



Programming manual  
Process interface

GB

**O2D5xx**  
**O2I4xx**  
**O2I5xx**

TCP | EtherNet/IP | PROFINET | IO-Link

## Contents

1	Preliminary note	5
1.1	Symbols used	5
1.2	Legal and copyright information	5
1.3	Disclaimer of warranties	5
2	Safety instructions	7
3	TCP	8
3.1	Sending commands	8
3.2	Receiving images	10
3.3	Image data	10
3.4	Process interface command reference	12
3.4.1	a Command (activate application)	12
3.4.2	A? Command (occupancy of application list)	12
3.4.3	b Command (execute configured button)	13
3.4.4	c Command (upload process interface output configuration)	13
3.4.5	C? Command (retrieve current process interface configuration)	14
3.4.6	d Command (turn viewindicator on or off)	14
3.4.7	E? Command (request current error state)	14
3.4.8	f Command (set temporary application parameter)	16
3.4.9	F? Command (get temporary application parameter)	16
3.4.10	g Command (gated software trigger on or off)	17
3.4.11	G? Command (request device information)	17
3.4.12	H? Command (return a list of available commands)	18
3.4.13	I? Command (request last image taken)	18
3.4.14	j Command (overwrites data of a string)	19
3.4.15	J? Command (read string from defined container)	19
3.4.16	L? Command (returns the current session ID)	19
3.4.17	o Command (set logic state of a IO)	19
3.4.18	O? Command (request state of a IO)	20
3.4.19	p Command (turn process interface output on or off)	20
3.4.20	s Command (reset current statistics)	20
3.4.21	S? Command (request current decoding statistics)	21
3.4.22	t Command (execute asynchronous trigger)	21
3.4.23	T? Command (execute synchronous trigger)	21
3.4.24	v Command (set current protocol version)	21
3.4.25	V? Command (request current protocol version)	22
3.5	Parameter IDs	22
4	EtherNet/IP	23
4.1	Data structures for consuming and producing assemblies	23
4.2	Command handling	24
4.2.1	Initialization of assembly buffers	24
4.2.2	Default endianness	24
4.2.3	Command execution	24
4.2.4	Handling of multiple command bits	26
4.2.5	Blocking of asynchronous messages	26
4.2.6	Client disconnect	26
4.2.7	Sending of response message to a command	26
4.2.8	Reset of error bit	27
4.2.9	Cueing of error codes	27
4.2.10	Functionality of asynchronous message bit	27
4.2.11	Bits for asynchronous message identifier	27
4.2.12	Message counter	27
4.3	Command description	28
4.3.1	Get last error	28
4.3.2	Get connection ID	28
4.3.3	Get statistics	28
4.3.4	Activate application	29
4.3.5	Get application list	29
4.3.6	Aborting application evaluation on application switch	29



4.3.7	Get IO state . . . . .	29
4.3.8	Set IO state . . . . .	30
4.3.9	Execute button function . . . . .	31
4.3.10	Execute synchronous trigger . . . . .	31
4.3.11	Execute gated software trigger . . . . .	31
4.3.12	Activate asynchronous data output . . . . .	32
4.3.13	Use extended command . . . . .	32
4.3.14	Get next data segment . . . . .	32
4.4	Extended commands . . . . .	32
4.4.1	ID 3: Enable / Disable data partitioning . . . . .	32
4.4.2	ID 4: Set the content of a specific match string container . . . . .	33
4.4.3	ID 5: Get the content of a specific match string container . . . . .	34
4.4.4	ID 6: Toggle the viewindicator on/off . . . . .	34
4.4.5	ID 7/8: Set/Get the current focus distance . . . . .	34
4.4.6	ID 9: Reset current statistics . . . . .	34
5	PROFINET . . . . .	35
5.1	Stack requirements . . . . .	35
5.2	Command structure . . . . .	37
5.2.1	Available slots . . . . .	37
5.2.2	Verifying the size of selected slots on connection setup . . . . .	38
5.2.3	Module size for slot 2 . . . . .	38
5.2.4	Module sizes for slots 3...12 . . . . .	38
5.2.5	Module size for slot 13 . . . . .	38
5.2.6	Truncation of data . . . . .	38
5.2.7	Data layout of slot 1 - process status . . . . .	39
5.2.8	Data layout of slots 2...12 . . . . .	39
5.2.9	Data layout of slot 13 . . . . .	40
5.2.10	Data layout of command word . . . . .	40
5.2.11	Identifier for synchronous and asynchronous messages . . . . .	40
5.3	Command handling . . . . .	41
5.3.1	Number of supported PROFINET connections . . . . .	41
5.3.2	Initialization of input and output buffers . . . . .	41
5.3.3	Default endianness . . . . .	41
5.3.4	Command execution . . . . .	41
5.3.5	Handling of multiple command bits . . . . .	42
5.3.6	Blocking of asynchronous messages . . . . .	42
5.3.7	Handling of disconnections during command execution . . . . .	42
5.3.8	Sending of response message to a command . . . . .	42
5.3.9	Reset of error bit . . . . .	43
5.3.10	Queuing of error codes . . . . .	43
5.3.11	Functionality of asynchronous message bit . . . . .	43
5.3.12	Bits for asynchronous message identifier . . . . .	43
5.3.13	Message counter . . . . .	43
5.4	Command description . . . . .	43
5.4.1	Execute gated software trigger . . . . .	43
5.4.2	Get last error . . . . .	44
5.4.3	Get connection ID . . . . .	45
5.4.4	Get statistics . . . . .	45
5.4.5	Activate application . . . . .	45
5.4.6	Get application list . . . . .	46
5.4.7	Get IO state . . . . .	46
5.4.8	Set IO state . . . . .	47
5.4.9	Execute button function . . . . .	47
5.4.10	Execute synchronous trigger . . . . .	47
5.4.11	Activate asynchronous data output . . . . .	48
5.4.12	Use extended command . . . . .	48
5.4.13	Get next data segment . . . . .	48
5.5	Extended commands . . . . .	48
5.5.1	ID 3: Enable / Disable data partitioning . . . . .	48
5.5.2	ID 4: Set the content of a specific match string container . . . . .	50
5.5.3	ID 5: Get the content of a specific match string container . . . . .	50

5.5.4	ID 6: Toggle the viewindicator on/off . . . . .	50
5.5.5	ID 7/8: Set/Get the current focus distance . . . . .	50
5.5.6	ID 9: Reset current statistics . . . . .	51
5.6	GSDML file . . . . .	51
5.7	I&M data . . . . .	53
6	IO-Link . . . . .	56
6.1	Data structures . . . . .	56
6.1.1	Semantics of command counter / message counter mirror . . . . .	56
6.2	Operating modes . . . . .	57
6.2.1	Normal operation . . . . .	57
6.2.2	Command execution . . . . .	58
6.3	Segmentation of payload data . . . . .	59
6.3.1	Data segmentation and segment counting . . . . .	59
6.3.2	Output payload data . . . . .	60
6.3.3	Input payload data . . . . .	60
6.4	Commands . . . . .	61
6.4.1	Software trigger . . . . .	61
6.4.2	Application switch . . . . .	61
6.4.3	Get application list . . . . .	62
6.4.4	Get statistics . . . . .	62
6.4.5	Reset statistics . . . . .	62
6.4.6	Set input string . . . . .	62
6.4.7	Get input string . . . . .	63
6.4.8	Set focus distance . . . . .	63
6.4.9	Get focus distance . . . . .	63
6.4.10	Execute button teach . . . . .	63
6.4.11	Extended commands . . . . .	64
7	Error codes . . . . .	65
	Glossar . . . . .	67

# 1 Preliminary note

You will find instructions, technical data, approvals and further information using the QR code on the unit / packaging or at [documentation.ifm.com](https://documentation.ifm.com).

## 1.1 Symbols used

- ✓ Requirement
- Instructions
- ▷ Reaction, result
- [...] Designation of keys, buttons or indications
- Cross-reference
-  Important note  
Non-compliance may result in malfunction or interference.
-  Information  
Supplementary note

## 1.2 Legal and copyright information

© All rights reserved by ifm electronic gmbh. No part of these instructions may be reproduced and used without the consent of ifm electronic gmbh.

All product names, pictures, companies or other brands used on our pages are the property of the respective rights owners.

## 1.3 Disclaimer of warranties

ifm electronic gmbh disclaims to the fullest extent authorized by law any and all warranties, whether express or implied, including, without limitation, any implied warranties of title, non-infringement, quiet enjoyment, integration, merchantability or fitness for a particular purpose.

Without limitation of the foregoing, ifm expressly does not warrant that:

- the software will meet your requirements or expectations,
- the software or the software content will be free of bugs, errors, viruses or other defects,
- any results, output, or data provided through or generated by the software will be accurate, up-to-date, complete or reliable,
- the software will be compatible with third party software,
- any errors in the software will be corrected.

### Demo software and templates

Demo software and templates are provided “as is” (that is: excluding warranty) and “as available”, without any warranty of any kind, either express or implied. The user acknowledges and agrees to use the software at user’s own risk. In no event shall ifm be held liable for any direct, indirect, incidental or consequential damages arising out of the use of or incorrect use the software. The user may use the software solely for demonstration purposes and to assess the software functionalities and capabilities.

### Customer-specific software

1. The software created and used has been put together by ifm especially for the customer using modular software components made by ifm for numerous applications (standard software modules) and adapted to the contractual service required (customer-specific application program).

2. Upon complete payment of the purchase price for the customer-specific application program, ifm transfers the non-exclusive, locally and temporarily unrestricted usage right thereof to the customer, without the customer acquiring any rights of any kind to the standard software module on which the individual or customer-specific adaptation is based. Notwithstanding these provisions, ifm reserves the right to produce and offer customer-specific software solutions of the same kind for other customers based on other terms of reference. In any case ifm retains a simple right of usage of the customer-specific solution for internal purposes.
3. By accepting the program, the user acknowledges and agrees to use the software at user's own risk. By accepting the program, the user also acknowledges that the software meets the requirements of the specifications agreed upon. ifm disclaims any and all warranties, in particular regarding fitness of the software for a particular purpose.

## 2 Safety instructions

Please read the operating instructions prior to set-up of the device. The device must be suitable for the application without any restrictions.

If the operating instructions or the technical data are not adhered to, personal injury and damage to property can occur.

## 3 TCP

The TCP process interface is used during the normal operation mode to get operational data from the device (e.g. images, process values).



The port for TCP communication is preset to 50010. A different port can be set using the ifm Vision Assistant software:

- Set in the software in the area [Device setup] -> [Interfaces] -> [TCP/IP port for PCIC]

### 3.1 Sending commands

For sending commands via the process interface the commands must be sent with a special protocol and as ASCII character strings. There are different protocol versions available:

Version	Input format	Output format
V1	<content>CR LF	as input
V2	<ticket><content>CR LF	as input
V3	<ticket><length>CR+LF<ticket><content>CR LF	as input



The default protocol version is "V3". It is recommended to use protocol version 3 for machine-to-machine communication. This is because only version 3 supports asynchronous messages and provides length information.

#### Structure of the protocol

```
<Ticket><length>CR LF<Ticket><content>CR LF
```

Abbreviation	Description	ASCII code (dec)	ASCII code (hex)
CR	Carriage Return	13	D
LF	Linefeed	10	A
< >	Marking of a placeholder (e.g. <code> is a placeholder for code)		
[ ]	Optional argument (possible but not required)		

Command	Description
<content>	The command to the device (e.g. trigger the unit).
<ticket>	It is a character string of 4 digits between 0...9. If a message with a specific ticket is sent to the device, it will reply with the same ticket. A ticket number must be > 0999. Use a ticket number from the range 1000...9999.
<length>	It is a character string beginning with the letter 'L' followed by 9 digits. It indicates the length of the following data (<ticket><content>CR LF) in bytes.

#### Ticket numbers for asynchronous messages:

Ticket number	Description
0000	Asynchronous results
0001	Asynchronous error messages / codes
0010	Asynchronous notifications / message codes



### Format of asynchronous notifications

The format of the asynchronous notifications is a combination of the unique message ID and a **JSON** formatted string containing the notification details:

```
<unique message ID>:<JSON content>
```

Example for protocol version 3:

```
<ticket=0010>L<length>CR LF<ticket=0010><unique message ID>:<JSON content>CR LF
```

Result:

```
0010L000000045\r\n0010000500000:{"ID": 1034160761,"Index":1,"Name": "Pos 1"}\r\n
```

Explanation of the result:

Command	Result
L<length>	L000000045
CR LF	\r\n
<ticket=0010>	0010
<unique message ID>	000500000
<JSON content>	{"ID": 1034160761,"Index":1,"Name": "Pos 1"}
CR LF	\r\n

### Asynchronous message IDs

Asynchronous message ID	Description	Example	Description
000500000	Application changed	{"ID": 1034160761,"Index":1,"Name": "Pos 1","valid":true}	
000500001	Application is not valid	{"ID": 1034160761,"Index":1,"Name": "Pos 1","valid":false}	If an application exists on given index but it is invalid, the ID and name are filled according to the application. If there is no application on given index, the application ID will contain "0" and the name an empty string "".
000500002	image acquisition finished	{}	The message signals the receiver that the device has finished the image acquisition. This can be used for cascading multiple devices with a software trigger.
000500003	Network settings changed	{"MACAddress": "00:02:01:42:12:97", "UseDHCP":false, "StaticIPv4Address": "192.168.0.69", "StaticIPv4SubnetMask": "255.255.255.0", "StaticIPv4Gateway": "192.168.0.201"}	If the network settings are changed with a teach code or by XML-RPC, the device sends the message to report the new settings to a <b>PCIC</b> client.

## 3.2 Receiving images

For receiving the image data, a TCP/IP socket communication is established. The default port number is 50010. The port number may differ based on the configuration. After opening the socket communication, the device (if the device is in free run mode) will automatically send the data through this socket to the TCP/IP client (PC).

**PCIC** output per frame. The following data is submitted in this sequence:

Component	Content
Ticket and length information	Sending commands ( <a href="#">→ Sending commands ▢ 8</a> )
Ticket	0000
Encoded JPEG image(s)	1...5 image(s)
Ticket signature	<CR><LF>

## 3.3 Image data

For every image there will be a separate chunk. The chunk is part of the response frame data of the process interface.

The header of each chunk contains different kinds of information. This information is separated into bytes. For example, the information contains the kind of image which will be in PIXEL\_DATA and the size of the chunk.

### Serialization format with header version 2

This is the serialization format of the image (matrix) data with header version 2. All information is stored in Little Endian format.

Offset	Name	Description	Size [byte]
0x0000	CHUNK_TYPE	Defines the type of the chunk.	4
0x0004	CHUNK_SIZE	Size of the whole image chunk in bytes. After this count of bytes, the next chunk starts.	4
0x0008	HEADER_SIZE	Number of bytes starting from 0x0000 until PIXEL_DATA.	4
0x000C	HEADER_VERSION	Version number of the header: 2.	4
0x0010	IMAGE_WIDTH	Image width in pixels.	4
0x0014	IMAGE_HEIGHT	Image height in pixels.	4
0x0018	PIXEL_FORMAT	Pixel format.	4
0x001C	TIME_STAMP	Time stamp in microseconds (deprecated).	4
0x0020	FRAME_COUNT	Frame count according to algorithm output.	4
0x0024	STATUS_CODE	Errors of the device.	4
0x0028	TIME_STAMP_SEC	Timestamp in seconds.	4
0x002C	TIME_STAMP_NSEC	Timestamp in nanoseconds.	4
0x0030	PIXEL_DATA	The pixel data in the given type and dimension of the image, padded to 4-byte boundary.	

### Serialization format with header version 3

This is the serialization format of binary data with header version 3. All information is stored in Little Endian format.

Offset	Name	Description	Size [byte]
0x0000	CHUNK_TYPE	Defines the type of the chunk.	4
0x0004	CHUNK_SIZE	Size of the whole image chunk in bytes.	4
0x0008	HEADER_SIZE	Number of bytes starting from 0x0000 until BINARY_DATA. The number of bytes must be a multiple of 16, and the minimum value is 0x40 (64).	4
0x000C	HEADER_VERSION	Version number of the header: 3.	4
0x0010	IMAGE_WIDTH	Image width in pixels. Applies only if BINARY_DATA contains an image. Otherwise, this is set to the length of BINARY_DATA.	4
0x0014	IMAGE_HEIGHT	Image height in pixels. Applies only if BINARY_DATA contains an image. Otherwise, this is set to 1.	4
0x0018	PIXEL_FORMAT	Pixel format. Applies only to image binary data. For generic binary data this is set to FORMAT_8U, unless specified otherwise for a particular chunk type.	4
0x001C	TIME_STAMP	Time stamp in microseconds (deprecated).	4
0x0020	FRAME_COUNT	Continuous frame count.	4
0x0024	STATUS_CODE	Errors of the device.	4
0x0028	TIME_STAMP_SEC	Timestamp in seconds.	4
0x002C	TIME_STAMP_NSEC	Timestamp in nanoseconds.	4
0x0030	META_DATA	UTF-8 encoded null-terminated JSON object. The content of the JSON object is depending on the CHUNK_TYPE.	minimum 16, multiple of 16
HEADER_SIZE	BINARY_DATA	The binary data. 16-byte-padding is guaranteed, as HEADER_SIZE must be a multiple of 16.	multiple of 16

### Available chunk types

Constant	Value	Description
MONOCHROM_2D_12BIT	250	12-bit monochrome 2D image
MONOCHROM_2D	251	Monochrome 2D image
JPEG_IMAGE	260	JPEG encoded image

### Pixel format

Constant	Value	Description
FORMAT_8U	0	8-bit unsigned integer
FORMAT_8S	1	8-bit signed integer
FORMAT_16U	2	16-bit unsigned integer
FORMAT_16S	3	16-bit signed integer
FORMAT_32U	4	32-bit unsigned integer
FORMAT_32S	5	32-bit signed integer

Constant	Value	Description
FORMAT_32F	6	32-bit floating point number
FORMAT_64U	7	64-bit unsigned integer
FORMAT_64F	8	64-bit floating point number
FORMAT_16U2	9	2x 16-bit unsigned integer
FORMAT_32F3	10	3x 32-bit floating point
FORMAT_12U	11	12-bit unsigned integer (nested)

## 3.4 Process interface command reference

All received messages which are sent because of the following commands will be sent without `start` / `stop` at the beginning or ending of the string.

### 3.4.1 a Command (activate application)

Command	a<application number>	
Description	Activates the selected application	
Type	Action	
Reply	*	
	!	<ul style="list-style-type: none"> <li>Application not available.</li> <li>&lt;application number&gt; contains wrong value.</li> <li>External application switching activated.</li> <li>Device is in an invalid state for the command, e.g. configuration mode.</li> <li>For devices with hardware application switching: hardware application switching is enabled.</li> </ul>
	?	Invalid command length.
Note	<application number> 2 digits for the application number as decimal value.	

### 3.4.2 A? Command (occupancy of application list)

Command	A?	
Description	Requests the occupancy of the application list	
Type	Request	
Reply	<amount><t><number active application><t> ... <number><t><number>	
	!	<ul style="list-style-type: none"> <li>Invalid state (e.g. no application active).</li> <li>No application stored on the device (E? command will return 100001002 ). <a href="#">Error codes</a> (→ <a href="#">65</a>)</li> </ul>
	?	Invalid command length.

Note	<ul style="list-style-type: none"> <li>• <code>&lt;amount&gt;</code> char string with 3 digits for the number of applications saved on the device as decimal number.</li> <li>• <code>&lt;t&gt;</code> tabulator (0x09).</li> <li>• <code>&lt;number active application&gt;</code> 2 digits for the active application.</li> <li>• <code>&lt;number&gt;</code> 2 digits for the application number.</li> </ul>	The active application is repeated within the application list.
------	--	---

### 3.4.3 b Command (execute configured button)

Command	b	
Description	Execute the currently configured button functionality	
Type	Action	
Reply	*	Teach was executed without an error
	!	<ul style="list-style-type: none"> <li>• No button function configured.</li> <li>• Button function already running.</li> <li>• Button function caused an error.</li> <li>• Device is in an invalid state for this command, e.g. configuration mode.</li> </ul>

### 3.4.4 c Command (upload process interface output configuration)

Command	c<length><configuration>	
Description	Uploads a <b>PCIC</b> output configuration lasting this session	
Type	Action	
Reply	*	
	!	<ul style="list-style-type: none"> <li>• Error in configuration: Includes a validation against a <b>JSON-schema</b> that is independent of the used element ID. Non-existing element IDs cannot be validated at the time of c-command execution, as the existence of IDs can dynamically change at run time. Changing the format should be possible even if currently no application is active.</li> <li>• Wrong data length</li> </ul>
	?	Invalid command length
Note	<ul style="list-style-type: none"> <li>• <code>&lt;length&gt;</code> 9 digits as decimal value for the data length</li> <li>• <code>&lt;configuration&gt;</code> configuration data</li> <li>• If the command is successfully issued, the PCIC output is guaranteed to never change during application switching, regardless of the state of the "PcicTcpSchemaAutoUpdate" parameter.</li> </ul>	

#### Examples

Illumination device temperature "33,5\_\_":

```
c000000226{ "layouter": "flexible", "format": { "dataencoding": "ascii" },
"elements": [ { "type": "float32", "id": "temp_illu", "format": { "width": 7,
"precision": 1, "fill": "_", "alignment": "left", "decimalseparator":
",," } } ] }
```

Illumination device temperature as binary (16-bit integer, 1/10 °C):

```
c000000194{ "layouter": "flexible", "format": { "dataencoding": "ascii" },
"elements": [ { "type": "int16", "id": "temp_illu", "format": { "dataencoding":
"binary", "order": "network", "scale": 10 } } ] }
```

Illumination device temperature in °F (e.g. "92.3 Fahrenheit"):

```
c000000227{ "layouter": "flexible", "format": { "dataencoding": "ascii" },
"elements": [ { "type": "float32", "id": "temp_illu", "format": { "precision":
1, "scale": 1.8, "offset": 32 } }, { "type": "string", "value": "
Fahrenheit" } ] }
```

### 3.4.5 C? Command (retrieve current process interface configuration)

Command	C?	
Description	Retrieves the current process interface configuration	
Type	Request	
Reply	<length><configuration>	
	?	Invalid command length
Note	<ul style="list-style-type: none"> <li>&lt;length&gt; 9 digits as decimal value for the data length</li> <li>&lt;configuration&gt; configuration data</li> </ul>	

### 3.4.6 d Command (turn viewindicator on or off)

Command	d<on-off state of viewindicator><duration>	
Description	Turn the viewindicator on or off. It does not change the global parameter that defines if the viewindicator is turned on when the camera switches to run mode.	
Type	Action	
Reply	*	
	!	<ul style="list-style-type: none"> <li>Device is not in run or simulation mode.</li> <li>The temperature is not in the right range for using the viewindicator.</li> <li>Passed duration exceeds 600 seconds.</li> </ul>
	?	Syntax error
Note	<ul style="list-style-type: none"> <li>&lt;on-off state of viewindicator&gt; 1 digit: <ul style="list-style-type: none"> <li>0 : viewfinders off</li> <li>1 : viewfinders on</li> </ul> </li> <li>&lt;duration&gt;3 digits with leading 0 : <ul style="list-style-type: none"> <li>000 : viewindicator is turned on until explicitly turned off again.</li> <li>001-600 : turn on the viewindicator for 1-600 seconds.</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>10000004 is set as an error if the time exceeds 600 seconds.</li> <li>10001013 is set as an error if the device is not in run or simulation mode.</li> <li>10001014 is set as an error if internal temperature is too high for viewindicator usage.</li> <li>10001022 if there is no viewindicator present on the device (e.g. O2x5xx).</li> </ul> <p><a href="#">Error codes (→ □ 65)</a></p>

### 3.4.7 E? Command (request current error state)

Command	E?	
Description	Requests the current error state	

Type	Request	
Reply	<code>	
	!	Invalid state (e.g. configuration mode)
	?	Invalid command length
Note	<code> Error code with 9 digits as a decimal value. It contains leading zeros.	

### 3.4.8 f Command (set temporary application parameter)

Command	f<Parameter-ID><reserved><value>	
Description	Set temporary application parameter	
	<Parameter-ID>	Parameter ID to be set. Fixed 5 bytes decimal ASCII padded with 0 e.g. 00003. Complete list of valid parameter IDs. (→ <a href="#">Parameter IDs</a> 22)
	<reserved>	Fixed to #00000.
	<value>	Fixed 5 bytes signed decimal ASCII padded with 0 and sign, e.g. +00777.
Type	Action	
Reply	*	Parameter ID successfully set.
	!	<ul style="list-style-type: none"> <li>Parameter ID invalid or syntax error (error 100001019).</li> <li>Parameter value out of range (error 100001020).</li> <li>Session not available (error 100001021).</li> <li>Device is not in run or simulation mode (error 100001013).</li> </ul> <a href="#">Error codes</a> (→ 65)
	?	Invalid command length.

#### Example

```
f00003#00000+00777
```

### 3.4.9 F? Command (get temporary application parameter)

Command	F<Parameter-ID>?	
Description	Get temporary application parameter	
	<Parameter-ID>	Parameter ID to get. Fixed 5 bytes decimal ASCII padded with 0 e.g. 00003. Complete list of valid parameter IDs. (→ <a href="#">Parameter IDs</a> 22)
Type	Request	
Reply	<Parameter-ID><reserved><value>	Parameter ID successfully obtained. <Parameter-ID> : repeated parameter ID from the request above. <reserved> : fixed to #00000 <Value> : value of the parameter. Fixed 5 bytes signed decimal ASCII padded with 0 and sign, e.g. +00777.
	!	<ul style="list-style-type: none"> <li>Parameter ID invalid or syntax error (error 100001019).</li> <li>Session not available (error 100001021).</li> <li>Device is not in run or simulation mode (error 100001013).</li> </ul> <a href="#">Error codes</a> (→ 65)
	?	Invalid command length.

#### Example

Request:



F00003?

Response:

00003#00000+00777

### 3.4.10 g Command (gated software trigger on or off)

Command	g<state>	
Description	Turn gated software trigger on or off	
Type	Request	
Reply	*	<ul style="list-style-type: none"> <li>• Trigger could be executed.</li> <li>• Gate was closed. Consecutive calls with g0 will not lead to an error.</li> </ul>
	!	<ul style="list-style-type: none"> <li>• Invalid argument</li> <li>• Invalid state</li> <li>• Trigger already executed</li> </ul>
	?	Something else went wrong.
Note	<state> 1 digit: <ul style="list-style-type: none"> <li>• 0 : turn gated software trigger off</li> <li>• 1 : turn gated software trigger on</li> </ul>	



Closing the gate though the gate is not an error case (automatically closed by the software).

### 3.4.11 G? Command (request device information)

Command	G?	
Description	Request device information	
Type	Request	
Reply	<vendor><t><article number><t><name><t><location><t><description><t><ip> <subnet mask><t><gateway><t><MAC><t><DHCP><t><port number>	
Note	<ul style="list-style-type: none"> <li>• &lt;vendor&gt; IFM ELECTRONIC</li> <li>• &lt;t&gt; Tabulator (0x09)</li> <li>• &lt;article number&gt; e.g. O2D500</li> <li>• &lt;name&gt; UTF8 Unicode string</li> <li>• &lt;location&gt; UTF8 Unicode string</li> <li>• &lt;description&gt; UTF8 Unicode string</li> <li>• &lt;ip&gt; IP address of the device as ASCII character sting, e.g. 192.168.0.96</li> <li>• &lt;port number&gt; port number of the XML-RPC</li> <li>• &lt;subnet mask&gt; subnet mask of the device as ASCII, e.g. 192.168.0.96</li> <li>• &lt;gateway&gt; gateway of the device as ASCII, e.g. 192.168.0.96</li> <li>• &lt;MAC&gt; MAC address of the device as ASCII, e.g. AA:AA:AA:AA:AA:AA</li> <li>• &lt;DHCP&gt; ASCII string:               <ul style="list-style-type: none"> <li>– 0 : DHCP off</li> <li>– 1 : DHCP on</li> </ul> </li> </ul>	

### 3.4.12 H? Command (return a list of available commands)

Command	H?	
Description	Returns a list of available commands	
Type	Request	
Reply	a - activate application A? - occupancy of application list b - execute configured button teach c - upload process interface output configuration C? - retrieve current process interface configuration d - change state of viewindicator E? - request current error state f - et application parameter value F? - get application parameter value g - gated software trigger on or off G? - request device information H? - show this list I? - request last image taken j - overwrites data of a string J<ID>? - read string from defined container L? - retrieves the connection ID o - set logic state of an ID O? - request state of an ID p - turn process interface output on or off s - reset current statistics S? - request current decoding statistics t - execute asynchronous trigger T? - execute synchronous trigger v - set current protocol version V? - request current protocol version z - change state of <b>PCIC</b> output	

### 3.4.13 I? Command (request last image taken)

Command	I<image-ID>?	
Description	Request last image taken	
Type	Request	
Reply	<length><image data>	
	!	<ul style="list-style-type: none"> <li>No image available</li> <li>Wrong ID</li> </ul>
	?	Invalid command length
Note	<ul style="list-style-type: none"> <li>&lt;image-ID&gt; 2 digits for the image type</li> <li>&lt;length&gt; char string with exactly 9 digits as decimal number for the image data size in bytes.</li> <li>&lt;image data&gt; image data / result data. The data is encapsulated in an image chunk.</li> </ul>	Valid image ID: 01 : all JPEG images 03 : reference image (must be first created using XML-RPC application edit object) 10 : last result output as formatted for this connection



If an image is requested via the I? command, the header of the JPEG file must be adjusted before saving:

► Delete all information before the start identifier.

▷ The start identifier for a JPG picture is **FF D8** (hex) / **ÿø** (ASCII).

### 3.4.14 j Command (overwrites data of a string)

Command	j<ID><length><data>	
Description	Overwrites the string data of a specific (ID) string container used in the logic layer	
Type	Action	
Reply	*	
	!	<ul style="list-style-type: none"> <li>Invalid argument or invalid state (other than run mode)</li> <li>Not existing string with input-container-ID</li> </ul>
	?	Syntax error
Note	<ul style="list-style-type: none"> <li>&lt;ID&gt; number from 00 to 09</li> <li>&lt;length&gt; 9 digits as decimal value for the data length</li> <li>&lt;data&gt; string of a maximum size of 256 bytes</li> </ul>	

### 3.4.15 J? Command (read string from defined container)

Command	J<input-container-ID>?	
Description	Read the data from the defined input string or input binary container. In both cases data is represented as byte array.	
Type	Request	
Reply	<length><data>	The byte array associated to the input container ID.
	!	<ul style="list-style-type: none"> <li>Invalid argument</li> <li>Not existing string with input-container-ID</li> </ul>
	?	Something else went wrong
Note	<ul style="list-style-type: none"> <li>&lt;input-container-ID&gt; number from 00...09</li> <li>&lt;length&gt; 9 digits as decimal value for the data length</li> <li>&lt;data&gt; content of byte array</li> </ul>	

### 3.4.16 L? Command (returns the current session ID)

Command	L?	
Description	Returns the current session ID	
Type	Request	
Reply	<ID>	
Note	<ID> 3 digits with leading 0	

### 3.4.17 o Command (set logic state of a IO)

Command	o<IO-ID><IO-state>	
Description	Sets the logic state of a specific IO	
Type	Action	
Reply	*	
	!	<ul style="list-style-type: none"> <li>Invalid state (e.g. configuration mode).</li> <li>Wrong ID or IO state. The id and the state must be numbers 0...9.</li> <li>Output is not configured as manual override.</li> </ul>

	?	Invalid command length
Note	<ul style="list-style-type: none"> <li>• <code>&lt;IO-ID&gt;</code> 2 digits for digital output: 01 : IO1 02 : IO2</li> <li>• <code>&lt;IO-state&gt;</code> 1 digit for the state: 0 : logic state low 1 : logic state high</li> </ul>	

### 3.4.18 O? Command (request state of a IO)

Command	O<IO-ID>?	
Description	Requests the state of a specific IO	
Type	Request	
Reply	<IO-ID><IO-state>	
	!	<ul style="list-style-type: none"> <li>• Invalid state (e.g. configuration mode).</li> <li>• Wrong ID or IO state. The ID and the state must be numbers 0...9 .</li> </ul>
	?	Invalid command length
Note	<ul style="list-style-type: none"> <li>• <code>&lt;IO-ID&gt;</code> 2 digits for digital output: 01 : IO1 02 : IO2</li> <li>• <code>&lt;IO-state&gt;</code> 1 digit for the state: 0 : logic state low 1 : logic state high</li> </ul>	The device supports ID 1 and ID 2.

### 3.4.19 p Command (turn process interface output on or off)

Command	p<state>	
Description	Turns the process interface output on or off	
Type	Action	
Reply	*	
	!	<state> contains wrong value
	?	Invalid command length
Note	<p>&lt;state&gt; 1 digit:</p> <p>0 : deactivates all asynchronous output</p> <p>1 : activates asynchronous result output</p> <p>2 : activates asynchronous error output</p> <p>3 : activates asynchronous error and data output</p> <p>4 : activates asynchronous notifications</p> <p>5 : activates asynchronous notifications and asynchronous result</p> <p>6 : activates asynchronous notifications and asynchronous error output</p> <p>7 : activates all outputs</p>	<ul style="list-style-type: none"> <li>• On device restart the value configured within the application is essential for the output of data.</li> <li>• The command can be executed in any device state.</li> <li>• By default, the error codes will not be provided by the device.</li> </ul>

### 3.4.20 s Command (reset current statistics)

Command	s	
Description	Reset current statistics	
Type	action	
Reply	*	Reset of statistics succeeded
	!	No fault scenarios

### 3.4.21 S? Command (request current decoding statistics)

Command	S?	
Description	Requests current decoding statistics	
Type	Request	
Reply	<number of results><t><number of positive decodings><t><number of false decodings>	
	!	No application active
Note	<ul style="list-style-type: none"> <li>• &lt;t&gt; tabulator (0x09)</li> <li>• &lt;number of results&gt; Images taken since application start. 10 digits decimal value with leading 0.</li> <li>• &lt;number of positive decodings&gt; Number of decodings leading to a positive result. 10 digits decimal value with leading 0.</li> <li>• &lt;number of false decodings&gt; Number of decodings leading to a negative result. 10 digits decimal value with leading 0.</li> </ul>	

### 3.4.22 t Command (execute asynchronous trigger)

Command	t	
Description	Executes trigger or burst trigger. The result data is sent asynchronously.	
Type	Action	
Reply	*	Trigger was executed, the device captures an image and evaluates the result.
Note	!	<ul style="list-style-type: none"> <li>• Device is busy with an evaluation.</li> <li>• Device is in an invalid state for the command, e.g. configuration mode.</li> <li>• Device is set to a different trigger source.</li> <li>• No active application.</li> </ul>

### 3.4.23 T? Command (execute synchronous trigger)

Command	T?	
Description	Executes trigger or burst trigger. The result data is sent synchronously.	
Type	Request	
Reply	Process data within the configured layout.	Trigger was executed, the device captures an image, evaluates the result and sends the process data.
Note	!	<ul style="list-style-type: none"> <li>• Device is busy with an evaluation.</li> <li>• Device is in an invalid state for the command, e.g. configuration mode.</li> <li>• Device is set to a different trigger source.</li> <li>• No active application.</li> </ul>

### 3.4.24 v Command (set current protocol version)

Command	v<version>	
Description	Sets the current protocol version. The device configuration is not affected	
Type	Action	

Reply	*	
	!	Invalid version
	?	Invalid command length
Note	<version> 2 digits for the protocol version: 01 / 02 / 03 Available protocol versions are v1, v2 and v3.	



The default protocol version is v3.

### 3.4.25 V? Command (request current protocol version)

Command	V?	
Description	Requests list of current and available protocol versions.	
Type	Request	
Reply	<current version><empty><min version><empty><max version>	
Note	<ul style="list-style-type: none"> <li>&lt;current version&gt; 2 digits for the currently set version</li> <li>&lt;empty&gt; space sign: 0x20</li> <li>&lt;min/max version&gt; 2 digits for the available min and max version that can be set</li> </ul>	

## 3.5 Parameter IDs

The following parameter IDs can be changed via **PCIC**.

Parameter ID	Name	Description	Value range
000003001... 000003999	O2x specific IDs	This ID range is used for O2x specific IDs	
000003001	FocusDistance	Value for FocusDistance to be used [mm]. The value range depends on the device optics.	unsigned int

## 4 EtherNet/IP

### 4.1 Data structures for consuming and producing assemblies

#### Assemblies

Instance	Size [bytes]	Type
100	8...450	Consuming. From device point of view: data buffer for receiving from PLC.
101	16...450	Producing. From device point of view: data buffer for sending to PLC.

#### Consuming assembly data layout

Byte	0...1	2...7	8...449
Description	Command word	Command data	Non mandatory

#### Layout of producing assembly

Byte	0...1	2...3	4...5	6...7	8...15	16...449
Description	Command word for mirroring	Synchronous / asynchronous message identifier	Message counter	Segment counter for data partitioning	Mandatory message data (e.g. error code)	Non mandatory data fields

#### Layout of command word

Bit	0	1...15
Description	Error bit This bit has no meaning in the consuming assembly. It is used for signalling an occurred error to the PLC.	Command bits Each bit represents a specific command.

#### Command bits 0...15 of command word

Bit	Description	Bit	Description
0	Error bit	8	Get statistics
1	N.a.	9	Activate application
2	N.a.	10	Get application list
3	Execute currently configured button function	11	Get IO state
4	Enable / disable gated software trigger	12	Set IO state
5	Acknowledge last received segment	13	Execute synchronous trigger and burst trigger
6	Get last error	14	Activate asynchronous PCIC output
7	Get connection ID	15	Use extended command

#### Synchronous / asynchronous message identifier

Bit	Description
0	Asynchronous message bit (→ <a href="#">Functionality of asynchronous message bit</a> □ 27) For asynchronous messages the bit is set to 1. For synchronous messages the bit is set to 0.

Bit	Description
1...15	Asynchronous message ID (→ <a href="#">Bits for asynchronous message identifier</a> □ 27)

### Data to send exceeds processing assembly data section size

If the size of the data exceeds the size of the configured processing assembly data section size, the data is truncated. No error is risen.

## 4.2 Command handling

This chapter describes the initialization of assembly buffers.



On initialization all buffers are set to 0.

### 4.2.1 Initialization of assembly buffers

On initialization all buffers are set to 0.

### 4.2.2 Default endianness

The default endianness is little-endian.

### 4.2.3 Command execution

A command is executed by setting and holding the specific bit in the command word. This must continue until the device indicates the completion of the command by setting the same bit in the mirroring command word. It is only permitted to set one command bit at a time.

If the device sets the command mirror bit, the command response data is made available in the producing assembly.

If the command execution has failed, the device additionally sets the error bit (bit 0) in the command word for mirroring.

#### Execute a new command

After the command has finished, the PLC can reset the specific command bit. The device resets the mirror bit and the command response data. If the error bit is set, it remains set until the PLC executes the “Get last error” command.

Each time the data in the input frame is changed, the device increments the message counter.

A synchronous trigger command sequence is shown below.



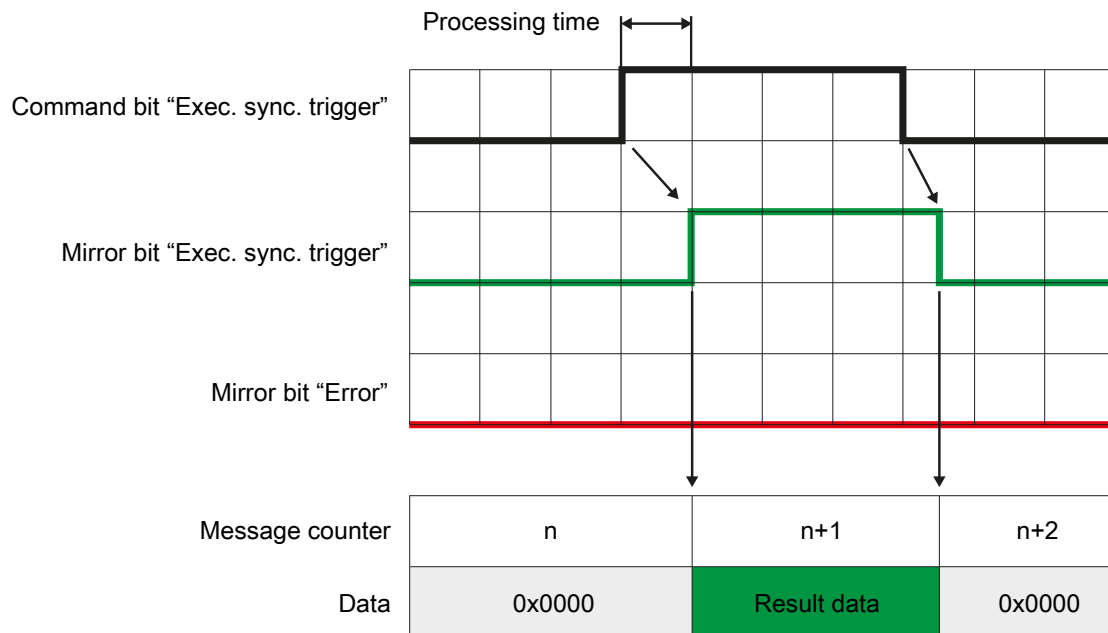
**Example for signal sequence with synchronous trigger**

Fig. 1: Signal sequence with synchronous trigger

### Example of signal sequence with failed trigger

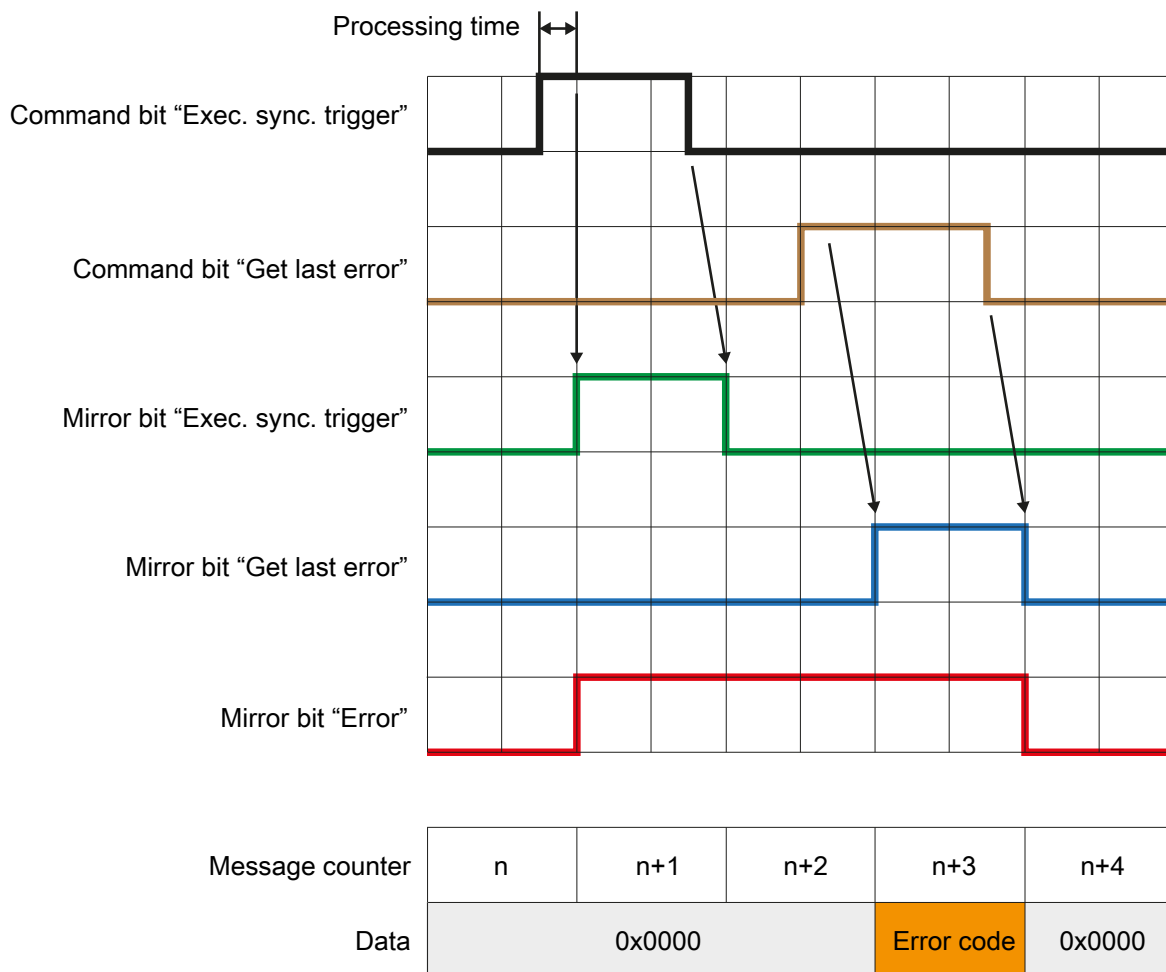


Fig. 2: Signal sequence with failed trigger

#### 4.2.4 Handling of multiple command bits

If more than one command bit is set to 1, an error will be reported.

#### 4.2.5 Blocking of asynchronous messages

If the command handshake procedure has not been finished, no asynchronous message is sent via the EtherNet/IP interface.

#### 4.2.6 Client disconnect

If the client is disconnecting before finishing the handshake procedure, the handshake procedure is cancelled and all buffers are reset.

#### 4.2.7 Sending of response message to a command

If a command has been executed successfully, the producing assembly contains the following information:

Error bit = 0

Command bits = mirror the command within the consuming assembly

Asynchronous message bit = 0

Asynchronous message identifier = 0

Message counter increased by 1

Message data according to the command description ([→ Command description](#) [□ 28](#))



The message data is available in byte 8...n of the producing assembly.

If the command execution has failed, the producing assembly contains the following information:

Error bit = 1

Command bits = mirror the command within the consuming assembly

Asynchronous message bit = 0

Asynchronous message identifier = 0

Message counter increased by 1



No error code is sent in the data section. The error code is polled with the “Get last error” command. [Get last error](#) ([→ □ 28](#))

#### 4.2.8 Reset of error bit

The error bit is reset to 0, if

- the error code caused by a command is retrieved from the client,
- a system error is not present anymore.

[Get last error](#) ([→ □ 28](#))

#### 4.2.9 Cueing of error codes

The EtherNet/IP application can hold the latest occurred system error and command error. The buffered system error and **PCIC** command error can be cleared after they are read by **PLC** with the “Get last error” command. [Get last error](#) ([→ □ 28](#))

#### 4.2.10 Functionality of asynchronous message bit

If the message contains asynchronous data (frame results, system errors, etc.), the asynchronous message bit is set to 1.

#### 4.2.11 Bits for asynchronous message identifier

If the message contains asynchronous data, the identifier represents the asynchronous message type:

0000 : ticket number for asynchronous results.

0001 : ticket number for asynchronous error codes.

0000...0099 : reserved for asynchronous messages sent by the server.

#### 4.2.12 Message counter

For each message sent via the producing assembly, the message counter is increased. The counter starts with the value 1. If the maximum counter is reached, it starts with 1 again.

## 4.3 Command description

### 4.3.1 Get last error

The command retrieves the last command and system error.

The content of the producing assembly is:

#### Bytes 8...11 command error code (32-bit unsigned integer)

Name	Value	Description
EIP_ERROR_NO_ERROR	0	No command error
EIP_ERROR_CMD_UNKNOWN	1	Unknown command
EIP_ERROR_CMD_FAILED	2	Command processing was unsuccessful
EIP_ERROR_CMD_DATA_INVALID	3	Invalid data given for the command
EIP_ERROR_TOO_MANY_CMDS	4	Too many commands executed

#### Bytes 12...15 device error code (32-bit unsigned integer)

Error codes ([→ Error codes](#) [□ 65](#))

#### Example for "Get last error"

Consuming assembly:

Byte 1...0	0x0040	Command word
Byte 3...2	0x0000	
Byte 5...4	0x0000	
Byte 7...6	0x0000	

Producing assembly:

Byte 1...0	0x0040	Command word mirror
Byte 3...2	0x0000	Message identifier
Byte 5...4	0x1234	Message counter
Byte 7...6	0x0000	Segment counter
Byte 11...8	0x0000_0002 *)	Command error code
Byte 15...12	0x05F5_E4E8 **)	Device error code
Byte 16...n	0x0000	

\*) 0x0000\_0002 = 2 = command processing was unsuccessful

\*\*) 0x05F5\_E4E8 = 100001000 = application configuration does not allow process interface trigger

### 4.3.2 Get connection ID

The command gets the connection ID of the current EtherNet/IP connection. The content of the producing assembly is:

Bytes 0...3 : connection ID, 32-bit unsigned integer

### 4.3.3 Get statistics

The command gets the current statistics. The content of the producing assembly is:

- Bytes 8...11 : total readings since application start
- Bytes 12...15 : passed readings
- Bytes 16...19 : failed readings



All values are 32-bit unsigned integers.

#### 4.3.4 Activate application

The command activates the application defined by bytes 6...7 of the consuming assembly. Bytes 2...5 must be set to 0. Otherwise, an error is raised.

The data content of the producing assembly sent after receiving "Activate application" is set to 0.

##### Example for "Activate application"

Consuming assembly:

Byte 1...0	0x0200	Command word
Byte 3...2	0x0000	
Byte 5...4	0x0000	
Byte 7...6	0x0005 *)	Application number

\*) Application 5 = 0x0005

Producing assembly:

Byte 1...0	0x0200	Command word mirror
Byte 3...2	0x0000	Message identifier
Byte 5...4	0x1234	Message counter
Byte 7...6	0x0000	Segment counter
Byte 8...n	0x0000	

#### 4.3.5 Get application list

The command retrieves the current configuration list. The content of the response sent in the producing assembly is:

- Bytes 8...11: total number of saved applications (32-bit unsigned integer)
- Bytes 12...15: number of the active application (32-bit unsigned integer)
- Bytes 16...n: list of application numbers which are configured in the sensor (32-bit unsigned integer)

#### 4.3.6 Aborting application evaluation on application switch

If an application change is initiated, the currently executed evaluation will be aborted. All intermediate results will be omitted, and no result will be added to the service report. The statistics stay unchanged, and no result is signalled via PCIC. This behaviour is valid for all fieldbuses.

#### 4.3.7 Get IO state

Retrieves the logic state of the given IO identifier. Bytes 4...5 of the consuming assembly define the IO identifier as a 16-bit unsigned integer value:

- 1: IO1
- 2: IO2

Bytes 2...3 and 6...7 must be set to 0. Otherwise, an error is raised.

The data content of the processing assembly is:

Bytes 8...11, logic state of the IO, 32-bit unsigned integer:

- 1: high
- 0: low

### Example for "Get IO state"

Consuming assembly:

Byte 1...0	0x0800	Command word
Byte 3...2	0x0000	
Byte 5...4	0x0001 *)	IO identifier
Byte 7...6	0x0000	

\*) IO1 = 0x0001 / IO2 = 0x0002

Producing assembly:

Byte 1...0	0x0800	Command word mirror
Byte 3...2	0x0000	Message identifier
Byte 5...4	0x1234	Message counter
Byte 7...6	0x0000	Segment counter
Byte 11...8	0x0000_0001	read IO state = TRUE
Byte 12...n	0x0000	

## 4.3.8 Set IO state

The command sets the given state of the given IO. Bytes 4...5 of the consuming assembly define the IO ID as a 16-bit unsigned integer value:

- 1: Digital OUT 1
- 2: Digital OUT 2

Bytes 6...7 define the logic state of the IO as 16-bit unsigned integer value.

Bytes 2...3 must be set to 0. Otherwise, an error is raised.

The data content of the producing assembly sent after receiving the command "Set IO state" is set to 0.



The outputs can only be influenced via the process interface if the digital output is connected to the process interface. The connection is established in the logic of the active application.

### Example for Set IO state

Consuming assembly:

Byte 1...0	0x1000	Command word
Byte 3...2	0x0000	
Byte 5...4	0x0001 *)	IO identifier
Byte 7...6	0x0000 **)	ON/OFF state

\*) IO1 = 0x0001 / IO2 = 0x0002

\*\*) ON = 0x0001 / OFF = 0x0000

Producing assembly:

Byte 1...0	0x1000	Command word mirror
Byte 3...2	0x0000	Message identifier
Byte 5...4	0x1234	Message counter
Byte 7...6	0x0000	Segment counter
Byte 8...n	0x0000	

### 4.3.9 Execute button function

The command executes the function the button is currently configured with.

#### 4.3.10 Execute synchronous trigger

The command executes a synchronous trigger. The content of the producing assembly depends on the user-defined process interface output for Ethernet/IP.

##### Example for "Execute synchronous trigger"

Consuming assembly:

Byte 1...0	0x2000	Command word
Byte 3...2	0x0000	
Byte 5...4	0x0000	
Byte 7...6	0x0000	

Producing assembly:

Byte 1...0	0x2000	Command word mirror
Byte 3...2	0x0000	Message identifier
Byte 5...4	0x1234	Message counter
Byte 7...6	0x0000	Segment counter
Byte 9...8	0x0001	Decoding result = Pass
Byte 11...10	0x0005	Number of byte in code content
Byte 12...13	0x4845	Code content "H" and "E"
Byte 14...15	0x4C4C	Code content "L" and "L"
Byte 16...17	0x4F00	Code content "O"
Byte 18...n	0x0000	

#### 4.3.11 Execute gated software trigger

If the device is configured for gated software trigger, the command activates or deactivates the trigger. Bytes 6...7 of the consuming assembly data section define the state as a 16-bit unsigned integer value:

0 : off

1 : on

Bytes 2...5 must be set to 0. Otherwise, an error is raised.

First the command "Execute gated software trigger ON" is finished and the command mirror bit is reset by the device. After resetting the command, the content of the producing assembly data section depends on the user-defined PCIC output for EtherNet/IP. The generated data will be marked as asynchronous data.

Before a new trigger can be issued the master has to perform a state change from 1 to 0.

### 4.3.12 Activate asynchronous data output

The command activates or deactivates the asynchronous data in the producing assembly. Bytes 6...7 of the consuming assembly define the ON/OFF state as a 16-bit unsigned integer value:

- 0 : off
- 1 : on

Bytes 2...5 must be set to 0. Otherwise, an error is raised.

The data content of the producing assembly sent after receiving "Activate asynchronous process interface output" command is set to 0.

#### Example for "Activate asynchronous data output"

Consuming assembly:

Byte 1...0	0x4000	Command word
Byte 3...2	0x0000	
Byte 5...4	0x0000	
Byte 7...6	0x0000 *)	ON/OFF state

\*) ON = 0x0001 / OFF= 0x0000

Producing assembly:

Byte 1...0	0x4000	Command word mirror
Byte 3...2	0x0000	Message identifier
Byte 5...4	0x1234	Message counter
Byte 7...6	0x0000	Segment counter
Byte 8...n	0x0000	

### 4.3.13 Use extended command

The command executes an extended command. ([→ Extended commands ¶ 32](#)) The ID of the extended command is stored as 16-bit integer in bytes 2...3. The remaining data depends on the extended command.

### 4.3.14 Get next data segment

Bit 5 of the command word ([→ Data structures for consuming and producing assemblies ¶ 23](#)) indicates to the device that the PLC is ready for the next data segment. [ID 3: Enable / Disable data partitioning \(→ ¶ 32\)](#)

## 4.4 Extended commands

The command executes an extended command. The ID of the extended command is stored as 16-bit integer in bytes 2...3. The remaining data depends on the extended command. The extended commands and available IDs are described in the next subchapters.

### 4.4.1 ID 3: Enable / Disable data partitioning

#### Enable / Disable data partitioning

The ID of the extended command is 3. Bytes 4...5 contain a 16-bit integer value.

Enable / Disable data partitioning:

- 0 : off



- 1: on

If data partitioning is turned off, bit 5 "Acknowledge last received segment" is ignored.

If data partitioning is turned off though not all segments have been transmitted to the scanner, the device quits the sequence of sending data partitions and waits for new commands or asynchronous messages to be sent.

### Send first data segment

If the device sends data to the scanner, it sets the "Segment counter for data partitioning". It indicates the number of segments whose reception is yet to be confirmed by the scanner.

If the data fits into one segment the value of "Segment counter for data partitioning" is 1 and reception still must be confirmed.

### Acknowledge last data segment

In data segmentation mode each segment sent by the device must be acknowledged by the scanner. This is done by toggling bit 5 "Acknowledge last received segment" of the command word.

If there is more data left, the device sends the data in the next cycle and reduces the segment counter by one. Data is sent by each transition from 0 to 1 or 1 to 0. The device mirrors the state of bit 5.

### Asynchronous message identifier of data segments

When sending asynchronous commands (like frame data) the asynchronous message identifier is set to 1 for the first segment. For each following segment of the same frame, the value is set to 0 by the device.

### Data partitioning disabled

If data partitioning is disabled and the bit 5 "Acknowledge last received segment" gets set from 0 to 1, the event will be ignored by the device. No error is raised.

### Block further asynchronous data

While the sending of partitions is in progress (segment counter for data partitioning != 0) no asynchronous data (new results, errors, ...) will be sent to the scanner. Instead, the device sets the error bit and stores any system or communication error within its internal buffer.

### Error for missed decoded frame

A "Missed new frame" error will be signalled if a new frame was decoded by the device while sending data partitions was in progress.

### Interrupt the sending of segments by sending another command

It is possible to interrupt the sequence of sending further partitions by sending another command to the device. The device stops sending further segments of data, processes the command and then waits for new commands or asynchronous messages to be sent.

### Abort a synchronous command with data partitioning

Some commands demand that the device sends large response data (such as the trigger command). If the PLC wants to abort the sending of multiple data frames, it sets the command word to zero. The device will not react with an error, but stop the sending of further data and mirror the command bit back to the PLC.

## 4.4.2 ID 4: Set the content of a specific match string container

The ID of the extended command is 4. Bytes 4...5 contain a 16-bit unsigned integer value identifying the match string container.

Bytes 6...7 contain a 16-bit unsigned integer defining the size of the string to be set. The content of the string is in the adjacent memory of the assembly.

**PCIC Input string block**

The PCIC Input string size is limited to 256 bytes.

**PCIC Input binary block**

The PCIC Input binary size is limited to 256 bytes.

**Binary string**

The purpose of a binary string is to allow the user to define binary data as a string containing hexadecimal characters.

- A valid binary string must contain an even number of characters from 0...9, a...f and A...F.
- For human readability the following separator characters are allowed: white space, comma, colon: dash-semicolon; Those characters are ignored during parsing.
- Any other character will result in an invalid binary string.
- White space characters are all characters identified by <https://doc.qt.io/archives/qt-4.8/qchar.html#isSpace>, which includes the following ascii characters: \t, \r, \n, \v, \f.

**4.4.3 ID 5: Get the content of a specific match string container**

The ID of the extended command is 5. Bytes 4...5 contain a 16-bit unsigned integer value identifying the match string container.

The result is placed in bytes 8...n of the producing assembly. Bytes 8...9 contain an unsigned 16-bit integer identifying the length of the match string. Bytes 10...n contain the match string itself.

**4.4.4 ID 6: Toggle the viewindicator on/off**

The ID of the extended command is 6. Bytes 4...5 contain a 16-bit unsigned integer value to toggle the viewindicator:

- 0: off
- 1: on

Bytes 6...7 contain a 16-bit unsigned integer defining how long the viewindicator is turned on. If the value is 0, the viewindicator is permanently turned on. The maximum value is 600 seconds. The value will be ignored if the viewindicator is turned off or the device is not in run or simulation mode.

**4.4.5 ID 7/8: Set/Get the current focus distance****Set the value of the temporary focus distance**

The ID of the extended command is 7. Bytes 4...5 contain a 16-bit unsigned integer value for the focus distance in millimetres. The value range depends on the optics of the device. An error will be returned if the value does not suit the value range or the device is not in run or simulation mode. The value is not permanently stored on the device.

**Get the value of the temporary focus distance**

The ID of the extended command is 8. The result is placed in bytes 8...9 of the producing assembly, containing a 16-bit unsigned integer value for the current focus distance in millimetres. An error will be returned if the device is not in run or simulation mode.

**4.4.6 ID 9: Reset current statistics**

The ID of the extended command for "Reset current statistics" is 9. There is no payload data necessary.

## 5 PROFINET

### PROFINET IO device structure

A PROFINET IO device consists of a number of slots which are further divided into subslots. Slot 0 is a special purpose slot representing the device. Slots 1...0x7FFF could be used by application processes. From a logical point of view, there are one or more application processes on an IO device which are implemented by sets of modules. Modules are decomposed into submodules. Modules are assigned (plugged into) slots, and submodules are assigned (plugged into) subslots.

In case of the O2x5xx device implementing the PCIC protocol, 14 slots are used:

- slot 0 for DAP,
- slot 1 for feedback input data,
- slot 2 for PCIC input data and command response,
- slots 3...12 for logic layer input data,
- slot 13 for output data.

Modules and submodules providing input data sent by the IO device to the controller are plugged into the corresponding number of slots / subslots. Modules and submodules handling output data sent by the controller to IO device are plugged into slot / subslot 13.

Beside the structure of IO cyclic data, the PROFINET IO device defines a set of record data which could be read or written by the controller using acyclic services. Beside mandatory record data objects defined by the standard (e.g. I&M0, ARData, etc.), a user can define application-specific data objects.

### Protocol for PROFINET application

PROFINET enables data exchange between the controller and the IO device by means of 3 different communication channels:

- input or output cyclic data,
- asynchronous alarms,
- record data object with read or write requests.

PCIC protocol commands and responses are using cyclic data frames:

- Cyclic input frames are sent from the IO device to the controller.
- Cyclic output frames are the ones sent by the controller to the device.

The frequency of frames sent on both sides is determined during connection establishment and the allowed values are powers of 2 (1, 2, 4, 8, 16, etc.). The possible cyclic data delivery intervals for the device are: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512.

PROFINET applications are configured and tested with 8 ms cycle time. Every cycle controller sends one data frame (output) to the device. The IO device sends one data frame (input) to the controller. The input and output cyclic frames carry the PCIC protocol requests which are described in detail in the subsequent sections.

## 5.1 Stack requirements

### Vendor ID parameter

The vendor ID parameter in the GSDML file is set to value 0136h. The same value is returned by the device in response to the I&M0 record data.

### Device ID parameter for O2I family

The device ID parameter in the GSDML file is set to value 0x02D5 (725d).

### Device ID parameter for O2D family

The device ID parameter in the GSDML file is set to value 0x02D6 (726d).

## Providing station name

The PROFINET stack provides the storage for the station name and the interface for changing the value of this parameter. There is an interface to set both a permanent and temporary value of the station name. Both functions always store the parameter permanently.

## Reading the station name

The PROFINET stack provides the possibility to read the current value of the station name by means of the **DCP** protocol.

## Setting the station name

The PROFINET stack stores the value of the station name by means of the DCP protocol permanently for the permanent and temporary interface.

## Reading device IP

The PROFINET stack provides the possibility to read the value of the device IP address by means of the DCP protocol.

## Writing device IP

The PROFINET stack provides the possibility to write the value of the device IP address by means of the DCP protocol.

## Station name interface for ifm Vision Assistant

The PROFINET stack provides the interface to read or write the station name via the ifm Vision Assistant.

## Encoding of station name

The default value is an empty string. It indicates that no NameOfStation is assigned. The field is coded as data type OctetString with 1...240 octets. The following syntax applies:

- 1 or more labels, separated by [.]
- The total length is 1...240
- The label length is 1...63
- Labels consist of [a...z, 0...9]
- Labels do not start with [-]
- Labels do not end with [-]
- The first label does not have the form "port-xyz" or "port-xyz-abcde" with a...z = 0...9, to avoid wrong similarity with the field AliasNameValue
- Station names do not have the form a.b.c.d with a, b, c, d = 0...999



Labels only start with xn-- if the original string contains characters other than a...z, 0...9.

- ▷ Example 1: device-1.machine-1.plant-1.vendor
- ▷ Example 2: mühle1.ölmühle1.plant.com is coded as  
xn--mhle1 -kva.xn--lmhle1 -vxa4c.plant.com



The field NameOfStationValue is not terminated by zero.



Devices with more than one Ethernet interface need more than one unique NameOfStation. In this case, the station name is used as interface name.

Historically, most devices have only one interface, thus the interface name was used as station name.

## Access to device name

The device name is accessible and changeable via the GUI.

### Device name behaviour at sensor cloning

The device name for **PNIO** is part of the sensor cloning functionality.

### Setting device name by GUI

The GUI is only able to set a non-volatile device name.

### Setting IP address by GUI

The GUI cannot change the IP address if the PNIO stack is activated.

### Setting volatile IP by PLC

If a **PLC** sets a volatile IP address, the PNIO stack signals to set a temporal IP address to the main software system. If the device has not rebooted, the PNIO stack signals the IP to the PLC / PNIO configuration tool. After rebooting, the device has the last non-volatile IP address and the PNIO stack signals **0.0.0.0** to the PLC / PNIO configuration tool.

### Setting non-volatile IP by PLC

If a **PLC** sets a non-volatile IP address, the PNIO stack signals to set a static IP address to the main software system. If the device has not rebooted, the PNIO stack signals this IP to the PLC / PNIO configuration tool. After rebooting, the IP is used and signalled to the PLC / PNIO configuration tool.

### LLDP support

LLDP support is provided by the PNIO application.

### SNMP support

SNMP support is provided by the PNIO application.

### Conformance type CC-B

The PNIO application must be approved for CC-B conformance class.

### Netload class 2

The PNIO application must be approved for netload class 2.

### User defined IP address

The user can set his own IP address via XML-RPC if PNIO is not activated.

## 5.2 Command structure

### 5.2.1 Available slots

The PROFINET interface consist of the following slots. Their direction and size are defined within the table.

Slot	Input / Output	Size [bytes]	Description
1	Input	8	Process Status
2	Input	18...254	Data or command response
3	Input	18...254	Logic Data Container 0
4	Input	18...254	Logic Data Container 1
5	Input	18...254	Logic Data Container 2
6	Input	18...254	Logic Data Container 3
7	Input	18...254	Logic Data Container 4
8	Input	18...254	Logic Data Container 5

Slot	Input / Output	Size [bytes]	Description
9	Input	18...254	Logic Data Container 6
10	Input	18...254	Logic Data Container 7
11	Input	18...254	Logic Data Container 8
12	Input	18...254	Logic Data Container 9
13	Output	8...136	Command and Data

The **PLC** input and output data image size depends on the selection of the user settings. Each slot and module can be set individually. Slot 1, 2 and 13 are mandatory and activated by the system. Slot 1 is configured completely and cannot be changed. The user needs to insert the submodule of his preferred size into the subslots 2.1 and 13.1.

If the total number of bytes of all selected modules exceed the limits of the multicode reader, the configuration is rejected and no data exchange with the PLC is possible.

The maximum available input or output data size is 1024 bytes.

## 5.2.2 Verifying the size of selected slots on connection setup

The **GSDML** fields MaxInputLength and MaxOutputLength must be set to the maximum allowed PROFINET frame of 1024 bytes. This protects the **PLC** from misconfigurations regarding the frame size. The corresponding PLC management program does not allow larger frames than MaxInputLength/MaxOutputLength to be specified.

On connection setup, the device verifies the sizes of the selected slots. If they exceed the maximum of an allowed PROFINET frame for IO data of 1024 bytes, an error will be returned.

The device sends a corresponding message to the controller in a corresponding **DCP** frame.

## 5.2.3 Module size for slot 2

Slot 2 is a mandatory slot. The following slot sizes are available:

18, 34, 66, 130, 254 bytes.



The input module of this slot is a submodule and must be inserted into subslot 2.1.

## 5.2.4 Module sizes for slots 3...12

Slots 3...12 are not mandatory and can be deselected. This must be considered in the **GSDML** file. The following slot sizes are available:

18, 34, 66, 130, 254 bytes.



The input modules of these slots are submodules and must be inserted into subslot 3.1, 4.1 etc.

## 5.2.5 Module size for slot 13

Slot 13 is a mandatory slot. The following slot sizes are available:

8, 16, 24, 40, 72, 136 bytes.



The output module of this slot is a submodule and must be inserted into subslot 13.1.

## 5.2.6 Truncation of data

If the data content exceeds the implemented submodule size, it will be truncated. The only exception is if data segmentation for slot 2 has been activated.

### 5.2.7 Data layout of slot 1 - process status

The size of the process status channel is fixed to 8 bytes. It is not possible to change or delete it from the slot mapping.

The layout is:

Bytes	Description
0...1	Command word for mirroring (→ <a href="#">Data layout of command word</a> ¶ 40)
2...3	Synchronous / asynchronous message identifier (→ <a href="#">Identifier for synchronous and asynchronous messages</a> ¶ 40)
4...5	Message counter (→ <a href="#">Message counter</a> ¶ 43)
6...7	Segment counter for data segmentation

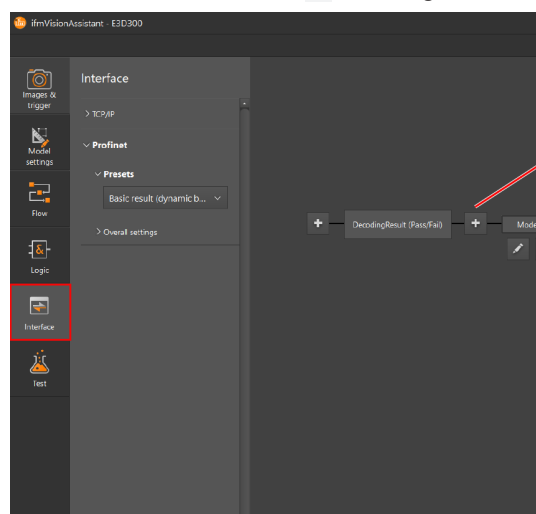
### 5.2.8 Data layout of slots 2...12

The slots 2...12 have the following data layout:

Bytes	Description
0...1	Message counter
2...254	Data

As this data must be consistent with the process status, the message counter of these slots must be synchronous with the process status. This is important in cases where the process status contains new synchronous or asynchronous frame data.

The data content for slot 2 is configured with the software ifm Vision Assistant:



Module	Rack	Slot	I...	Type
▼ O2I5xx-PN	0	0		O2I51x
▶ Interface	0	0 X1		O2I5xx-PN
Process Status_1	0	1		Process Status
▼ Data or Command Response_1	0	2		Data or Command Response
Input 066 Byte	0	2 1		Input 066 Byte
▼ Logic Data Container 0_1	0	3		Logic Data Container 0
Input 034 Byte	0	3 1		Input 034 Byte
	0	4		
	0	5		
▼ Logic Data Container 3_1	0	6		Logic Data Container 3
Input 018 Byte	0	6 1		Input 018 Byte
	0	7		
	0	8		
	0	9		
	0	10		
	0	11		
	0	12		
▼ Command and Data_1	0	13		Command and Data
Output 008 Byte	0	13 1		Output 008 Byte

Fig. 3: Data content for slot 2

The data content for slots 3...12 is configured with the software ifm Vision Assistant:

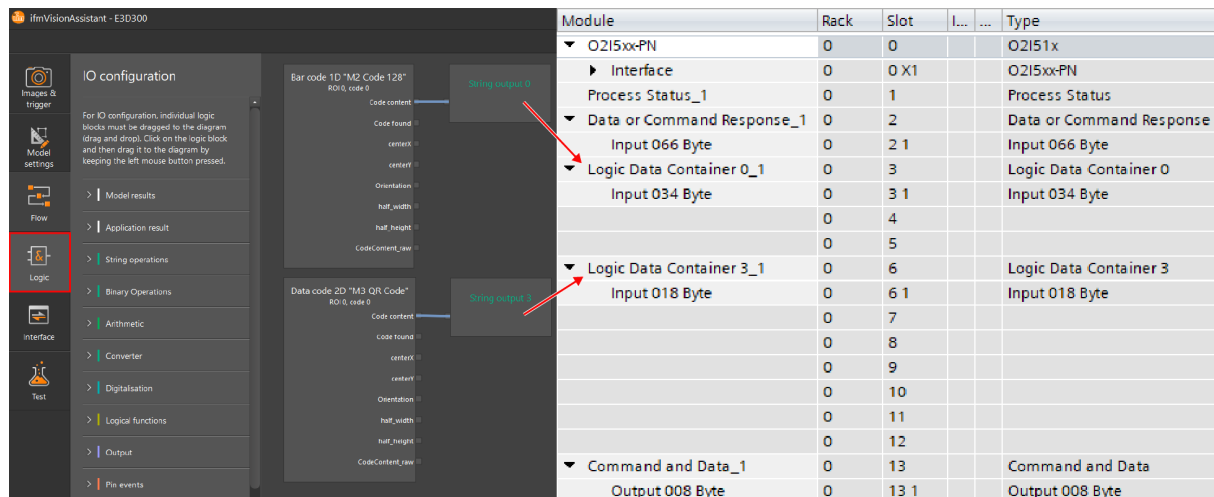


Fig. 4: Data content for slots 3...12

The string outputs are fix-linked to the corresponding Logic Data Containers. Example:

String output 3 (Binary output 3) is fix linked with Logic Data Container 3.

- Download the software ifm Vision Assistant via the download area of the unit:  
[documentation.ifm.com](https://documentation.ifm.com)

## 5.2.9 Data layout of slot 13

Slot 13 is the channel for the command word and command data. It has the following data layout:

Bytes	Description
0...1	Command word
2...7	Command data
8...135	Extended command data

## 5.2.10 Data layout of command word

The command word has the following data layout:

Bit	Description	Bit	Description
0	Error bit	8	Get statistics
1	N.a.	9	Activate application
2	N.a.	10	Get application list
3	Execute currently configured button function	11	Get IO state
4	Enable / disable gated software trigger	12	Set IO state
5	Acknowledge last received segment	13	Execute synchronous trigger and burst trigger
6	Get last error	14	Activate asynchronous PCIC output
7	Get connection ID	15	Use extended command

## 5.2.11 Identifier for synchronous and asynchronous messages

The synchronous and asynchronous identifiers have the following data layout:



Bit	Description
0	Asynchronous message bit (→ <a href="#">Functionality of asynchronous message bit</a> □ 43) For asynchronous messages the bit is set to 1. For synchronous messages the bit is set to 0.
1...15	Asynchronous message ID (→ <a href="#">Bits for asynchronous message identifier</a> □ 43)

## 5.3 Command handling

The section describes handling of the **PCIC** commands sent by the controller. The **PLC** sends the commands to the device in the output frames by setting the appropriate bit in the command word. The current value of the command word and command data can be obtained from the output module by the application using the API provided by the stack.

After detecting that one of the command bits changed the state from 0 to 1, the PROFINET application will execute the corresponding command and set the response in the input frames.

The handling of the commands is done by the API, which provide the functionality and the information required to prepare the response. The response is written to the buffer associated with the input module and will be sent by the stack to the controller during next cycle.

### 5.3.1 Number of supported PROFINET connections

The device supports the connection with 1 controller.

### 5.3.2 Initialization of input and output buffers

After the connection has been established, the input and output buffers are initialized with 0.

### 5.3.3 Default endianness

The default endianness is big-endian.

### 5.3.4 Command execution

A command is executed by setting and holding the specific bit in the command word. This must continue until the device indicates the completion of the command by setting the same bit in the mirroring command word. It is only permitted to set one command bit at a time.

If the device sets the command mirror bit, the command response data is made available in the "Data or command response" module.

If the command execution has failed, the device additionally sets the error bit (bit 0) in the command word for mirroring.

#### Execute a new command

After the command has finished, the **PLC** can reset the specific command bit. The device resets the mirror bit and the command response data. If the error bit is set, it remains set until the PLC executes the "Get last error" command.

Each time the data in the input frame is changed, the device increments the message counter.

A synchronous trigger command sequence is shown below.

### Example for signal sequence with synchronous trigger

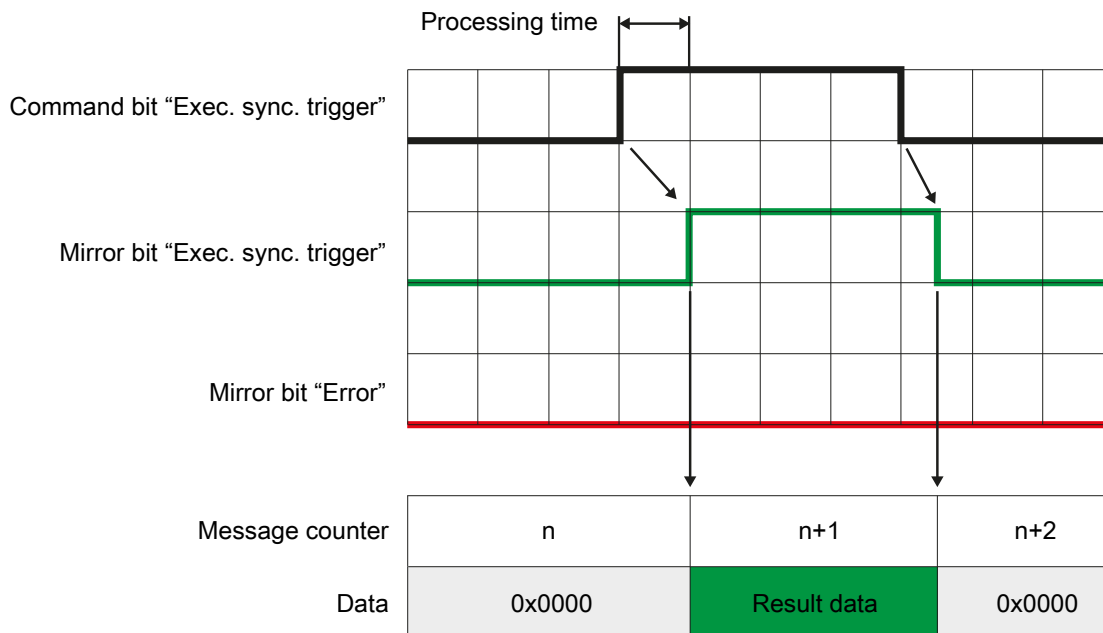


Fig. 5: Example for signal sequence with synchronous trigger

### 5.3.5 Handling of multiple command bits

If more than one command bit is set to **1**, an error will be reported.

### 5.3.6 Blocking of asynchronous messages

If the command handshake procedure has not been finished, no asynchronous message is sent from the sensor.

### 5.3.7 Handling of disconnections during command execution

If the client is disconnecting before finishing the handshake procedure, the handshake procedure is cancelled and all buffers are reset.

### 5.3.8 Sending of response message to a command

If a command has been executed successfully, the input frame contains the following information:

Error bit = **0**

Command bits = mirror the command within the output frame

Asynchronous message bit = **0**

Asynchronous message identifier = **0**

Message counter increased by **1**

Message data according to the command description (→ [Command description](#) 43)



The message data is available in slot 2 "Data or Command Response".

If the command execution has failed, the input frame contains the following information:

Error bit = 1

Command bits = mirror the command within the output frame

Asynchronous message bit = 0

Asynchronous message identifier = 0

Message counter increased by 1

Message data set to 0x00



No error code is sent in the data section. The error code is polled with the "Get last error" command. [Get last error](#) (→ [44](#))

### 5.3.9 Reset of error bit

The error bit is reset to 0, if

- the error code caused by a command is retrieved from the client,
- a system error is not present anymore.

[Get last error](#) (→ [44](#))

### 5.3.10 Queuing of error codes

The PROFINET application can buffer

- the last 1 system error and
- the last 1 command error.

The buffered system error and process interface command error will be cleared after they are read by the PLC with the "get last error" command. [Get last error](#) (→ [44](#))

### 5.3.11 Functionality of asynchronous message bit

If the message contains asynchronous data (frame results, system errors, etc.), the asynchronous message bit is set to 1.

### 5.3.12 Bits for asynchronous message identifier

If the message contains asynchronous data, the identifier represents the asynchronous message type according to the following tickets:

- 0000: asynchronous results.
- 0001: asynchronous error codes.
- 0000...0099: asynchronous messages sent by the server.

### 5.3.13 Message counter

For each new data sent in the input frame, the message counter is incremented. The counter starts with 1. If the maximum counter value is reached, it starts again at 1.

## 5.4 Command description

### 5.4.1 Execute gated software trigger

If the device is configured for gated software trigger, the command activates or deactivates the gated trigger.

The bytes 6...7 of slot 13 define the on/off state as a 16-bit unsigned integer value:

- 0: off
- 1: on

The bytes 2...5 must be set to 0.

First the command "Execute gated software trigger ON" is finished and the command mirror bit is reset by the device. The content of the incoming data section depends on the user-defined process interface output for PROFINET. The generated data will be marked as asynchronous data. ([→ Functionality of asynchronous message bit □ 43](#))

#### Example for "execute gated software trigger ON"

Command data byte 0...1	0x0010	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0001 *)	ON/OFF

\*) ON = "0x0001" / OFF = "0x0000"

Command response byte 0...1	0x1234	Message counter
Command response byte 2...n	0x00	

### 5.4.2 Get last error

The command retrieves the last command and system error. The contents in logic data container 0...9 are not changed.

The content of the data sent in the "Data or command response" module:

#### Bytes 2...5 command error code (32-bit unsigned integer)

Name	Value	Description
EIP_ERROR_NO_ERROR	0	No command error
EIP_ERROR_CMD_UNKNOWN	1	Unknown command
EIP_ERROR_CMD_FAILED	2	Command processing was unsuccessful
EIP_ERROR_CMD_DATA_INVALID	3	Invalid data given for the command
EIP_ERROR_TOO_MANY_CMDS	4	Too many commands executed

#### Bytes 6...9 device error code (32-bit unsigned integer)

Error codes ([→ Error codes □ 65](#))

#### Example for "Get last error"

Command data byte 0...1	0x0040	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0000	

Command response byte 0...1	0x1234	Message counter
Command response byte 2...5	0x0000_0002 *)	Command error code
Command response byte 6...9	0x05F5_E4E8 **)	Device error code
Command response byte 10...n	0x0000	

\*) 0x0000\_0002 = 2 = Command processing was unsuccessful

\*\*) 0x05F5\_E4E = 100001000 = Application configuration does not allow process interface trigger

### 5.4.3 Get connection ID

The command retrieves the connection ID of the current PROFINET connection. The response sent in the "Data or command response" module contains 16 bytes of **AR UUID**. The contents in logic data container 0...9 are not changed.

#### Example for "Get connection ID"

Command data byte 0...1	0x0080	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0000	

Command response byte 0...1	0x1234	Message counter
Command response byte 2...5	0x0000_0001	Connection ID
Command response byte 6...n	0x00	

### 5.4.4 Get statistics

The command gets the current statistics. The content of the response sent in the "Data or command response" module is:

- Bytes 2...5 : total readings since application start
- Bytes 6...9 : passed readings
- Bytes 10...13 : failed readings



All values are 32-bit unsigned integers.

### 5.4.5 Activate application

The command activates the application defined by bytes 6...7 of the output frame data section. Bytes 2...5 must be set to 0. Otherwise, an error is raised.

The data content of the "Data or command response" module sent after receiving "Activate application" is set to 0. The contents in the logic data container 0...9 are not changed.

#### Example for "Activate application"

Command data byte 0...1	0x0200	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0005 *)	Application number

\*) Application 5 = 0x0005

Command response byte 0...1	0x1234	Message counter
Command response byte 2...n	0x0000	

### 5.4.6 Get application list

The command retrieves the current configuration list. The content of the response sent in the "Data or command response" module is:

- Byte 2...5 : total number of saved applications (32-bit unsigned integer)
- Bytes 6...9 : number of the active application (32-bit unsigned integer)
- Bytes 10...n : list of application numbers which are configured in the sensor (32-bit unsigned integer)

#### Example for "Get application list"

Command data byte 0...1	0x0400	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0000	

Command response byte 0...1	0x1234	Message counter
Command response byte 2...5	0x0000_0007	Total number of saved applications
Command response byte 6...9	0x0000_0005	Number of the active application
Command response byte 10...13	0x0000_0001	Application no. 1 exists
Command response byte 14...17	0x0000_0002	Application no. 2 exists
Command response byte 18...21	0x0000_0005	Application no. 5 exists
Command response byte 22...25	0x0000_0006	Application no. 6 exists
Command response byte 26...29	0x0000_0007	Application no. 7 exists
Command response byte 30...33	0x0000_0008	Application no. 8 exists
Command response byte 34...37	0x0000_000A	Application no. 10 exists
Command response byte 38...n	0x0000	

### 5.4.7 Get IO state

Retrieves the logic state of the given IO identifier. Bytes 4...5 of the output frame data section define the IO identifier as a 16-bit unsigned integer value:

- 1 : IO1
- 2 : IO2

Bytes 2...3 and 6...7 must be set to 0. Otherwise, an error is raised.

The data sent in the "Data or command response" module is:

Bytes 2...5, logic state of the IO, 32-bit unsigned integer:

- 1 : high
- 0 : low

#### Example for "Get IO state"

Command data byte 0...1	0x0800	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0001 *)	IO identifier
Command data byte 6...7	0x0000	

\*) IO1 = 0x0001 / IO2 = 0x0002

Command response byte 0...1	0x1234	Message counter
Command response byte 2...5	0x0000_0001	read IO state = TRUE
Command response byte 6...n	0x00	

### 5.4.8 Set IO state

The command sets the given state of the given IO. Bytes 4...5 of the output frame data section define the IO ID as a 16-bit unsigned integer value:

- 1: Digital OUT 1
- 2: Digital OUT 2

Bytes 6...7 define the logic state of the IO as 16-bit unsigned integer value.

Bytes 2...3 must be set to 0. Otherwise, an error is raised.

The data content of the "Data or command response" module sent after receiving the command "Set IO state" is set to 0. The contents in logic data container 0...9 are not changed.



The outputs can only be influenced via the process interface if the digital output is connected to the process interface. The connection is established in the logic of the active application.

#### Example for Set IO state

Command data byte 0...1	0x1000	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0001 *)	IO identifier
Command data byte 6...7	0x0000 **)	ON/OFF state

\*) IO1 = 0x0001 / IO2 = 0x0002

\*\*\*) ON = 0x0001 / OFF = 0x0000

Command response byte 0...1	0x1234	Message counter
Command response byte 2...n	0x00	

### 5.4.9 Execute button function

The command executes the function the button is currently configured with.

#### 5.4.10 Execute synchronous trigger

The command executes a synchronous trigger. The content of the "Data or command response" module depends on the user defined process interface output for PROFINET.

#### Example for "Execute synchronous trigger"

Command data byte 0...1	0x2000	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0000	

Command response byte 0...1	0x1234	Message counter
Command response byte 2...3	0x0001	Decoding result = Pass
Command response byte 4...5	0x0005	Number of byte in code content

Command response byte 6...7	0x4845	Code content "H" and "E"
Command response byte 8...9	0x4C4C	Code content "L" and "L"
Command response byte 10...11	0x4F00	Code content "O"
Command response byte 12...n	0x00	

### 5.4.11 Activate asynchronous data output

The command activates or deactivates the asynchronous data exchange of the slots 2...12. Bytes 6...7 of the output frame data section define the ON/OFF state as a 16-bit unsigned integer value:

- 0 : off
- 1 : on

Bytes 2...5 must be set to 0. Otherwise, an error is raised.

The data content of the "Data or command response" module sent after receiving "Activate asynchronous process interface output" command is set to 0.

#### Example for "Activate asynchronous data output"

Command data byte 0...1	0x4000	Command word
Command data byte 2...3	0x0000	
Command data byte 4...5	0x0000	
Command data byte 6...7	0x0000 *)	ON/OFF state

\*) ON = 0x0001 / OFF= 0x0000

Command response byte 0...1	0x1234	Message counter
Command response byte 2...n	0x00	

### 5.4.12 Use extended command

The command executes an extended command. (→ [Extended commands](#) 48) The ID of the extended command is stored as 16-bit integer in bytes 2...3. The remaining data depends on the extended command.

### 5.4.13 Get next data segment

Bit 5 of the command word (→ [Data layout of command word](#) 40) indicates to the device that the PLC is ready for the next data segment. [ID 3: Enable / Disable data partitioning](#) (→ 48)

## 5.5 Extended commands

The command executes an extended command. The ID of the extended command is stored as 16-bit integer in bytes 2...3. The remaining data depends on the extended command. The extended commands and available IDs are described in the next subchapters.

### 5.5.1 ID 3: Enable / Disable data partitioning

#### Data partitioning

Data partitioning is only available for slot 2 "Data or command response" of the input data.

#### Enable / Disable data partitioning

The ID of the extended command is 3. Bytes 4...5 contain a 16-bit integer value.



Enable / Disable data partitioning:

- 0: off
- 1: on

If data partitioning is turned off, bit 5 "Acknowledge last received segment" is ignored.

If data partitioning is turned off though not all segments have been transmitted to the controller, the device quits the sequence of sending data partitions and waits for new commands or asynchronous messages to be sent.

### Send first data segment

If the device sends data to the controller, it sets the "Segment counter for data partitioning". It indicates the number of segments whose reception is yet to be confirmed by the controller.

If the data fits into one segment, the value of "Segment counter for data partitioning" is 1 and reception still must be confirmed.

### Acknowledge last data segment

In data segmentation mode, each segment sent by the device must be acknowledged by the controller. This is done by toggling bit 5 "Acknowledge last received segment" of the command word.

If there is more data left, the device sends the data in the next cycle and reduces the segment counter by one. Data is sent by each transition from 0 to 1 or 1 to 0. The device mirrors the state of bit 5.

### Asynchronous message identifier of data segments

When sending asynchronous commands (like frame data), the asynchronous message identifier is set to 1 for the first segment. For each following segment of the same frame, the value is set to 0 by the device.

### Data partitioning disabled

If data partitioning is disabled and the bit 5 "Acknowledge last received segment" gets set from 0 to 1, the event will be ignored by the device. No error is raised.

### Block further asynchronous data

While the sending of partitions is in progress (segment counter for data partitioning != 0), no asynchronous data (new results, errors, ...) will be sent to the controller. Instead, the device sets the error bit and stores any system or communication error within its internal buffer.

### Error for missed decoded frame

A "Missed new frame" error will be signalled if a new frame was decoded by the device while sending data partitions was in progress.

### Interrupt the sending of segments by sending another command

It is possible to interrupt the sequence of sending further partitions by sending another command to the device. The device stops sending further segments of data, processes the command and then waits for new commands or asynchronous messages to be sent.

### Abort a synchronous command with data partitioning

Some commands demand that the device sends large response data (such as the trigger command). If the PLC wants to abort the sending of multiple data frames, it sets the command word to zero. The device will not react with an error, but stop the sending of further data and mirror the command bit back to the PLC.

### Example

Command data byte 0...1	0x8000	Command word
Command data byte 2...3	0x0003 *)	Command ID
Command data byte 4...5	0x0001 **)	ON/OFF state

Command data byte 6...7	0x0000	
-------------------------	--------	--

\*) 0x0003 = data segmentation

\*\*) ON = 0x0001 / OFF= 0x0000

Command response byte 0...1	0x1234	Message counter
Command response byte 2...n	0x00	

### 5.5.2 ID 4: Set the content of a specific match string container

The ID of the extended command is 4. Bytes 4...5 contain a 16-bit unsigned integer value identifying the match string container.

Bytes 6...7 contain a 16-bit unsigned integer defining the size of the string to be set. The content of the string is in the adjacent memory of the command and data module.

#### Binary string

The purpose of a binary string is to allow the user to define binary data as a string containing hexadecimal characters.

- A valid binary string must contain an even number of characters from 0...9, a...f and A...F.
- For human readability the following separator characters are allowed: white space, comma, colon: dash-semicolon; Those characters are ignored during parsing.
- Any other character will result in an invalid binary string.
- White space characters are all characters identified by <https://doc.qt.io/archives/qt-4.8/qchar.html#isSpace>, which includes the following ascii characters: \t, \r, \n, \v, \f.

### 5.5.3 ID 5: Get the content of a specific match string container

The ID of the extended command is 5. Bytes 4...5 contain a 16-bit unsigned integer value identifying the match string container (0...9).

The result is stored in the "Data or command response" module. Bytes 2...3 contain an unsigned 16-bit integer identifying the length of the match string. Bytes 4...n contain the string itself. The contents in logic data container 0...9 are not changed.

### 5.5.4 ID 6: Toggle the viewindicator on/off

The ID of the extended command is 6. Bytes 4...5 contain a 16-bit unsigned integer value to toggle the viewindicator:

- 0 : off
- 1 : on

Bytes 6...7 contain a 16-bit unsigned integer defining how long the viewindicator is turned on. If the value is 0, the viewindicator is permanently turned on. The maximum value is 600 seconds. The value will be ignored if the viewindicator is turned off or the device is not in run or simulation mode.

### 5.5.5 ID 7/8: Set/Get the current focus distance

#### Set the value of the temporary focus distance

The ID of the extended command is 7. Bytes 4...5 contain a 16-bit unsigned integer value for the focus distance in millimetres. The value range depends on the optics of the device. An error will be returned if the value does not suit the value range or the device is not in run or simulation mode. The value is not permanently stored on the device.

### Get the value of the temporary focus distance

The ID of the extended command is 8. The result is placed in bytes 2...3 in the “Data or command response” module, containing a 16-bit unsigned integer value for the current focus distance in millimetres. An error will be returned if the device is not in run or simulation mode.

### 5.5.6 ID 9: Reset current statistics

The ID of the extended command for “Reset current statistics” is 9. There is no payload data necessary.

## 5.6 GSDML file

### Name of GSDML file

The name of a GSDML file is composed in the following order:

1. GSDML
2. The version ID in format Vx.y whereby x and y are unsigned numbers. The version ID refers to the ID of the GSDML schema used.
3. Vendor name ifm
4. Device name (= product code) or device family name (e.g., O2I5xx).
5. Release date of the GSD in format yyyyymmdd (e.g., 20230823).
6. File extension .xml.

Example for O2I5xx devices:

```
GSDML-V2.4-ifm-O2I5xx-20230823.xml
```

Example for O2D5xx devices:

```
GSDML-V2.4-ifm-O2D5xx-20230823.xml
```

### Parameter “Device identity” for O2I5xx

GSDML parameter	Value
VendorID	0x0136
VendorName	ifm electronic
infoText	O2I5xx - 1D/2D code reader
DeviceID	0x02D5 (725d)

### Parameter “Device identity” for O2D5xx

GSDML parameter	Value
VendorID	0x0136
VendorName	ifm electronic
infoText	O2D5xx - object inspection and detection
DeviceID	0x02D6 (726d)

**Parameter “Device function” for O2I5xx**

GSDML parameter	Value
ProductFamily	MCR
MainFamily	Ident System

**Parameter “Device function” for O2D5xx**

GSDML parameter	Value
ProductFamily	Optical recognition
MainFamily	Sensors

**Parameter “Device access point item” for O2I5xx**

GSDML parameter	Value
ID	DAP 100
ModuleIdentNumber	0x00000100
PNIO version	V2.41 or latest one
DNS_CompatibleName	O2I5xx-PN
ObjectUUID_LocalIndex	1
NameOfStationNotTransferable	false

**Parameter “Device access point item” for O2D5xx**

GSDML parameter	Value
ID	DAP 100
ModuleIdentNumber	0x00000100
PNIO version	V2.41 or latest one
DNS_CompatibleName	O2D5xx-PN
ObjectUUID_LocalIndex	1
NameOfStationNotTransferable	false

**Parameter “Module Info in Device Access Point Item” for O2I5xx**

GSDML parameter	Value
CategoryRef	not used
Name / TextId	O2I5xx
InfoText / TextId	ifm 1D/2D code reader
OrderNumber	O2I5xx
HardwareRelease	not used
SoftwareRelease	not used

**Parameter “Module Info in Device Access Point Item” for O2D5xx**

GSDML parameter	Value
CategoryRef	not used
Name / TextId	O2D5xx
InfoText / TextId	ifm O2D5xx - object inspection and detection
OrderNumber	O2D5xx
HardwareRelease	not used
SoftwareRelease	not used

### Graphic item in “Device Access Point Item”

The name of a graphic item must follow the following structure:

```
GSDML-0136-0500-ifm-O2I5xx.bmp
```

### Device tree in TIA portal for O2I5xx

According to the GSDML parameters described above, the following device tree appears in the Siemens TIA Portal:

Other fieldbus devices	Given by TIA portal
PROFINET IO	Given by TIA portal
Ident Systems	Main family
ifm electronic	Vendor name
MCR	Product family
O2I5xx	DAP name

### Device tree in TIA portal for O2D5xx

According to the GSDML parameters described above, the following device tree appears in the Siemens TIA Portal:

Other fieldbus devices	Given by TIA portal
PROFINET IO	Given by TIA portal
Sensors	Main family
ifm electronic	Vendor name
Optical Recognition	Product family
O2D5xx	DAP name

## 5.7 I&M data



Unused characters should be filled with blanks (ASCII character 0x20):

- ▷ E.g., when the product code has 6 bytes ( O2I5xx ), the ORDER\_ID will be “ O2I500 ” (13 blanks).

### I&M data for O2I5xx

ID	Datatype	Content
MANUFACTURER_ID	2 octets	0x0136
ORDER_ID	20 octets	O2I5xx
SERIAL_NUMBER	16 octets	Mac address of the device
HARDWARE_REVISION	2 octets	Hardware revision in ifm convention ( AA, AB, AC, ... ) coded as binary number. The current version is AB , which is as a binary number 0x0102 .

ID	Datatype	Content
SOFTWARE_REVISION	4 octets	Software version number in format Vx.y.z (e.g. V1.2.3) has to be coded as bytes: 0x56 0x01 0x03 0x00 V as Ascii char, version number as 3 bytes without dot. Current version: 1.6.12 The version for the fieldbuses is always counted manually, prior to building the application.
REVISION_COUNTER	2 octets	According to I&M specification ( 0x00 )
PROFILE_ID	2 octets	GENERIC_DEVICE
PROFILE_SPECIFIC_TYPE	2 octets	0x00
IM_VERSION	2 octets	According to I&M specification, currently 1.1 ( 0x0101 )
IM_SUPPORTED	2 octets	According to I&M specification ( 0x0E for all except interface slot. Stack takes care of interface)

### I&M data for O2D5xx

ID	Datatype	Content
MANUFACTURER_ID	2 octets	0x0136
ORDER_ID	20 octets	02D5xx
SERIAL_NUMBER	16 octets	Mac address of the device
HARDWARE_REVISION	2 octets	Hardware revision in ifm convention ( AA, AB, AC, ... ) coded as binary number. The current version is AB, which is as a binary number 0x0102.
SOFTWARE_REVISION	4 octets	Software version number in format Vx.y.z (e.g. V1.2.3) has to be coded as bytes: 0x56 0x01 0x02 0x03
REVISION_COUNTER	2 octets	According to I&M specification ( 0x00 )
PROFILE_ID	2 octets	GENERIC_DEVICE
PROFILE_SPECIFIC_TYPE	2 octets	0x00
IM_VERSION	2 octets	According to I&M specification, currently 1.1 ( 0x0101 )
IM_SUPPORTED	2 octets	According to I&M specification ( 0x0E for all except interface slot. Stack takes care of interface)

### I&M5 / AM\_FirmwareOnlyInformation

ID	Datatype	Content
IM_UniqueIdentifier	16 octets	UUID version 5 generated using 207663fa-996f-471a-b216-cc6d59ab7919 namespace and firmware version.
AM_location	16 octets	Zeros
IM_Annotation	64 octets	Camera firmware
IM_OrderId	64 octets	Same as order ID in I&M0
AM_SoftwareRevision	64 octets	Firmware version
IM_SerialNumber	16 octets	Same as in I&M0

ID	Datatype	Content
AM_DeviceIdentification.DeviceSubId	2 octets	Zeros
AM_DeviceIdentification.DeviceId	2 octets	Same as in I&M0
AM_DeviceIdentification.VendorId	2 octets	Same as in I&M0
AM_DeviceIdentification.Organization	2 octets	Zeros (PROFINET)
AM_TypeIdentification	2 octets	Zeros

## 6 IO-Link

The O2I4xx family is equipped with an IO-Link interface. The chapter describes the protocol used inside the cyclically exchanged data of IO-Link.

### 6.1 Data structures

From the application perspective, the IO-Link connection consists of data containers that are exchanged between the device and the IO-Link master.

#### General layout of the IO-Link data containers

The size of the data containers that are exchanged over an IO-Link connection is 32 bytes. Bytes 0...3 of the data container contain the control data. The payload data consists of the bytes 4...31 of the data container.

#### Control data of the input data container

The control data of the input data container consist of these values:

- Byte 0: command (unsigned 8-bit value)
- Byte 1: command counter / message counter mirror (unsigned 8-bit value) [Semantics of command counter / message counter mirror \(→ 56\)](#)
- Byte 2: segment counter (unsigned 8-bit value)
- Byte 3: reserved

#### Control data of the output data container

The control data of the output data container consist of these values:

- Byte 0: command mirror (unsigned 8-bit value)
- Byte 1: message counter / command counter mirror (unsigned 8-bit value) [Semantics of command counter / message counter mirror \(→ 56\)](#)
- Byte 2: segment counter (unsigned 8-bit value)
- Byte 3: Status (unsigned 8-bit value)

#### Change of control data in the output container

The device will change the control data in the output container if one of the following events occur:

- The device replies to a command sent by the IO-Link controller.
- The device has produced a new result while in normal operation mode. ([→ Normal operation 57](#))
- An error event occurred on the device while in normal operation mode.
- The device wants to transmit the next part of segmented payload data (only if segmentation is enabled). ([→ Segmentation of payload data 59](#))

#### Endianness of multi-byte data in the payload data

The byte order of values in the payload data consisting of more than 1 byte is big endian.

#### 6.1.1 Semantics of command counter / message counter mirror

Byte 1 in the output control data has two different meanings depending on the value of the "command mirror" value (byte 0):

- If the "command mirror" value is not equal to 0, the value acts as a "command counter mirror" and mirrors the command counter value of the input control data. This is the case, e.g., if
  - the device is replying to a command sent by the controller,



- the device is acknowledging a segment of a command.
- If the "command mirror" value is 0, the value reflects an independent message counter controlled by the device.  
This is the case, e.g., if the device is sending an asynchronous result or error.

## 6.2 Operating modes

### 6.2.1 Normal operation

#### Activation of normal operation

The IO-Link controller activates the normal operation mode by setting the command value of the input control data to 0.

#### Output of application results

During normal operation, the device will send results appropriately formatted to the IO-Link controller. The production of results is controlled by the configured trigger mode of the active application together with the corresponding trigger sources.

#### Mirroring the activation of the normal operation mode to the IO-Link controller

When the IO-Link controller activates the normal mode, the device:

- sets the command mirror = 0,
- outputs initial value = 0 in the command mirror counter, segment counter and payload data.

After that it starts to generate output data. The special case that the IO-Link controller does not receive the intermediate output of all data = 0 is accepted.

#### Output control data contents of an application result

The control data is set to the following values:

- Command mirror = 0 (indicating the normal operation mode)
- Byte 1 acts as a "message counter". ([→ Semantics of command counter / message counter mirror](#) [□ 56](#)) The value remains constant for all segments that belong to a particular result. The device increments the message counter for each new result, wrapping the value around the limits of the unsigned 8-bit value. I.e., message counter 1 follows message counter 255.
- Segment counter is set to
  - 0 if the application result fits into the payload data or if segmentation is disabled.
  - > 0 if the application result does not fit into the payload data and if segmentation is enabled. [Segmentation of payload data](#) ([→ □ 59](#))
- Status = 0

#### Output payload data contents of an application result

The payload data consist of the result formatted according to the output configuration of the active application.

The default value is device-dependant. Default value for O2I application:

```
{ \"format\": { \"dataencoding\": \"binary\", \"order\": \"busdepending\" },
  \"layouter\": \"flexible\", \"elements\": [ { \"id\":
  \"ApplicationDecodingResult\", \"type\": \"int16\" }, { \"id\": \"Models\",
  \"type\": \"records\", \"elements\": [ { \"id\": \"GroupResults\", \"type\":
  \"records\", \"elements\": [ { \"id\": \"codes\", \"type\": \"records\",
  \"elements\": [ { \"id\": \"content_number_of_bytes\", \"type\": \"int16\" },
  { \"id\": \"content\", \"type\": \"blob\" } ] } ] } ] } ] }
```

### Application results are only output when run mode is active

All application results that are generated when the device is in edit mode will not be output over the IO-Link interface. Only results that are generated during run mode and simulation mode will be sent.

### New outputs in segmentation mode

All results and errors that are generated are not output over the IO-Link interface when the device is in the segmentation phase of the old output. Results and errors are sent only when segmentation is not in progress.

### Output of error events

During normal operation, the device can notify the IO-Link controller about error events that have occurred on the device. The control data is set to the following values:

- Command mirror = 0 (indicating the normal operation mode)
- Byte 1 acts as a "message counter". (→ [Semantics of command counter / message counter mirror](#) □ 56) The value remains constant for all segments that belong to a particular result. The device increments the message counter for each new result, wrapping the value around the limits of the unsigned 8-bit value. I.e., message counter 1 follows message counter 255.
- Segment counter = 0 (error event payloads are never segmented)
- Status = 1 (indicating the error event as opposed to 0 indicating a regular result)

The payload data consists of a single 32-bit unsigned value containing an error code.

## 6.2.2 Command execution

### Initiation of a command execution

The IO-Link controller can initiate a command by setting the command value of the input control data to a value other than 0. Command payload data must be provided in the input data container Byte 4...n.

### Repetition of a command execution

If a previously executed command must be repeated, the IO-Link controller leaves the command value of the input control data at the already appropriate value for the command. Instead, the IO-Link controller modifies the command counter value.

Any input payload data must be provided and sent by the controller, even if the same input is to be used for the repeated command.

### Start of command execution on the device

If the transmitted command does not require segmentation of the input payload data, the execution of the command will start immediately after the successful reception of the input data container containing the command.

With segmented input payload data, the command execution will start after the reception of the last input segment (with segment counter value = 0). It is assumed that all segments belonging to the command have been received successfully.

### Commands sent in rapid succession

The device can only buffer a single command. If the IO-Link controller sends commands in rapid succession, every following command will overwrite the previously buffered command. This may result in some commands not being executed.

The only way to ensure the execution of a command is to wait for the device's reply to said command before sending the next command.

### Output control data contents after the execution of a command

The device responds to the command by setting the control data to the following values:

- Command mirror is set to the command value of the input control data.
- Byte 1 acts as a "command counter mirror". (→ [Semantics of command counter / message counter mirror](#) [□ 56](#))
- Segment counter is set according to [Segmentation of payload data](#) (→ [□ 59](#)).
- Status is set to 0 if the command was executed successfully, otherwise to a value unequal 0. The value is depending on the executed command.

Status value	Name	Description
0	EIP_ERROR_NO_ERROR	No command error
1	EIP_ERROR_CMD_UNKNOWN	Unknown command
2	EIP_ERROR_CMD_FAILED	Command processing was unsuccessful
3	EIP_ERROR_CMD_DATA_INVALID	Invalid data given for the command
4	EIP_ERROR_TOO_MANY_CMDS	Too many commands executed

Tab. 1: Possible values for status

### Non-existing / invalid commands

If the command value of the input control data is set to a value that does not correspond to a valid command, the device still replies to the command with the "command mirror" set to this invalid value. The status is set to 1 and the payload data containing the error code 100000005 (invalid command, unsigned 32-bit value). [Error codes](#) (→ [□ 65](#))

### Output payload data contents after the execution of a command

The payload data after a command execution is depending on the executed command.

## 6.3 Segmentation of payload data

### 6.3.1 Data segmentation and segment counting

#### Number of data containers for segmented payload data

If the payload data size is larger than 28 bytes and data segmentation is enabled, the number of required output data containers can be computed as

$$(\text{payload data size} - 1) \text{ div } 28 + 1$$

#### Order of data segments and initial segment counter value

For each transmitted segment, the segment counter value indicates how many segments will follow the current segment and is therefore initialized to

$$\text{number of required output data containers} - 1.$$

For each following data segment, the segment counter value is decremented by 1 until the last data segment which uses the segment counter value 0.

#### Data segmentation

A data container in a sequence of segmented data contains the payload data starting at offset

$$(\text{initial segment counter value} - \text{segment counter value}) * 28 \text{ up to offset } (\text{initial segment counter value} - \text{segment counter value}) * 28 + 27.$$

### 6.3.2 Output payload data

#### Device parameter for enabling segmentation of output data

The segmentation of output payload data can be controlled by the boolean device parameter `IOLinkEnableSegmentation`:

- When set to true: IO-Link result and command responses will be split up into several segments.
- When set to false: IO-Link result and command responses will be truncated to fit the IO-Link output container.

#### Device parameter for controlling the maximum hold time of a segment of output data

The device parameter `IOLinkMaximumHoldTime` provides the maximum hold time of a segment. It defines the maximum time a single segment of a result or a command response is presented in the IO-Link output container before switching to the next segment.

The parameter can be set in the range of 20 ms to 2000 ms. The default value is 50 ms.

#### Truncation of output payload data if segmentation is not enabled

If segmentation is not enabled, output payload data will be truncated to 28 bytes.

#### Truncation of output payload data if segmentation is enabled

If segmentation is enabled, output payload data will be truncated to 7168 bytes (256 \* 28 bytes).

#### Distribution of output payload data to more than one output data container if segmentation is enabled

If segmentation is enabled and the output payload data size is larger than 28 bytes, the device will transmit the payload data distributed over several output data containers.

#### Progress through data segments depending on maximum hold time

If the device has started the transmission of a data segment, it will continue to do so until the `IOLinkMaximumHoldTime` has elapsed.

It will then proceed with the transmission of the next data segment.

#### Speed up transmission of segmented data by acknowledgement of data segments

The IO-Link controller can speed up the transmission of segmented output data by acknowledging the reception of a segment. This is done by setting the input control data to the following values:

- Command value remains at the previous value if the output data is the response to said command. Otherwise, it is 0 if the output data is a result or an error that arose while the device was in normal operation mode.
- Command counter value remains at its previous value if the controller wants to acknowledge the segment of a reply to a command. Otherwise, it mirrors the message counter of the output control data if the controller wants to acknowledge the segment of an asynchronous result or error.
- The segment counter should mirror the segment counter value of the latest received output data segment.

#### New command aborts transmission of segmented data

If the IO-Link controller executes a new command while a segmented data transmission is in progress, the device will abort that transmission. I.e., any unsent segments are being discarded. After that, immediately start the execution of the new command.

### 6.3.3 Input payload data

#### Segmentation of input payload data is always possible

The device accepts segmented input data without any restriction.

### Acknowledgement for received input data segment

The device acknowledges the reception of an input data segment by setting its output control data to the following values:

- The command mirror value is set to the command value of the input data.
- Byte 1 acts as a "command counter mirror". (→ [Semantics of command counter / message counter mirror](#) □ 56)
- The segment counter value mirrors the segment counter value of the received input data segment.
- Status = 0

No acknowledgement is sent for the last input data segment (with segment counter value 0). Instead, the device will simply send the reply to the command.

### Await acknowledgement before sending the next input data segment

The IO-Link controller must wait for the acknowledgement of an input data segment before sending the next data segment in the sequence.

## 6.4 Commands

### 6.4.1 Software trigger

#### Software trigger input data

The command value for "software trigger" is 1.

There is no input payload data for the "software trigger" command.

#### Software trigger output data

Upon successful execution, the response transmits the application result that has been produced by the execution of this trigger.

The output payload data is identical to the output payload data of an asynchronously sent result as defined below:

The payload data consist of the result formatted according to the output configuration of the active application.

The default value is device-dependant. Default value for O2I application:

```
{ "format": { "dataencoding": "binary", "order": "busdepending" },
  "layouter": "flexible", "elements": [ { "id":
  "ApplicationDecodingResult", "type": "int16" }, { "id": "Models",
  "type": "records", "elements": [ { "id": "GroupResults", "type":
  "records", "elements": [ { "id": "codes", "type": "records",
  "elements": [ { "id": "content_number_of_bytes", "type": "int16" },
  { "id": "content", "type": "blob" } ] } ] } ] } ] }
```

### 6.4.2 Application switch

#### Application-switch input data

The command value for "application switch" is 2.

In byte 4 of the input data container, a single unsigned 8-bit value defines the index of the application that the device switches to.

### Application-switch output data

Upon successful execution, the device replies with status 0 that the switch to the selected application is completed. I.e., the device is "ready for trigger" in case of a triggered application.

## 6.4.3 Get application list

### Get application list input data

The command value for "get application list" is 3.

There is no input payload data for the "get application list" command.

### Get application list output data

Upon successful execution the output data container contains the following values:

- Byte 4 : unsigned 8-bit value containing the total number of applications (total\_apps) on the device.
- Byte 5 : unsigned 8-bit value containing the index of the currently active application.
- Byte 6...n : for every application on the device its index appears in this list as an unsigned 8-bit value. The indices in this list are sorted in ascending order.

## 6.4.4 Get statistics

### Get statistics input data

The command value for "get statistics" is 4.

There is no input payload data for the "get statistics" command.

### Get statistics output data

Upon successful execution the output data container contains the following values:

- Bytes 4...7 : the total number of readings as an unsigned 32-bit value.
- Bytes 8...11 : the number of "pass" results as an unsigned 32-bit value.
- Bytes 12...15 : the number of "fail" results as an unsigned 32-bit value.

## 6.4.5 Reset statistics

### Reset statistics input data

The command value for "reset statistics" is 5.

There is no input payload data for the "reset statistics" command.

### Reset statistics output data

There is no output payload data for the "reset statistics" command.

## 6.4.6 Set input string

### Set input string input data

The command value for "set input string" is 6.

Bytes 4...n of the input data container contain the command payload with following content for the first payload data segment:

- Byte 4 : unsigned 8-bit value containing the ID of the input container to be set.
- Byte 5 : unsigned 8-bit value containing the length of the string.

- Byte 6...n : string content.

For further payload data segments, the input container contains following content:

- Byte 4...n : remaining string content

#### **Set input string output data**

There is not output payload data for the "set input string" command.

### **6.4.7 Get input string**

#### **Get input string input data**

The command value for "get input string" is 7.

In byte 4 of the input data container, a single unsigned 8-bit value defines the ID of the container to be read.

#### **Get input string output data**

Upon successful execution the output data container contains the following values:

- Byte 4 : length of the string as an 8-bit unsigned value
- Byte 5...n : read string content

### **6.4.8 Set focus distance**

#### **Set focus distance input data**

The command value for "set focus distance" is 8.

In bytes 4...7 of the input data container, a single unsigned 32-bit value defines the desired focus distance in mm.

#### **Set focus distance output data**

There is no output payload data for the "set focus distance" command.

### **6.4.9 Get focus distance**

#### **Get focus distance input data**

The command value for "get focus distance" is 9.

There is no input payload data for the "get focus distance" command.

#### **Get focus distance output data**

Upon successful execution, the output data container contains the current focus distance in mm as a single unsigned 32-bit value in bytes 4...7.

### **6.4.10 Execute button teach**

#### **Execute button teach input data**

The command value for "execute button teach" is 10.

There is no input payload data for the "execute button teach" command.

#### **Execute button teach output data**

There is no output payload data for the "execute button teach" command.

### 6.4.11 Extended commands

Currently there are no extended commands.

#### Command value for extended commands

Command value 255 is used for all extended commands. If a value does not represent a command, the error 10000005 is returned as an error. [Error codes \(→ 65\)](#)



## 7 Error codes

By default, the error codes will not be provided by the device. The p command can activate their provision. (→ [p Command \(turn process interface output on or off\)](#) ¶ 20)

Code ID	Error
0	No error
101013	Application number is not available in the device
101022	Application number is invalid
101048	Device is not in run mode
100000001	Maximum number of connections exceeded
100000002	Internal failure during a D-Bus call
100000003	Unspecified internal error
100000004	Generic invalid parameter
100000005	Invalid command
100001000	Application configuration does not allow PCIC trigger
100001001	Video mode does not allow PCIC trigger
100001002	There is no application configured
100001003	Invalid image ID in I? command
100001004	Invalid pin ID in o/O? command
100001005	Invalid pin configuration in o/O? command
100001006	Invalid conversion type selected
100001007	No trigger has been run yet
100001008	Missed decoded frame
100001009	No more segments left
100001010	Command bit 4 not reset to 0
100001012	No such block in logic layer
100001013	Device is not in run or simulation mode
100001014	Internal temperature too high for viewindicator usage
100001015	No button function configured
100001016	Button function already running
100001017	Button function caused an error
100001018	Device in invalid state for button function
100001019	Invalid temporal parameter ID
100001020	Temporal parameter out of range
100001021	Session request failed
100001022	No viewindicators available
110001001	Boot timeout
110001002	Fatal software error
110001003	Unknown hardware
110001004	Fatal log message of diagnostic controller got lost
110001005	Warning log message of diagnostic controller got lost
110001006	Trigger overrun
110001007	Ethernet configuration was changed. The socket is going to be closed.
110002000	Short circuit on OUT3 (Ready for Trigger)
110002001	Short circuit on OUT1

Code ID	Error
110002002	Short circuit on OUT2
110002003	Reverse feeding
110002004	Short circuit on OUT4
110002005	Short circuit on OUT5
110003000	Vled overvoltage
110003001	Vled undervoltage
110003002	Vmod overvoltage
110003003	Vmod undervoltage
110003004	Mainboard overvoltage
110003005	Mainboard undervoltage
110003006	Supply overvoltage
110003007	Supply undervoltage
110003008	VFEMon alarm
110003009	PMIC supply alarm
110004000	Illumination overtemperature
120000001	NTP server not available
120000002	Other NTP error

# Glossar

## AR

---

Application Relationship

## DAP

---

Device Access Point

## DCP

---

The Discovery and Basic Configuration Protocol (DCP) is a protocol definition within the PROFINET context. It is a Data Link Layer based protocol to configure station names and IP addresses. It is restricted to one subnet and mainly used in small and medium applications without an installed DHCP server.

## GSDML

---

General Station Description Markup Language

## JSON

---

Java Script Object Notation

## PCIC

---

Process Communication Interface Component

## PLC

---

Programmable Logic Controller

## PNIO

---

The PROFINET/IO (PNIO) protocol is a fieldbus protocol related to decentralized periphery. PROFINET/IO is based on connectionless DCE/RPC and the "lightweight" PROFINET/RT (Ethernet type 0x8892) protocols: - The context manager (CM) part is handling context information (like establishing) and is using connectionless DCE/RPC as its underlying protocol. - The actual cyclic data transfer and acyclic notification uses the "lightweight" PROFINET/RT protocol. - There are some other related PROFINET protocols (e.g. PROFINET/DCP, which is handling addressing topics).

## UUID

---

Universal Unique Identifier (UUID) used by COM to identify an object and interfaces. Controls the unique identification of a particular functionality in PROFINET.