	0	Channel 1: No overload
		1	Channel 1: Overload
...	...	...	...
3	Overload04	0	Channel 4: No overload
		1	Channel 4: Overload
4 - 7	Reserved	-	

## Register description

### 5.8.3 ChannelStatus

Name:

ChannelStatus0X

This register is used to display the current status of the IO-Link channel.

Data type	Values	Information	State
USINT	0	Channel inactive	Disabled
	1	Used as a digital SIO output	SIO mode
	2	Used as a digital SIO input	
	3	IO-Link device startup, mode PREOPERATIONAL	Communication is running, but no process data is being exchanged. However, acyclic access is possible.
	4	Operation, OPERATE mode	Communication in progress
	5	Operation, parameter server data OK	
	6	Parameter server: Upload active	
	7	Parameter server: Download active	Communication is running and process data is being provided.
	8	Parameter server: Delete active	
	9	IODD parameters are written.	
	10 to 20	Reserved	Communication in progress. However, an error has occurred on the parameter server. Parameter server errors can be acknowledged via register "DsControl" on page 75.
	21	General error in the <a href="#">parameter server</a> , e.g. <ul style="list-style-type: none"> <li>The parameter server is not supported.</li> <li>Error accessing an object managed by the parameter server</li> <li>Internal error</li> </ul>	
	22	The parameter server is locked by the IO-Link device.	
	23	Parameter server empty: An attempt was made to load data to the IO-Link device although no data is stored in the EEPROM of the DS module.	
	24	New serial number detected: The user must use register " <a href="#">DsControl</a> " on page 75 to decide what should be done (upload - download - restore default values).	
	25	Parameter server incompatible (new device ID or new vendor ID detected): The data in EEPROM does not match the connected IO-Link device. The user must use register " <a href="#">DsControl</a> " on page 75 to decide whether an upload should be performed.	
	26	Upload request received: The user must use register " <a href="#">DsControl</a> " on page 75 to decide what should be done (upload - download - restore default values).	
	27	The parameter checksum of the IO-Link device has changed: The user must use register " <a href="#">DsControl</a> " on page 75 to decide what should be done (upload - download - restore default values).	
	28	Error transmitting the SAVE command	
	29	Reserved	
	30	Process data invalid	Communication in progress. However, the device marked the process data as invalid.
	31 to 39	Reserved	No communication
	40	No connection	
	41	The configured revision ID is not supported by the connected device.	Communication is running, but no process data is being exchanged. However, acyclic access is possible.
	42	The device ID or vendor ID of the connected IO-Link device does not match the specified IDs.	
	43	The configured serial number does not match the serial number of the connected device.	
	44	Timestamp error The IO-Link device does not support IO-Link timestamps.	
	45	Error during device startup.	
	46 to 255	Reserved	No communication

### 5.8.4 FrameCount

Name:

FrameCount0X

The received IO-Link frames are counted in this register. In contrast to the [sync bits](#), register FrameCount ensures that all frames are really recognized, even if X2X cycles are lost or if the IO-Link cycle is faster than the X2X cycle.

Data type	Values
SINT	-128 to 127



### 5.8.5 CycleStartNettime

Name:

CycleStartNettime0X

This register is used to read out the NetTime value at the start time of the last IO-Link cycle.

For additional information about NetTime and timestamps, see "[NetTime Technology](#)" on page 26.

Data type	Values
INT	-32768 to 32767
DINT	-2147483648 to 2147483647

### 5.8.6 CycleEndNettime

Name:

CycleEndNettime0X

This register is used to read out the NetTime value at the end time of the last IO-Link cycle.

For additional information about NetTime and timestamps, see "[NetTime Technology](#)" on page 26.

Data type	Values
INT	-32768 to 32767
DINT	-2147483648 to 2147483647

## 5.9 IO-Link event interface

The event interface involves interrupt-controlled background communication. It enables the connected IO-Link devices to transmit special messages, or "event codes", to the master.

### 5.9.1 EventPortSeq

Name:  
EventPortSeq

As soon as a new event is generated by an IO-Link device, the sequence number is increased in this register. The affected channel number is also displayed.

Data type	Values
USINT	0 to 255

Bit structure:

Bit	Description	Value	Information
0 - 3	Sequence number	0 to 15	
4 - 6	IO-Link channel number	001	IF1 (channel 1)
		010	IF2 (channel 2)
		011	IF3 (channel 3)
		100	IF4 (channel 4)
7	Reserved	0	

### 5.9.2 EventQualifier

Name:  
EventQualifier

This register contains additional information about the event.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0 - 2	Instance layer that generated the event	000	Unknown
		001	Hardware
		010	Data exchange layer of the IO-Link device
		011	Application layer of the IO-Link device
		100	Application
3	Cause of the event	0	Device
		1	Master
4 - 5	Type of event	00	Reserved
		01	Information
		10	Warning
		11	Error
6 - 7	Event mode	00	Reserved
		01	One-time event
		10	Event no longer reported (e.g. voltage OK again)
		11	Event reported (e.g. voltage too low)

### 5.9.3 EventCode

Name:  
EventCode

This register contains the event code of the transferred event. The event codes can consist of vendor-specific event codes or event codes specified by the IO-Link specification.

Data type	Values
UINT	0 to 65535

### 5.9.4 EventsLeft

Name:  
EventsLeft

This register specifies the number of events in FIFO memory that have not yet been processed.

Data type	Values
USINT	0 to 15

### 5.9.5 EventQuit

Name:  
EventQuit

This register can be used to acknowledge events. This is done by copying the sequence number of the event to be acknowledged to this register.

Data type	Values
USINT	0 to 15

### 5.9.6 EventQuitReadBack

Name:  
EventQuitReadBack

This register contains the sequence number of the last acknowledged event.

Data type	Values
USINT	0 to 15

## 5.10 IO-Link parameter server

The parameter server permits the module to read configuration parameters of the connected IO-Link device. The data of the third-party device is stored in the EEPROM and can then be restored automatically, e.g. after replacing the IO-Link device.

### 5.10.1 DsControl

Name:  
DsControl0X

This register is used to manually control the ["parameter server" on page 19](#). Each action is executed exactly one time when the corresponding value is set. If the same action should be executed several times, this register must be set to the value 0 between them.

Data type	Values	Information
USINT	0	No action (bus controller default setting)
	1	Parameter server operating mode: Automatic upload and download
	2	Upload if data storage parameters are available in the device
	3	Download if data storage parameters are available in controller memory and the device can process data memory parameters
	4	Acknowledge error state from parameter server. (See <a href="#">"ChannelStatus" on page 72</a> : Error messages 21 to 28.)
	5	Delete data memory parameters in controller memory
	6	Start dummy upload. Starts an upload without saving the data. Can be used to acknowledge an upload request.
	7 to 255	Reserved

### 5.10.2 DsProgress

Name:  
DsProgress0X

The module uses these registers to report the progress of the upload or download from the parameter server. Values from 0 to 100 can be used to implement a progress indicator, for example.

Data type	Values
USINT	0 to 100

## Register description

### 5.10.3 CfO\_DS\_Config

Name:

CfO\_DS\_Config0X

The parameter server behavior can be set using these registers (when the parameter server is operated manually). A corresponding reaction can be assigned to each trigger event here.

Data type	Values	Bus controller default setting
UDINT	See the bit structure.	0

Bit structure:

Bit	Event	Value	Reaction
0 - 3	The device ID of the connected device does not match the device ID stored together with the parameters.	000	No reaction (bus controller default setting)
		001	Cancel
		010	User-defined reaction. See "ChannelStatus" on page 72: Status message 25
		011	Upload
4 - 7	The device transmitted an upload request.	000	No reaction (bus controller default setting)
		001	Cancel
		010	User-defined reaction. See "ChannelStatus" on page 72: Status message 26
		011	Upload
8 - 11	A new parameter checksum was detected during device startup.	000	No reaction (bus controller default setting)
		001	Cancel
		010	User-defined reaction. See "ChannelStatus" on page 72: Status message 27
		011	Upload
		100	Download
12 - 15	The serial number of the connected device does not match the serial number stored together with the parameters.	000	No reaction (bus controller default setting)
		001	Cancel
		010	User-defined reaction. See "ChannelStatus" on page 72: Status message 24
		011	Upload
		100	Download
16 - 23	Reserved	-	
24 - 26	Specifies the order in which the individual events are checked.	000	Device ID, serial number, upload request, parameter checksum (bus controller default setting)
		001	Device ID, serial number, parameter checksum, upload request
		010	Device ID, upload request, parameter checksum, serial number
		011	Device ID, upload request, serial number, parameter checksum
		100	Device ID, parameter checksum, upload request, serial number
		101	Device ID, parameter checksum, serial number, upload request
27 - 31	Reserved	-	

## 5.11 Saving IO-Link data

Some IO-Link devices must be instructed to save imported data storage parameters remanently after a download.

### 5.11.1 CfO\_DS\_SaveCtrl

Name:

CfO\_DS\_SaveCtrl0X

This register is used together with "[CfO\\_DS\\_SaveData](#)" on page 77.

Some IO-Link devices must be instructed to save imported data storage parameters remanently after a download. To apply these parameters to the remanent memory of these devices, the index and subindex stored in these registers must be transmitted together with the memory command (e.g. value 163 on index 2, subindex 0).

Data type	Values	Bus controller default setting
UDINT	See the bit structure.	0

Bit structure:

Bit	Description	Value	Information
0 - 15	Index	0 to 255	Device-specific index for the save command
16 - 24	Subindex	0 to 255	Device-specific subindex for the save command
24 - 26	Data length	0	Save command disabled
		1 to 4	Data length of the save command expected by the device in bytes
27 - 31	Reserved		

### 5.11.2 CfO\_DS\_SaveData

Name:

CfO\_DS\_SaveData0X

This register is used together with "[CfO\\_DS\\_SaveCtrl](#)" on page 77 and contains the value that is written to the index configured in register CfO\_DS\_SaveCtrl.

Data type	Values	Information
UDINT	0 to 4,294,967,295	Bus controller default setting: 0

## 5.12 IO-Link timestamp

The IO-Link timestamp registers allow the assignment of IO-Link timestamps to the NetTime of a controller, and vice versa.

### 5.12.1 IoLinkTimestampIn

Name:

IoLinkTimestampIn0X

This register indicates the NetTime instant at which the application event occurred.

For additional information about NetTime and timestamps, see ["NetTime Technology" on page 26](#).

Data type	Values
INT	-32768 to 32767
DINT	-2147483648 to 2147483647

### 5.12.2 IoLinkTimestampInStatusSeq

Name:

IoLinkTimestampInStatusSeq0X

This register indicates information about the [input timestamp](#).

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0 - 3	Sequence number	0 to 15	The sequence number is incremented by 1 with each valid timestamp received. In the event that the sequence number has been increased by more than 1, an event has been lost.
4	Event 1 triggered by the application	x	Signal state at occurrence of the timestamp
5	Event 2 triggered by the application	x	Signal state at occurrence of the timestamp <b>Example:</b> Signal state at occurrence of the timestamp – Light barrier was interrupted → This bit = 0 – Light barrier free → This bit = 1
6	Reserved	-	
7	Timestamp error	0	No error
		1	An error occurred on the IO-Link device. Possible causes: • More timestamps were generated than could be transferred. • The value of the IO-Link timestamp exceeded the permissible range of values. In both cases, reducing the IO-Link cycle time can help.

### 5.12.3 IoLinkTimestampOut

Name:

IoLinkTimestampOut0X

The user can write the NetTime for the output timestamp to this register.

The NetTime is automatically converted to an IO-Link timestamp. The event is triggered at the defined NetTime. Register ["IoLinkTimestampOutStatus" on page 79](#) is used for acknowledgment.

For additional information about NetTime and timestamps, see ["NetTime Technology" on page 26](#).



#### Information:

**The NetTime must be at least three IO-Link cycles in the future; otherwise, a warning is set in IoLinkTimestampOutStatus.**

**The data type of this register must be identical to the data type defined in bit 26 of register ["ChannelMode" on page 64](#).**

Data type	Values	Information
INT	-32768 to 32767	Bus controller default setting: 0
DINT	-2147483648 to 2147483647	

### 5.12.4 IoLinkTimestampOutCtrlSeq

Name:

IoLinkTimestampOutCtrlSeq0X

This register is used to control how the [timestamp](#) is applied.

Data type	Values	Bus controller default setting
USINT	See the bit structure.	0

Bit structure:

Bit	Description	Value	Information
0 - 3	Sequence number	0 to 15	The output timestamp and application event bits are applied when the sequence number is incremented by 1.
4	Application event 1	x	Initial state at timestamp
5	Application event 2	x	Initial state at timestamp
6	Acknowledge warning	0	Do not acknowledge (bus controller default setting)
		1	Acknowledge warning
7	Acknowledge error	0	Do not acknowledge (bus controller default setting)
		1	Acknowledge error

### 5.12.5 IoLinkTimestampOutStatus

Name:

IoLinkTimestampOutStatus0X

This register is used to indicate the status of the [output timestamp](#).

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0 - 3	Sequence number acknowledgment	0 to 15	If an output timestamp could be applied, then the sequence number from " <a href="#">IoLinkTimestampOutCtrlSeq</a> " on page 79 is acknowledged here.
4 - 5	Reserved	-	
6	Warning	0	No warning
		1	A timestamp was not at least 3 cycles ahead, so its output may have been delayed.
7	Error	0	No error
		1	More timestamps were transferred to the module than could be output.

## 5.13 IO-Link device IDs

IO-Link device IDs are defined by the manufacturer of the IO-Link device and cannot be modified by the user.

### 5.13.1 VendorId

Name:

VendorId0X

This register contains the unique vendor ID of the IO-Link device.

Data type	Values
UINT	0 to 65535

### 5.13.2 DeviceId

Name:

DeviceId0X

This register contains the unique ID of the IO-Link device.

Data type	Values
UDINT	0 to 4,294,967,295

## Register description

### 5.13.3 FunctionId

Name:

FunctionId0X

This register contains the function class of the device assigned by the vendor.

Data type	Values
UINT	0 to 65535

### 5.13.4 CycleTime

Name:

CycleTime0X

Some IO-Link devices cannot cope with high-speed cycles and require a higher cycle time. This register can be used to read back the current IO-Link cycle time of the channel. The time used for communication is always a multiple of 100  $\mu$ s, e.g. 50 for a 5 ms cycle time.

Data type	Values	Information
UINT	0 to 65535	Specified in 1 $\mu$ s increments

### 5.13.5 CycleMultiple

Name: CycleMultiple0X

This register can be used to read back the ["multiplier" on page 68](#) currently applied for the IO-Link cycle.

Data type	Values
UINT	0 to 65535

### 5.13.6 CycleDivisor

Name:

CycleDivisor0X

This register can be used to read back the ["divisor" on page 68](#) currently applied for the IO-Link cycle.

Data type	Values
UINT	0 to 65535

### 5.13.7 MinCycleTime

Name:

MinCycleTime0X

The minimum IO-Link cycle time can be read back in this register. The minimum IO-Link cycle time depends on the IO-Link device and is read out by the module after communication with the IO-Link device has been established.

Data type	Values
UINT	0 to 65535

### 5.13.8 PDI\_Size

Name:

PDI\_Size0X

The size of the input process data specified by the device can be read back in this register. This value is read out during startup of the IO-Link device.

Data type	Values
USINT	0 to 255

### 5.13.9 PDO\_Size

Name:

PDO\_Size0X

The size of the output process data defined by the IO-Link device can be read back in this register. This value is read out during startup of the IO-Link device.

Data type	Values
USINT	0 to 255



### 5.13.10 Baud rate

Name:

Baudrate0X

The baud rate specified by the IO-Link device can be read back in this register. This value is read out during startup of the IO-Link device.

Data type	Values	Information
USINT	1	COM1 = 4.8 kbit/s
	2	COM2 = 38.4 kbit/s
	3	COM3 = 230.4 kbit/s

### 5.13.11 IoLinkVersionID

Name:

IoLinkVersionID0X

The IO-Link version can be read back in this register.

Data type	Values	Information
USINT	16 (0x10)	V1.0
	17 (0x11)	V1.1

## 5.14 Statistics counter

Communication errors that have occurred between individual IO-Link components are mapped in the statistics counter.

### 5.14.1 Number of command retries

Name:

RetryCnt0X

These registers contain the number of command retries caused by communication errors between the I/O processor and IO-Link device.

Data type	Values
UINT	0 to 65535

### 5.14.2 Number of checksum errors for the controller

Name:

SpiErrorCnt0X

These registers contain the number of checksum errors between the I/O processor and channel-specific IO-Link interface.

Data type	Values
UINT	0 to 65535

### 5.14.3 Number of communication errors

Name:

TransmErrCnt0X

These registers contain the number of communication errors between the I/O processor and channel-specific IO-Link interface.

Data type	Values
UINT	0 to 65535

### 5.14.4 Number of parity errors

Name:

ParityErrCnt0X

These registers contain the number of parity errors between the channel-specific IO-Link interface and IO-Link device.

Data type	Values
UINT	0 to 65535

### 5.14.5 Number of protocol errors

Name:

FrameErrCnt0X

These registers contain the number of protocol errors between the channel-specific IO-Link interface and IO-Link device.

Data type	Values
UINT	0 to 65535

### 5.14.6 Number of byte count errors

Name:

RxSizeErrCnt0X

These registers contain the number of faulty bytes received between the channel-specific IO-Link interface and IO-Link device.

Data type	Values
UINT	0 to 65535

### 5.14.7 Number of checksum errors for the IO-Link device

Name:

RxChksumErrCnt0X

These registers contain the number of checksum errors between the channel-specific IO-Link interface and IO-Link device.

Data type	Values
UINT	0 to 65535

### 5.14.8 Number of response errors

Name:

DeviceDlyErrCnt0X

These registers contain the number of response errors. These occur if the IO-Link device does not respond in time to the request frame of the master or if the pause between the individual bytes in the response frame is too large.

Data type	Values
UINT	0 to 65535

### 5.14.9 Number of cycle errors

Name:

CycleTimeErrorCnt0X

These registers contain the number of cycle errors. These occur if an IO-Link cycle is started before the previous one could be completed and processed. These errors can be corrected by setting a lower cycle time.

Data type	Values
UINT	0 to 65535

## 5.15 IO-Link communication via the command interface

The command interface provides the possibility of accessing the object dictionary of the IO-Link device via index and subindex. Alternatively, access can also take place using library AsIoLink or the Flatstream.

### 5.15.1 ParameterIndexOut

Name:

ParameterIndexOut

This register is used to define the index of the object in the object dictionary that should be accessed.

Data type	Values
UINT	0 to 65535

### 5.15.2 ParameterSubIndexOut

Name:

ParameterSubIndexOut

This register is used to define the subindex of the object in the object dictionary that should be accessed.

Data type	Values
USINT	0 to 255

### 5.15.3 ParameterCtrlOut

Name:

ParameterCtrlOut

This register is used to define the type of access desired.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0 - 1	Sequence number	0 to 3	
2 - 3	IO-Link channel number	0	IF1 (channel 1)
		1	IF2 (channel 2)
		2	IF3 (channel 3)
		3	IF4 (channel 4)
4 - 6	Data length	0 to 4	
7	Read or write	0	Read
		1	Write

### 5.15.4 ParameterDataOut

Name:

ParameterDataOut\_0

This register contains the data that should be written.

Data type	Values
UDINT	0 to 4,294,967,295

## Register description

### 5.15.5 ParameterCtrlIn

Name:

ParameterCtrlIn

This register is used to monitor the access status.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Description	Value	Information
0 - 1	Sequence confirmation	0 to 3	
2 - 3	IO-Link channel number	0	IF1 (channel 1)
		1	IF2 (channel 2)
		2	IF3 (channel 3)
		3	IF4 (channel 4)
4 - 6	Data length	0 to 4	
7	Error bit	0	No error
		1	Error. The <a href="#">error code</a> is indicated in "ParameterDataIn" on <a href="#">page 84</a> .

### 5.15.6 ParameterDataIn

Name:

ParameterDataIn\_0

This register contains the input data after successful read access or the [error codes](#) in the event of error.

Data type	Values
UDINT	0 to 4,294,967,295

#### Error display

- If the ["error code" on page 18](#) is not equal to 8 (i.e. error reported by the device), then the LSB contains the error code.
- In the event of an error reported by the device, the error specified by the IO-Link device is also displayed.

UDINT			
MSB			LSB
Reserved	IO-Link error code	Additional IO-Link error code	8

## 5.16 Flatstream registers

At the absolute minimum, registers "InputMTU" and "OutputMTU" must be set. All other registers are filled in with default values at the beginning and can be used immediately. These registers are used for additional options, e.g. to transfer data in a more compact way or to increase the efficiency of the general procedure.



### Information:

For detailed information about Flatstream, see [Flatstream communication](#).

### 5.16.1 Number of enabled Tx and Rx bytes

Name:

OutputMTU

InputMTU

These registers define the number of enabled Tx or Rx bytes and thus also the maximum size of a sequence. The user must consider that the more bytes made available also means a higher load on the bus system.

Data type	Values
USINT	See the register overview.

### 5.16.2 Transporting payload data and control bytes

Name:

TxByte1 to TxByteN

RxByte1 to RxByteN

(The value the number N is different depending on the bus controller model used.)

The Tx and Rx bytes are cyclic registers used to transport the payload data and the necessary control bytes. The number of active Tx and Rx bytes is taken from the configuration of registers "OutputMTU" and "InputMTU", respectively.

- "T" - "Transmit" → Controller transmits data to the module.
- "R" - "Receive" → Controller receives data from the module.

Data type	Values
USINT	0 to 255

### 5.16.3 Communication status of the controller

Name:

OutputSequence

This register contains information about the communication status of the controller. It is written by the controller and read by the module.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0 - 2	OutputSequenceCounter	0 - 7	Counter for the sequences issued in the output direction
3	OutputSyncBit	0	Output direction (disable)
		1	Output direction (enable)
4 - 6	InputSequenceAck	0 - 7	Mirrors InputSequenceCounter
7	InputSyncAck	0	Input direction not ready (disabled)
		1	Input direction ready (enabled)

#### OutputSequenceCounter

The OutputSequenceCounter is a continuous counter of sequences that have been issued by the controller. The controller uses OutputSequenceCounter to direct the module to accept a sequence (the output direction must be synchronized when this happens).

#### OutputSyncBit

The controller uses OutputSyncBit to attempt to synchronize the output channel.

#### InputSequenceAck

InputSequenceAck is used for acknowledgment. The value of InputSequenceCounter is mirrored if the controller has received a sequence successfully.

#### InputSyncAck

The InputSyncAck bit acknowledges the synchronization of the input channel for the module. This indicates that the controller is ready to receive data.

## Register description

### 5.16.4 Communication status of the module

Name:

InputSequence

This register contains information about the communication status of the module. It is written by the module and should only be read by the controller.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0 - 2	InputSequenceCounter	0 - 7	Counter for sequences issued in the input direction
3	InputSyncBit	0	Not ready (disabled)
		1	Ready (enabled)
4 - 6	OutputSequenceAck	0 - 7	Mirrors OutputSequenceCounter
7	OutputSyncAck	0	Not ready (disabled)
		1	Ready (enabled)

#### InputSequenceCounter

The InputSequenceCounter is a continuous counter of sequences that have been issued by the module. The module uses InputSequenceCounter to direct the controller to accept a sequence (the input direction must be synchronized when this happens).

#### InputSyncBit

The module uses InputSyncBit to attempt to synchronize the input channel.

#### OutputSequenceAck

OutputSequenceAck is used for acknowledgment. The value of OutputSequenceCounter is mirrored if the module has received a sequence successfully.

#### OutputSyncAck

The OutputSyncAck bit acknowledges the synchronization of the output channel for the controller. This indicates that the module is ready to receive data.

### 5.16.5 Flatstream mode

Name:

FlatstreamMode

A more compact arrangement can be achieved with the incoming data stream using this register.

Data type	Values
USINT	See the bit structure.

Bit structure:

Bit	Name	Value	Information
0	MultiSegmentMTU	0	Not allowed (default)
		1	Permitted
1	Large segments	0	Not allowed (default)
		1	Permitted
2 - 7	Reserved		

### 5.16.6 Number of unacknowledged sequences

Name:

Forward

With register "Forward", the user specifies how many unacknowledged sequences the module is permitted to transmit.

Recommendation:

X2X Link: Max. 5

POWERLINK: Max. 7

Data type	Values
USINT	1 to 7 Default: 1

### 5.16.7 Delay time

Name:

ForwardDelay

This register is used to specify the delay time in microseconds.

Data type	Values
UINT	0 to 65535 [ $\mu$ s] Default: 0

## 5.17 Minimum cycle time

The minimum cycle time specifies how far the bus cycle can be reduced without communication errors occurring. It is important to note that very fast cycles reduce the idle time available for handling monitoring, diagnostics and acyclic commands.

Minimum cycle time	
Without IO-Link (all channels in SIO mode)	$\geq 400 \mu$ s
With IO-Link	$\geq 1$ ms

## 5.18 Minimum I/O update time

The minimum I/O update time specifies how far the bus cycle can be reduced so that an I/O update is performed in each cycle.

Minimum I/O update time	
Without IO-Link (all channels in SIO mode)	$\geq 400 \mu$ s
With IO-Link	$\geq 1$ ms (depending on the minimum IO-Link cycle time of the IO-Link device)