Protocol API

Object Dictionary

for CANopen and EtherCAT

V3.4

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC110106API04EN | Revision 4 | English | 2017-05 | Released | Public

Table of contents

1 Introduction

1.1 About this document

In CANopen and EtherCAT, the object dictionary is an area to store parameters, application data and the PDO mapping, i.e. the mapping information between process data and application data.

The object dictionary functionality of all three communication systems is quite similar. In all cases, it is based on the CANopen standard which has later been extended by EtherCAT.

Access to the object dictionary is possible via Service Data Objects (SDO) which provide a mailbox-based access functionality.

All data objects are contained in the object dictionary and can be accessed in a standardized manner. You can view the object dictionary as a container for device parameter data structures.

The following SDO services are provided by the protocol stacks for maintaining the object dictionary:

- SDO Upload
- SDO Download
- Services for creating and deleting object containers, entries and data types.

The Protocol API Manuals of the respective protocol describes SDO Upload and SDO Download, while this document concentrates on the description of the object dictionary and the services for creating and deleting object containers and objects therein.

1.2 List of revisions

Rev	Date	Name	Chapter	Revision
3	2013-05-23	RG		Version V3.3.1
			All	Removed all occurrences of Powerlink
			8.1	Added new error messages
			2.5	New access flag added
			2.7	Updated list of packets
			3.15, 4	Added new Packet descriptions
			5.1	Added description of *Multiple Parameter Read/Write Access Flags*
4	2017-05-30	JH, HH		Version 3.4
			2.5	Table 10 expanded.
			3.9	Table 38 corrected.
			3.11	Section ODV3_GET_OBJECT_ACCESS_INFO_REQ/CNF – Get Object Access Info: Structure ODV3_GET_OBJECT_ACCESS_INFO_CNF_DATA_T corrected.
			3.12	Section ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info: Description added.
			3.20	Description added.
			5.1	Description for ODV3_VALUE_INFO_VIRTUAL added.
			5.1	Table 83 updated.
			5.2	Table 96 updated.
			6	Section *Application holds object data and/or object descriptions* added.
			7	Section *Protocol-specific Information* added.

Table 1: List of revisions

1.3 Terms, abbreviations and definitions

Term	Description
AP (-task)	Application (-task) on top of the stack
CAN	Controller Area Network
CiA	CAN in Automation
CoE	CAN application protocol over EtherCAT
DPM	Dual Port Memory
ETG	EtherCAT Technology Group
IP	Internet-Protocol
MAC	Medium Access Control
OD	Object Dictionary
ODv3	Object Dictionary version 3
PDO	Process Data Object
SDO	Service Data Object

Table 2: Terms, abbreviations and definitions

All variables, parameters, and data used in this manual have the LSB/MSB (“Intel”) data format. This corresponds to the convention of the Microsoft C Compiler.

1.4 References to documents

This document refers to the following documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX-based products. Revision 12, English, 2013

[2] CAN in Automation e.V.: CANopen Application Layer and Communication Profile, CiA Draft Standard 301, Version 4.02

[3] International Electrotechnical Commission: IEC 61158-3-12, Industrial communication networks – Fieldbus specifications – Part 3-12: Data-link layer service definition – Type 12 elements

[4] International Electrotechnical Commission: IEC 61158-4-12, Industrial communication networks – Fieldbus specifications – Part 4-12: Data-link layer protocol specification– Type 12 elements

[5] International Electrotechnical Commission: IEC 61158-5-12, Industrial communication networks – Fieldbus specifications – Part 5-12: Application layer service definition – Type 12 elements

[6] International Electrotechnical Commission: IEC 61158-6-12, Industrial communication networks – Fieldbus specifications – Part 6-12: Application layer service definition – Type 12 elements

[7] IETF Network Working Group, K. McCloghrie, M. Rose (ed.): RFC1213 „Management Information Base for Network Management of TCP/IP-based internets: MIB II“,1991

Table 3: References to documents

1.5 Technical data

Features	Parameter
Maximum number of indices	65535
Maximum number of subindices	255
Supported by CANopen Slave protocol stack	Version 3.x
Supported by EtherCAT Slave protocol stack	Version 4.x

Table 4: Technical data

The firmware and stack has been written in order to meet the respective specifications.

For technical data about CANopen Slave or EtherCAT Slave protocol stack refer to the respective Protocol API Manual.

1.6 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert

damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fusion processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

1.7 Registered trademarks

CANopen® is a registered trademark of CAN in AUTOMATION - International Users and Manufacturers Group e.V. (CiA), Nürnberg.

EtherCAT® is a registered trademark and patented technology of Beckhoff Automation GmbH, Verl, Germany.

All other mentioned trademarks are property of their respective legal owners.

1.8 EtherCAT Disclaimer, Vendor ID, Conformance Test, Membership and Network Logo

1.8.1 EtherCAT Disclaimer

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

To get details and restrictions regarding using the EtherCAT technology refer to the following documents:

- “EtherCAT Marking rules”
- “EtherCAT Conformance Test Policy”
- “EtherCAT Vendor ID Policy”

These documents are available at the ETG homepage www.ethercat.org or directly over info@ethercat.org.

A summary over Vendor ID, Conformance test, Membership and Network Logo follows.

1.8.2 Vendor ID

The communication interface product is shipped with Hilscher’s secondary vendor ID, which has to be replaced by the Vendor ID of the company shipping end products with the integrated communication interface. End Users or Integrators may use the communication interface product without further modification if they re-distribute the interface product (e.g. PCI Interface card products) only as part of a machine or machine line or as spare part for such a machine. In case of questions, contact Hilscher and/or your nearest ETG representative. The ETG Vendor-ID policies apply.

1.8.3 Conformance

EtherCAT Devices have to conform to the EtherCAT specifications. The EtherCAT Conformance Test Policies apply, which can be obtained from the EtherCAT Technology Group (ETG, www.ethercat.org).

Hilscher range of embedded network interface products are conformance tested for network compliance. This simplifies conformance testing of the end product and can be used as a reference for the end product as a statement of network conformance (when used with standard operational settings). It must however be clearly stated in the product documentation that this applies to the network interface and not to the complete product.

Conformance Certificates can be obtained by passing the conformance test in an official EtherCAT Conformance Test lab. Conformance Certificates are not mandatory, but may be required by the end user.

1.8.4 Certified Product vs. Certified Network Interface

The EtherCAT implementation may in certain cases allow one to modify the behavior of the EtherCAT network interface device in ways which are not in line with EtherCAT conformance requirements. For example, certain communication parameters are set by a software stack, in which case the actual software implementation in the device application determines whether or not the network interface can pass the EtherCAT conformance test. In such cases, conformance test of the end product must be passed to ensure that the implementation does not affect network compliance.

Generally, implementations of this kind require in-depth knowledge in the operating fundamentals of EtherCAT. To find out whether or not a certain type of implementation can pass conformance testing and requires such testing, contact EtherCAT Technology Group (“ETG”, www.ethercat.org) and/or your nearest EtherCAT conformance test centre. EtherCAT may allow the combination of an untested end product with a conformant network interface. Although this may in some cases make it possible to sell the end product without having to perform network conformance tests, this approach is generally not endorsed by Hilscher. In case of questions, contact Hilscher and/or your nearest ETG representative.

1.8.5 Membership and Network Logo

Generally, membership in the network organization and a valid Vendor-ID are prerequisites in order to be able to test the end product for conformance. This also applies to the use of the EtherCAT name and logo, which is covered by the ETG marking rules.

Vendor ID Policy accepted by ETG Board of Directors, November 5, 2008

2 Getting started

2.1 Questions and answers about the ODV3

What is an object dictionary?

An object dictionary is a storage area for device parameter data and acts as a link between the application and the protocol stack.

What is ODv3?

ODv3 implements the object dictionaries defined in the specifications of these systems:

- CANopen,
- EtherCAT

In these systems, all device parameters are stored within the object dictionary and all acyclic communication is done using the object dictionary.

What is the contents of the object dictionary and how can the data in the object dictionary be classified?

The object dictionary contains three different kinds of objects and data types (also see subsection *General structure* on page 14):

1. Predefined Standard objects and data types as defined in the specification of the respective system (CANopen: reference [2], EtherCAT: references [3], [4], [5], and [6])
2. Manufacturer-specific objects and data types
3. Objects and data types defined in device profiles

Besides the data itself, the object dictionary also stores the data types to be applied.

- For more information about the predefined objects see section 10.1.1 “0x0020: PDO_COMMUNICATION_PARAMETER” and the following sections of the appendix of this document.
- For more information about the predefined data types see section 10.1 “*Detailed Description of Data Types*” in the appendix of this document.

What kind of services does ODv3 offer?

ODv3 offers three classes of services:

- Basic services
- Compound services
- Management services

What are basic services?

Basic services are services such as reading and writing access to the object dictionary and validation services.

What are compound services?

Compound services are services that uses basic services several times.

What are management services?

Management services are services such as creation and deletion of objects, subobjects and data types within the object dictionary, read and write notification services, locking and renaming services.

Which services do I need?

This depends on the kind of user you are. ODv3 users can be classified as follows:

1. If you are only interested in performing cyclic I/O, you will not have to use ODv3 at all.
2. If you intend to work with the pre-defined data sets defined in the object dictionary, you only need basic services for read and write access and possibly the information services of ODv3.
3. If you want to change, extend or add objects you will need some management services like create object, create subobject and probably the notification services.
4. If you want to implement profiles or manufacturer-specific extensions or you want to have control over all objects, you will have to use the management services extensively, especially the notification services.

This chapter

The rest of this chapter presents the following important topics of ODv3:

- General structure
- Lists of available data types
- List of the objects of the communication area
- Object access masks
- Access rights
- Fragmentation
- Sequence Diagrams for indications

The next chapters contain a detailed description of the packet interface of the basic services, information services and management services. A list of error codes and an appendix describing all predefined objects of the CANopen and EtherCAT specifications are also supplied.

Finally, a glossary explains some special terms.

2.2 General structure of the object dictionary

The object dictionary is structured in separate areas. Each area has its own range of permitted index values and its special purpose as defined in the following table:

Index range	Area name	Purpose
0x0000 – 0x0FFF	Data type area	Definition and description of data types.
0x1000 – 0x1FFF	Communication area	Definition of generally applicable variables (communication objects for all devices as defined by CANopen standard DS 301).
0x2000 – 0x5FFF	Manufacturer-specific area	Definition of manufacturer-specific variables
0x6000 – 0x9FFF	Profile area	Definition of variables related to a specific profile
0xA000 – 0xFFFF	Reserved area	This area is reserved for future use

Table 5: General structure of object dictionary

2.2.1 Object codes

Object code specifies the kind of object. For additional information you may see protocol specifications. The following kinds of objects may be defined within the object directory:

Object code	Object name	Comment
0000	NULL	Entry with no data fields
0002	DOMAIN	Large amount of data
0005	DEFTYPE	Denotes a basic type definition
0006	DEFSTRUCT	Defines a structure type definition
0007	VAR	A single value
0008	ARRAY	Object with a set of subobjects of the same data type
0009	RECORD	Object with a set of subobjects of any i.e. mixed data type
0028	EtherCAT only: ENUM definition	

Table 6: Object codes

2.3 Lists of available data types

2.3.1 Available data type definitions – Basic data type area

Data types can be defined in the data type area of the object dictionary (object DEFTYPE) using the following indices as follows:

Data type index	Name
0001	BOOLEAN
0002	INTEGER8
0003	INTEGER16
0004	INTEGER32
0005	UNSIGNED8
0006	UNSIGNED16
0007	UNSIGNED32
0008	REAL32
0009	VISIBLE_STRING
000A	OCTET_STRING
000B	UNICODE_STRING
000C	TIME_OF_DAY
000D	TIME_DIFFERENCE
000E	Reserved
000F	DOMAIN
0010	INTEGER24
0011	REAL64
0012	INTEGER40
0013	INTEGER48
0014	INTEGER56
0015	INTEGER64
0016	UNSIGNED24
0017	Reserved
0018	UNSIGNED40
0019	UNSIGNED48
001A	UNSIGNED56
001B	UNSIGNED64
001C-001F	Reserved

Table 7: Available data type definitions – Basic data type area

For more information please refer for CANopen to reference [2], for EtherCAT to references [3], [4], [5], and [6]. There you will find the exact definitions of all of these data types.

2.3.2 Available data type definitions – Extended data type area

Data type index	Name	Object code	CANopen	EtherCAT
0020	PDO_COMMUNICATION_PARAMETER	DEFSTRUCT	x	
0021	PDO_MAPPING	DEFSTRUCT	x	x
0022	SDO_PARAMETER	DEFSTRUCT	x	
0023	IDENTITY	DEFSTRUCT	x	x
0024	Reserved		x	x
0025	COMMAND_PAR	DEFSTRUCT		x
0026-0028	Reserved		x	x
0029	SYNC_PAR	DEFSTRUCT		x
002A-002F	Reserved		x	x
0030	BIT1	DEFTYPE		x
0031	BIT2	DEFTYPE		x
0032	BIT3	DEFTYPE		x
0033	BIT4	DEFTYPE		x
0034	BIT5	DEFTYPE		x
0035	BIT6	DEFTYPE		x
0036	BIT7	DEFTYPE		x
0037	BIT8	DEFTYPE		x
0038-003F	Reserved		x	x
0040-005F	Manufacturer Specific Complex Data Types	DEFSTRUCT	x	x
0060-007F	Device Profile 0 Specific Standard Data Types	DEFTYPE	x	x
0080-009F	Device Profile 0 Specific Complex Data Types	DEFSTRUCT	x	x
00A0-00BF	Device Profile 1 Specific Standard Data Types	DEFTYPE	x	x
00C0-00DF	Device Profile 1 Specific Complex Data Types	DEFSTRUCT	x	x
00E0-00FF	Device Profile 2 Specific Standard Data Types	DEFTYPE	x	x
0100-011F	Device Profile 2 Specific Complex Data Types	DEFSTRUCT	x	x
0120-013F	Device Profile 3 Specific Standard Data Types	DEFTYPE	x	x
0140-015F	Device Profile 3 Specific Complex Data Types	DEFSTRUCT	x	x
0160-017F	Device Profile 4 Specific Standard Data Types	DEFTYPE	x	x
0180-019F	Device Profile 4 Specific Complex Data Types	DEFSTRUCT	x	x
01A0-01BF	Device Profile 5 Specific Standard Data Types	DEFTYPE	x	x
01C0-01DF	Device Profile 5 Specific Complex Data Types	DEFSTRUCT	x	x
01E0-01FF	Device Profile 6 Specific Standard Data Types	DEFTYPE	x	x
0100-021F	Device Profile 6 Specific Complex Data Types	DEFSTRUCT	x	x
0220-023F	Device Profile 7 Specific Standard Data Types	DEFTYPE	x	x
0240-025F	Device Profile 7 Specific Complex Data Types	DEFSTRUCT	x	x
0260-07FF	Reserved	Reserved		x
0800-0FFF	Enumerated data type area (see EtherCAT Specification, Part 6, Table 65)			x

Table 8: Available data type definitions – Extended data type area

This table mentions various structured data types (for instance, all structure definitions marked with DEFSTRUCT in column ‘object’).

For more information please refer to the appendix and the CANopen or EtherCAT specifications. There you will also find the exact definitions of all of these data types.

2.4 List of the objects of the communication area

This section provides a common view onto the communication area of CANopen and EtherCAT describing all objects defined in at least one of these systems (in the range 0x1000 to 0x1AFF).

The following table shows, how the communication area is structured and via color coding or column *System*, which objects belong to which communication system (CANopen or EtherCAT):

Communication area						
Data type index (hex)	**Object**	**Name**	**Type**	**Access**	**M/O/C**	**System (CANopen or EtherCAT)**
1000	VAR	Device type	UNSIGNED32	ro	M	CANopen, EtherCAT
1001	VAR	Error register/ ERR_ErrorRegister_U8	UNSIGNED8	ro	M	CANopen
1002	VAR	Manufacturer status register	UNSIGNED32	ro	O	CANopen
1003	ARRAY	Pre-defined error field/ ERR_History_ADOM	UNSIGNED32	ro	O	CANopen
1004	(reserved)					
1005	VAR	COB-ID SYNC	UNSIGNED32	rw	O	CANopen
1006	VAR	Communication cycle period/ NMT_CycleLen_U32	UNSIGNED32	rw	O	CANopen
1007	VAR	Synchronous window length	UNSIGNED32	rw	O	CANopen
1008	VAR	Manufacturer device name	String	const	O	CANopen, EtherCAT
1009	VAR	Manufacturer hardware version	String	const	O	CANopen, EtherCAT
100A	VAR	Manufacturer software version	String	const	O	CANopen, EtherCAT
100B	reserved					
100C	VAR	Guard time	UNSIGNED16	rw		CANopen
100D	VAR	Life time factor	UNSIGNED8	rw		CANopen
100E	reserved					
100F	reserved					
1010	ARRAY	Store parameters/ NMT_StoreParam_REC	UNSIGNED32	rw		CANopen
1011	ARRAY	Restore default parameters/ NMT_RestoreDefParam_REC	UNSIGNED32	rw		CANopen
1012	VAR	COB-ID TIME	UNSIGNED32	rw		CANopen
1013	VAR	High resolution time stamp	UNSIGNED32	rw		CANopen
1014	VAR	COB-ID EMCY	UNSIGNED32	rw		CANopen
1015	VAR	Inhibit time EMCY	UNSIGNED16	rw		CANopen
1016	ARRAY	Consumer heartbeat time	UNSIGNED32	rw		CANopen
1017	VAR	Producer heartbeat time	UNSIGNED16	rw		CANopen
1018	RECORD	Identity object	Identity (23h)	ro	M	CANopen,

Communication area						
Data type index (hex)	**Object**	**Name**	**Type**	**Access**	**M/O/C**	**System (CANopen or EtherCAT)**
						EtherCAT
1019		Reserved				
101A		Reserved				
::::	::::	::::	::::		::::	
10F1		Error settings		rw	O	EtherCAT
10F3	RECORD	Diagnosis history				EtherCAT
10F4	RECORD	External synchronization status				EtherCAT
10F5	RECORD	External synchronization settings				EtherCAT
::::	::::	::::	::::		::::	
1200-127F	RECORD	CANopen: SDO server parameter	CANopen: SDO_PARAMETER		C	CANopen
1280-12FF	RECORD	CANopen: SDO client container parameter	CANopen: SDO_PARAMETER		C o	CANopen
1400-14FF	RECORD	CANopen: Receive PDO communication parameters	CANopen: PDO CommPar		C	CANopen
1600-17FF	ARRAY	CANopen, EtherCAT: Receive PDO Mapping	UNSIGNED64		C	CANopen, EtherCAT
1800-19FF	RECORD	CANopen: Transmit PDO communication parameters	CANopen: PDO CommPar		C	CANopen
1A00-1AFF	ARRAY	CANopen, EtherCAT: Transmit PDO mapping	UNSIGNED64		C	CANopen, EtherCAT
1C00	RECORD	Sync manager communication type	SYNC_PAR		M	EtherCAT

Table 9: Communication area - General overview

In this context:

- M means Mandatory, O means Optional, C means Conditional.
- ro means “read only”.
- rw means “read/write”.

The color coding of Table 9 is as follows:

- Red lines indicate “only valid for CANopen”.
- Orange lines indicate “only valid for CANopen”.
- Blue lines indicate “only valid for EtherCAT”.

2.5 Object access masks

Object access masks are used for generation of lists of objects matching some special criteria. They can be used for filter purpose e.g. to display a list of objects and should not be mixed up with access rights. If a flag should be set on a subobject, the flag also has to be set on object level otherwise the subobject will not occur in the list.

The following packets uses object access masks:

- ODV3_CREATE_OBJECT_REQ/CNF – Create Object
- ODV3_GET_OBJECT_COUNT_REQ/CNF – Get Object Count
- ODV3_GET_OBJECT_COUNT_IND/RES – Get Object Count
- ODV3_GET_OBJECT_ACCESS_INFO_REQ/CNF – Get Object Access Info

Object access masks have the following values:

Bit	Name and description
D31	ODV3_ACCESS_FLAGS_SETTINGS Settings: This is a predefined flag for use by the application to indicate that an object belongs to the configuration set. E.g. 0x80xx objects in EtherCAT´s modular device profile could be marked this way. It is ment for filter purpose if the object dictionary is very large. The attribute does not make sense for mapping objects. Values of setting objects are volatile if they are not also backup objects.
D30	ODV3_ACCESS_FLAGS_BACKUP Backup: This is a predefined object access type for backup purposes. The value of the object marked with this attribute is saved non-volatile in the slave.
D29	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE TXPDO mappable: This is a predefined flag for use by the application to indicate that an object can be mapped into a TXPDO (= Transmit PDO).
D28	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE RXPDO mappable: This is a predefined flag for use by the application to indicate that an object can be mapped into a RXPDO (= Receive PDO).
D27	ODV3_ACCESS_FLAGS_FORCE_INDEXED Force indexed (if bMaxSubIdx == 0, otherwise implied) Since the object type code is not evaluated, this flag defines the access type to the object. There are two modes: Non-indexed operation: This mode accesses a single sub-object at sub index 0. This mode is selected if bMaxSubIdx == 0 and the flag is not set. (For objects with object code VAR.) Indexed operation This mode accesses a set of sub-objects. Sub index 0 indicates the number of additional sub-objects. This mode is selected if bMaxSubIdx != 0 or the flag is set. Set this flag for objects which do not have the object code VAR but have bMaxSubIdx = 0.
D26	ODV3_ACCESS_FLAGS_CREATE_SUBINDEX_0 Create subindex 0: This is a predefined object access type for creation of sub index 0. **Note:** Only valid on objects like RECORD, ARRAY etc. The creation of subindex 0 is done during creation of the object, if this flag is set with data type UNSIGNED8.
D25	ODV3_ACCESS_FLAGS_SUBINDEX_0_WRITE_0_FIRST Write subindex 0 first On writing all subindexes starting from subindex 0, write 0 to subindex 0 first. This flag is needed for EtherCAT objects that can be written using Complete Access.

Table 10: Access flags usAccessFlags

2.6 Access rights

CANopen

The access rights are defined as follows for CANopen:

Value	Meaning
rw	Read and write access
wo	Write only access
ro	Read only access
const	Read only access, value is constant

Table 11: Access rights for CANopen

EtherCAT

For EtherCAT, the access rights are defined as bit mask according to the table below:

Access rights			
Bit	**Abbreviation**	**Full Name**	**Description**
D15	-	ECAT_OD_WRITE_INIT	Write Access in Initial State
D14	-	ECAT_OD_READ_INIT	Read Access in Initial State
D13-D10	-		Reserved
D9	-	ECAT_OD_SETTINGS	Object can be used for settings
D8	-	ECAT_OD_BACKUP	Object can be used for backup
D7	TX	ECAT_OD_MAPPABLE_IN_TXPDO	Object is mappable in TXPDO
D6	RX	ECAT_OD_MAPPABLE_IN_RXPDO	Object is mappable in RXPDO
D5	WO	ECAT_OD_WRITE_OPERATIONAL	Write Access in Operational State
D4	WS	ECAT_OD_WRITE_SAFEOP	Write Access in Safe-Operational State
D3	WP	ECAT_OD_WRITE_PREOP	Write Access in Pre-Operational State
D2	RO	ECAT_OD_READ_OPERATIONAL	Read Access in Operational State
D1	RS	ECAT_OD_READ_SAFEOP	Read Access in Safe-Operational State
D0	RP	ECAT_OD_READ_PREOP	Read Access in Pre-Operational State

Table 12: Access rights for EtherCAT

2.7 Fragmentation

The object dictionary supports fragmentation for the following packets:

- ODV3_READ_OBJECT_REQ/CNF – Read Object from Dictionary
- ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary
- ODV3_WRITE_OBJECT_REQ/CNF – Write Object to Dictionary
- ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary
- ODV3_READ_OBJECT_NO_IND_REQ/CNF – Read Object No Indication
- ODV3_GET_OBJECT_LIST_REQ/CNF – Get Object List matching the provided Access Mask
- ODV3_GET_OBJECT_LIST_IND/RES – Get Object List matching the provided Access Mask Indication
- ODV3_GET_OBJECT_INFO_REQ/CNF – Get Object Info
- ODV3_GET_OBJECT_INFO_IND/RES – Get Object Info
- ODV3_GET_SUBOBJECT_INFO_REQ/CNF – Get Subobject Info
- ODV3_GET_SUBOBJECT_INFO_IND/RES – Get Subobject Info Indication
- ODV3_CREATE_OBJECT_REQ/CNF – Create Object
- ODV3_CREATE_SUBOBJECT_REQ/CNF – Create Subobject
- ODV3_READ_ALL_BY_INDEX_REQ/CNF – Read All by Index
- ODV3_WRITE_ALL_BY_INDEX_REQ/CNF – Write All by Index

This allows transferring data being larger than the available maximum packet size.

2.7.1 Destination is receiving fragmented request/indication data

<REQUEST TYPE> refers to the actual command. So, if an ODV3_WRITE_OBJECT_REQ is assumed, <REQUEST_TYPE> would be replaced by WRITE_OBJECT.

Note: Not all services support indications. On those services, only the parts about Request/Confirmation apply

2.7.1.1 Request/Confirmation

Successful transfer

Note: If the fragmentation has only two segments, there is no TLR_PACKET_SEQ_MIDDLE segment.

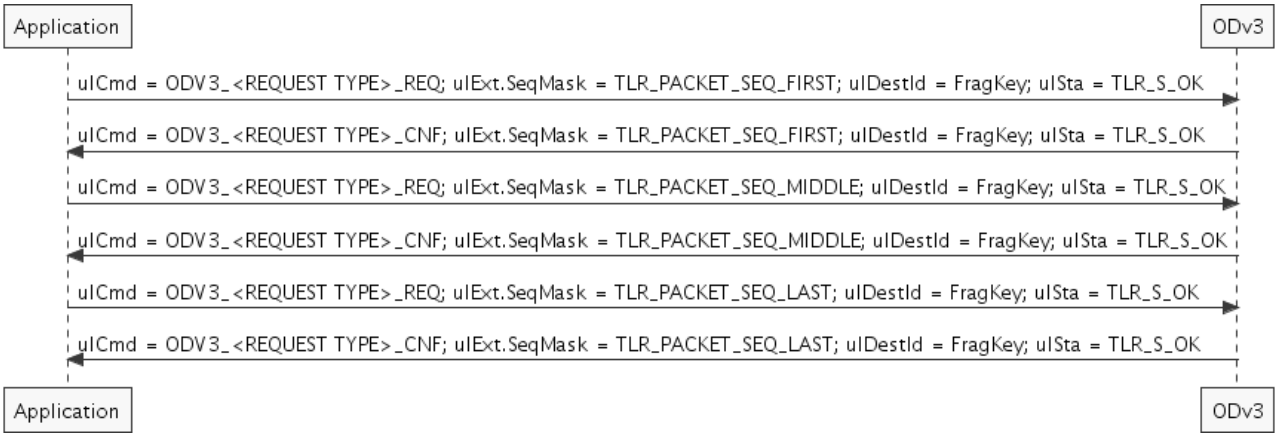

Figure 1: Request/Confirmation - Successful transfer

Aborted transfer by ODv3 – Fragment in progress by ODv3

The transfer was started by application but the ODv3 detected an error in a fragment.

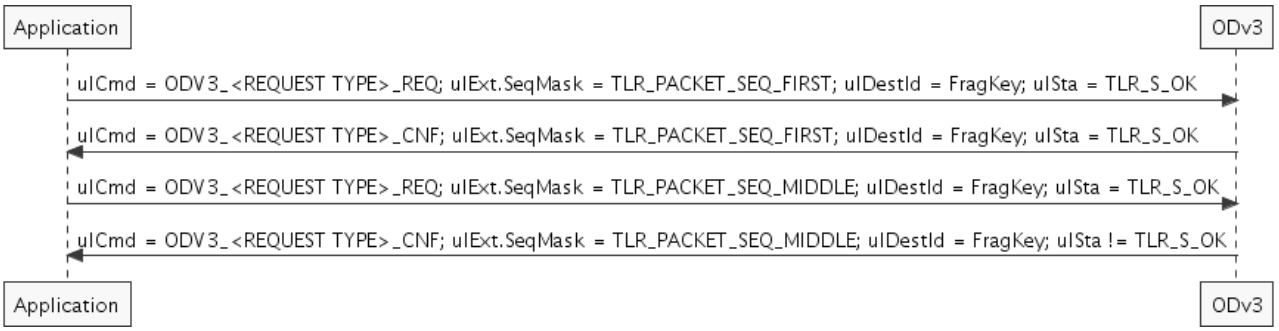

Figure 2: Request/Confirmation - Aborted transfer by ODv3 – Fragment in progress by ODv3

Aborted transfer by ODv3 – No fragment in progress by ODv3

The transfer was started by application but the ODv3 detected a timeout or an error without having a fragment available.

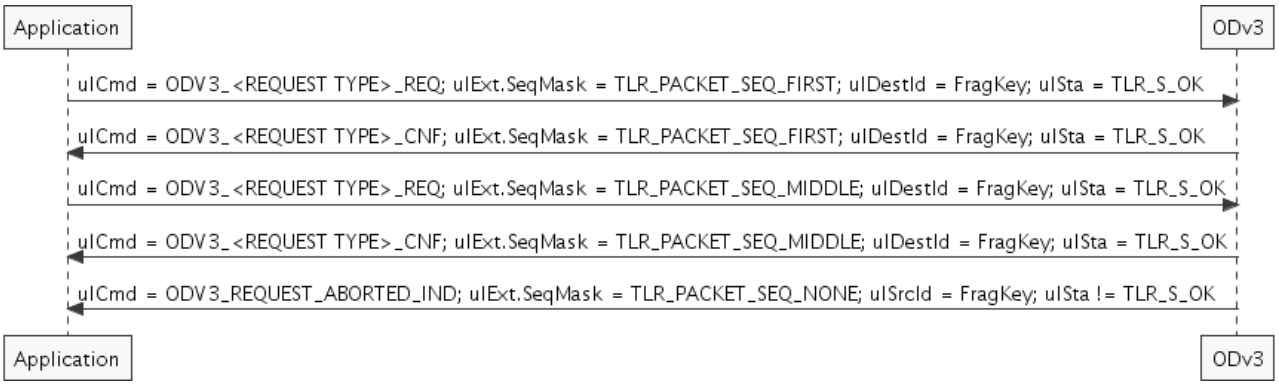

Figure 3: Request/Confirmation - Aborted transfer by ODv3 – No fragment in progress by ODv3

Aborted transfer by application task

The application task started a transfer and decided not to continue the transfer, so it sends a fragment with an error code.

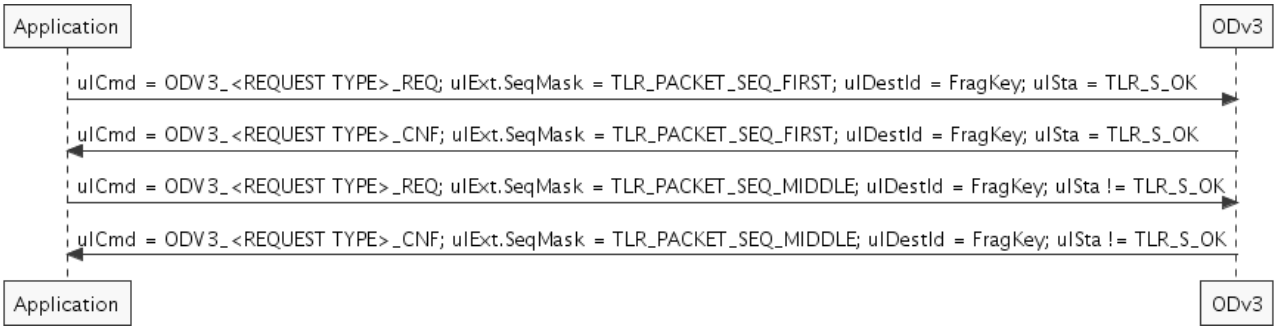

Figure 4: Request/Confirmation - Aborted transfer by application task

2.7.1.2 Indication/Response

Successful transfer

Note: If the fragmentation has only two segments, there is no TLR_PACKET_SEQ_MIDDLE segment.

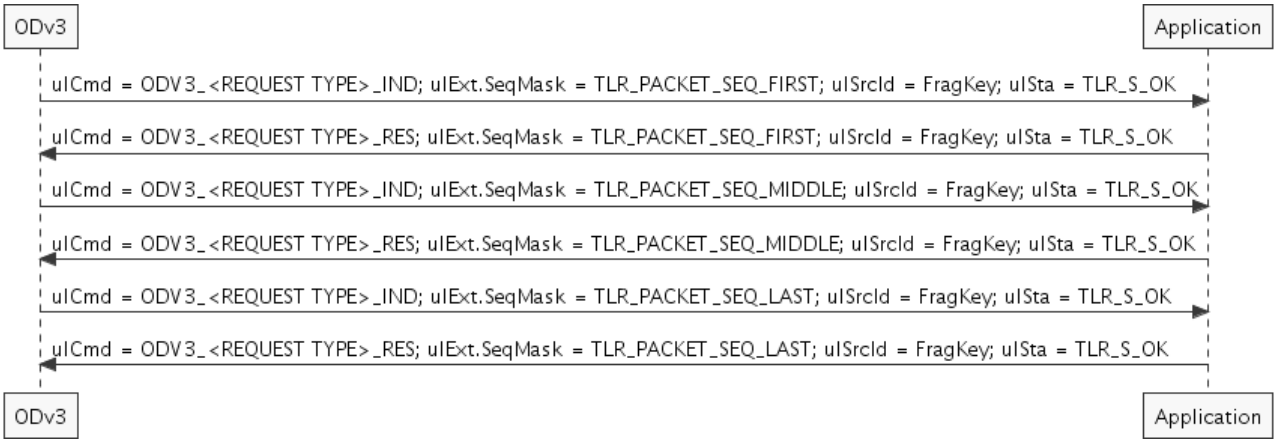

Figure 5: Indication/Response - Successful transfer

Aborted transfer by registered application

ODv3 sends an indication to a registered application and the registered application decided not to continue the transfer or abort it.

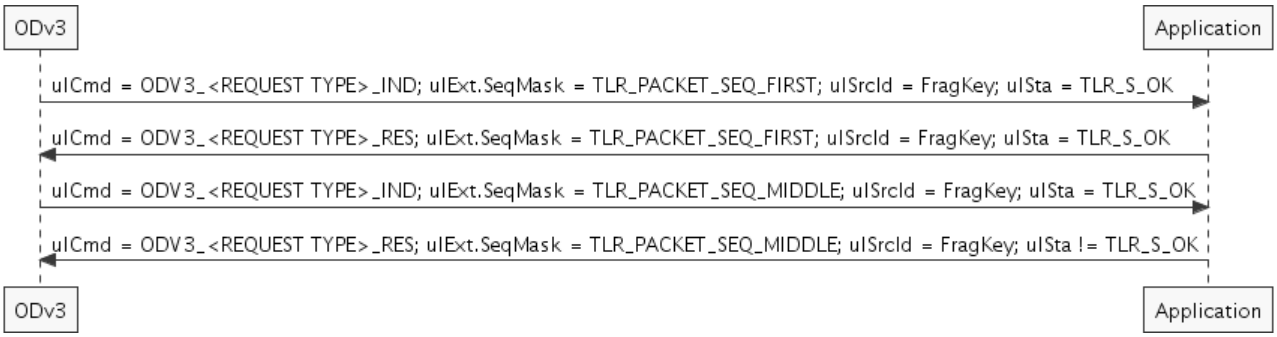

Figure 6: Indication/Response - Aborted transfer by registered application

Aborted transfer by ODv3 task

ODv3 sends indication to a registered application and detects an error e.g. timeout or any other error on the actual request being processed within ODv3.

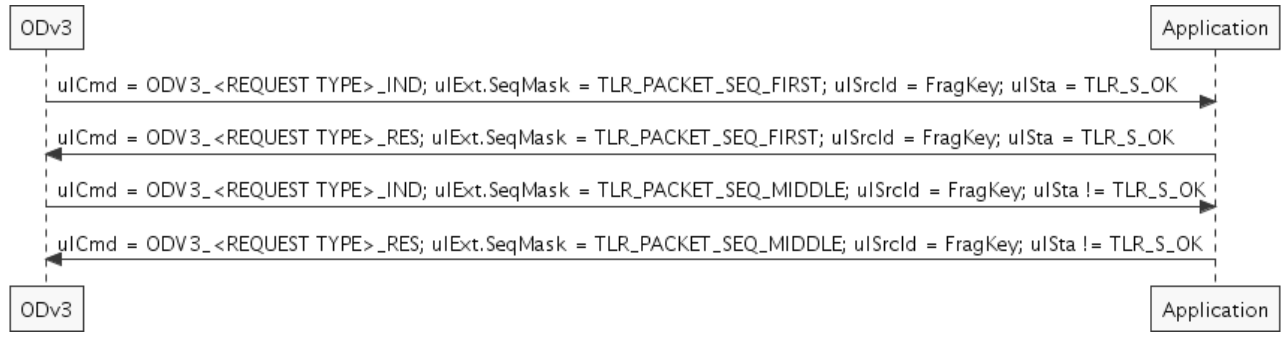

Figure 7: Indication/Response - Aborted transfer by ODv3 task

2.7.2 Destination is replying with fragmented confirmation/response data

<REQUEST TYPE> refers to the actual command. So, if an ODV3_READ_OBJECT_REQ is assumed, <REQUEST_TYPE> would be replaced by READ_OBJECT.

Note: Not all services support indications. On those services, only the parts about Request/Confirmation apply.

2.7.2.1 Request/Confirmation

Successful transfer

Note: If the fragmentation has only two segments, there is no TLR_PACKET_SEQ_MIDDLE segment.

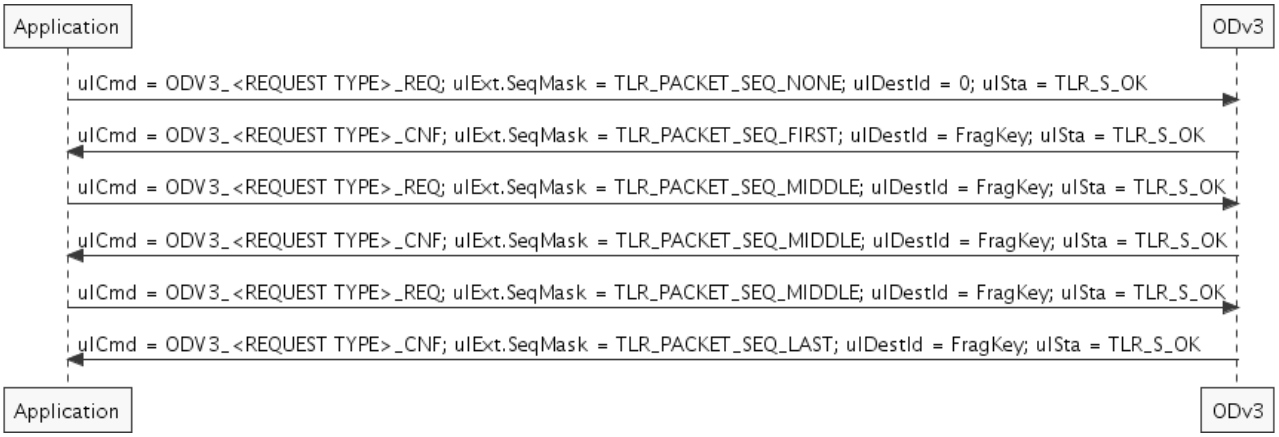

Figure 8: Request/Confirmation - Successful transfer

Aborted transfer by ODv3 – Fragment in progress by ODv3

The transfer was started by the application but ODv3 detected an error during the transfer.

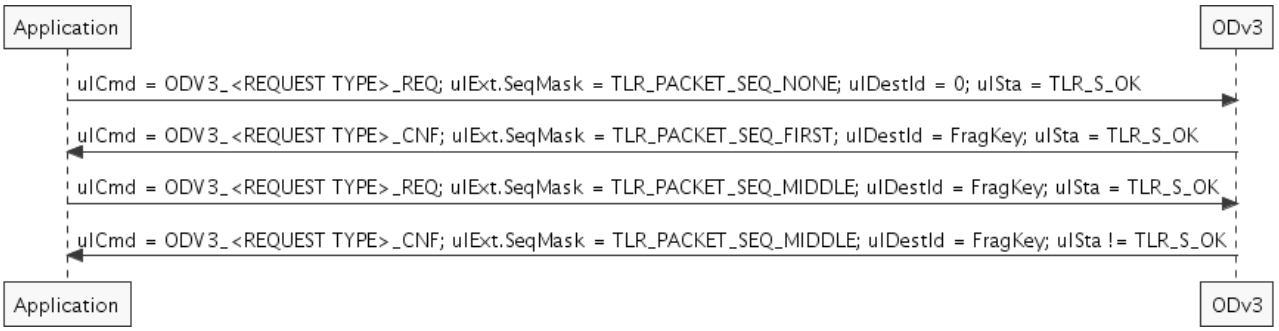

Figure 9: Request/Confirmation - Aborted transfer by ODv3 – Fragment in progress by ODv3

Aborted transfer by ODv3 – No fragment in progress by ODv3

The transfer was started by application but the ODv3 detected a timeout or an error without having a fragment available.

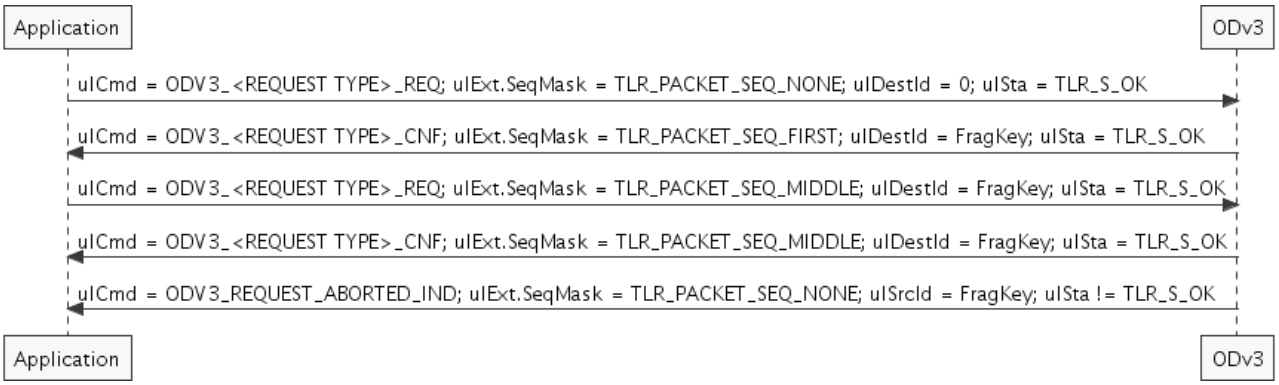

Figure 10: Request/Confirmation - Aborted transfer by ODv3 – No fragment in progress by ODv3

Aborted transfer by application task

The application task started a transfer and decided not to continue the transfer, so it sends a fragment with an error code.

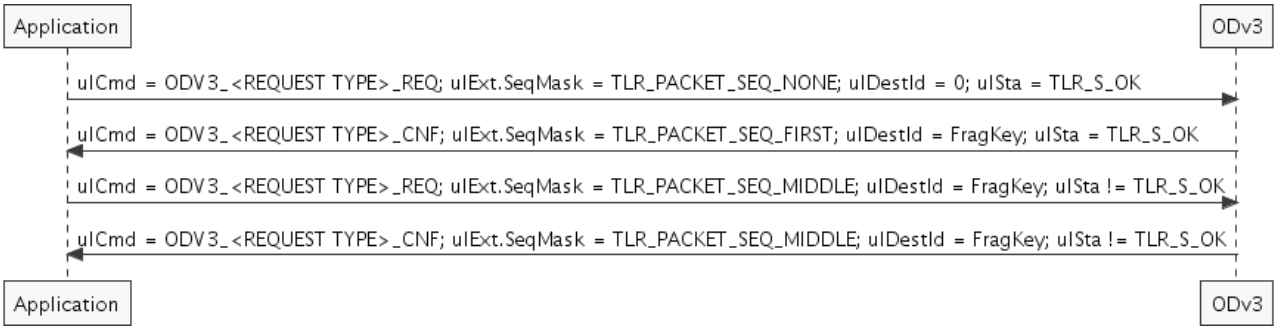

Figure 11: Request/Confirmation - Aborted transfer by application task

2.7.2.2 Indication/Response

Successful transfer

Note: If the fragmentation has only two segments, there is no TLR_PACKET_SEQ_MIDDLE segment.

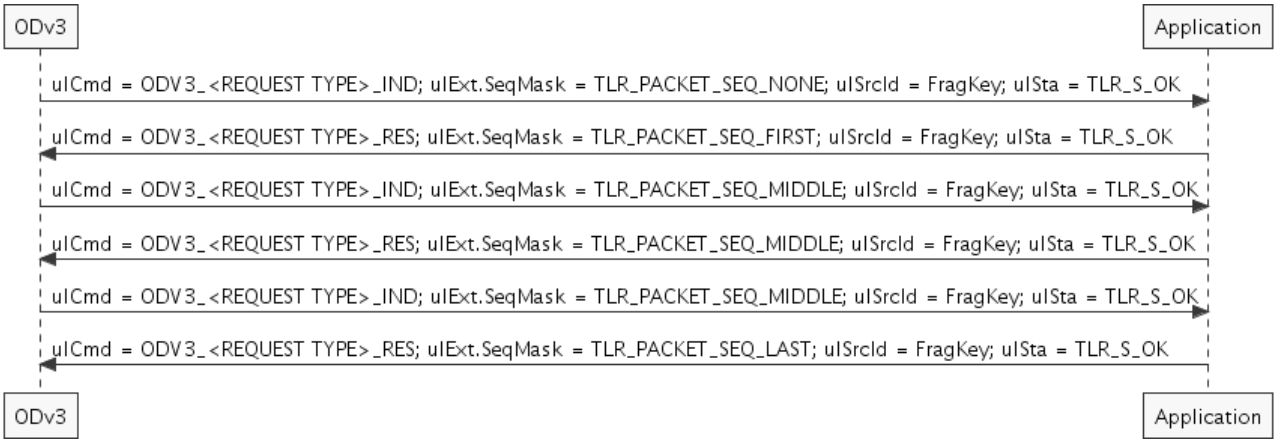

Figure 12: Indication/Response - Successful transfer

Aborted transfer by registered application

ODv3 sends an indication to a registered application and the registered application decided not to continue the transfer or abort it.

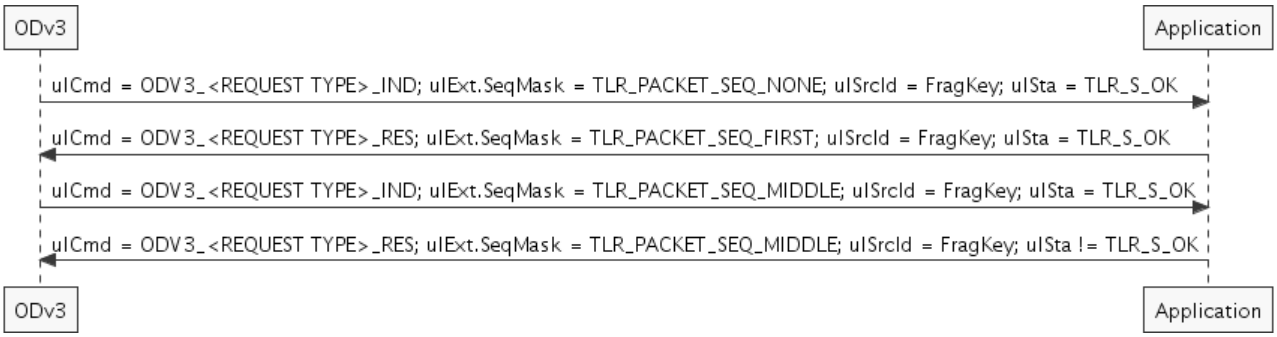

Figure 13: Indication/Response - Aborted transfer by registered application

Aborted transfer by ODv3 task

ODv3 sends indication to a registered application and detects an error e.g. timeout or any other error on the actual request being processed within ODv3.

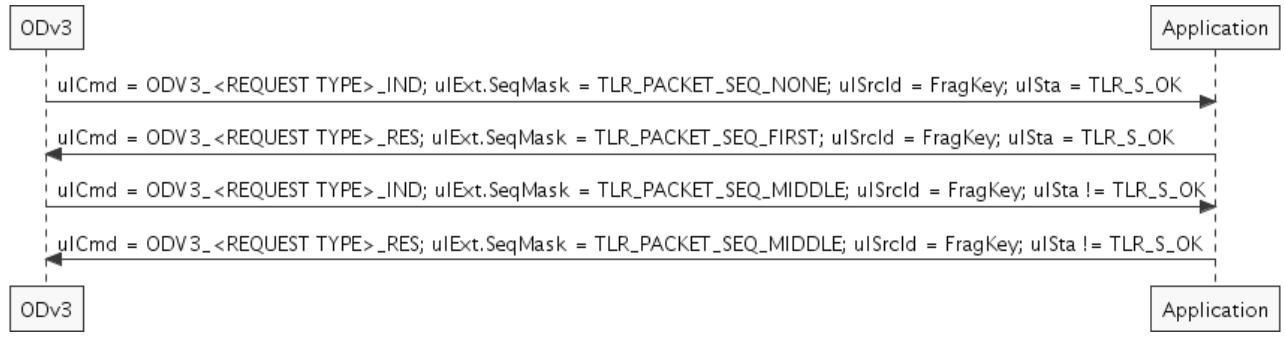

Figure 14: Indication/ Response - Aborted transfer by ODv3 task

2.8 Sequence Diagrams for indications

2.8.1 A single application is registered for write indication

If a single application is registered for write indication of an object/subobject, the following diagrams apply. In all of the following diagrams the time axis extends vertically from top to bottom.

2.8.1.1 Application successful

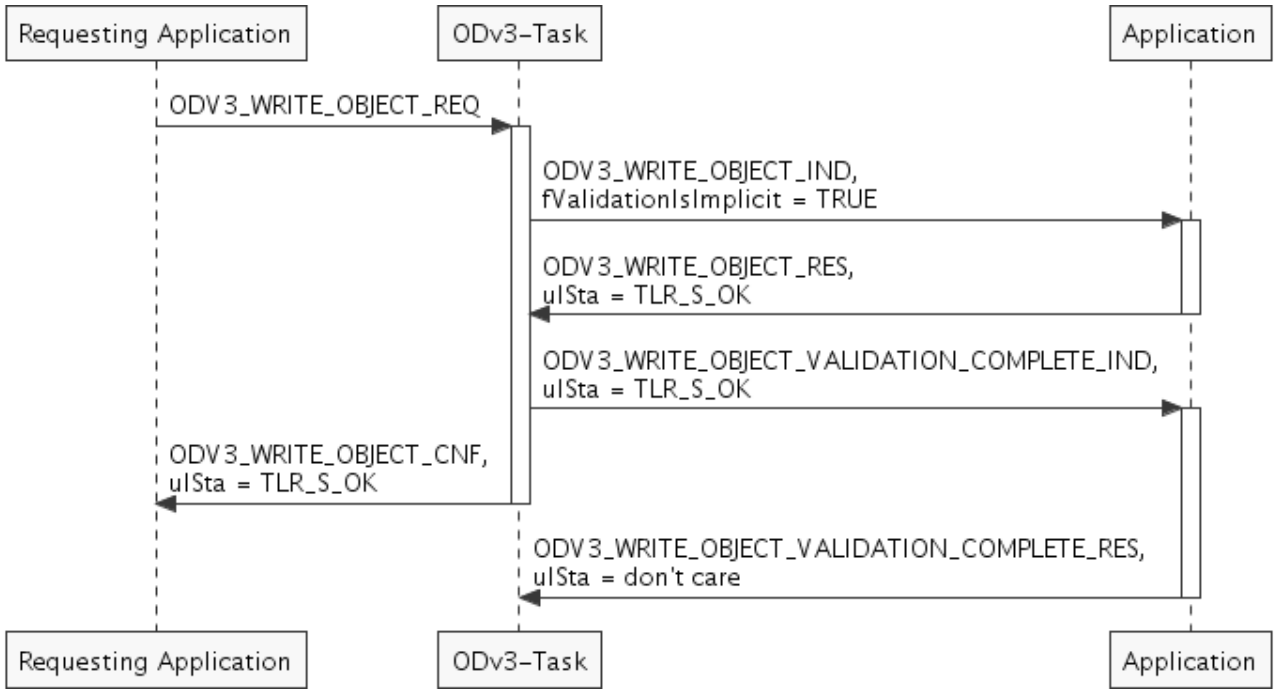

Figure 15: Sequence Diagrams - Single Application registered - Application successful

2.8.1.2 Application unsuccessful

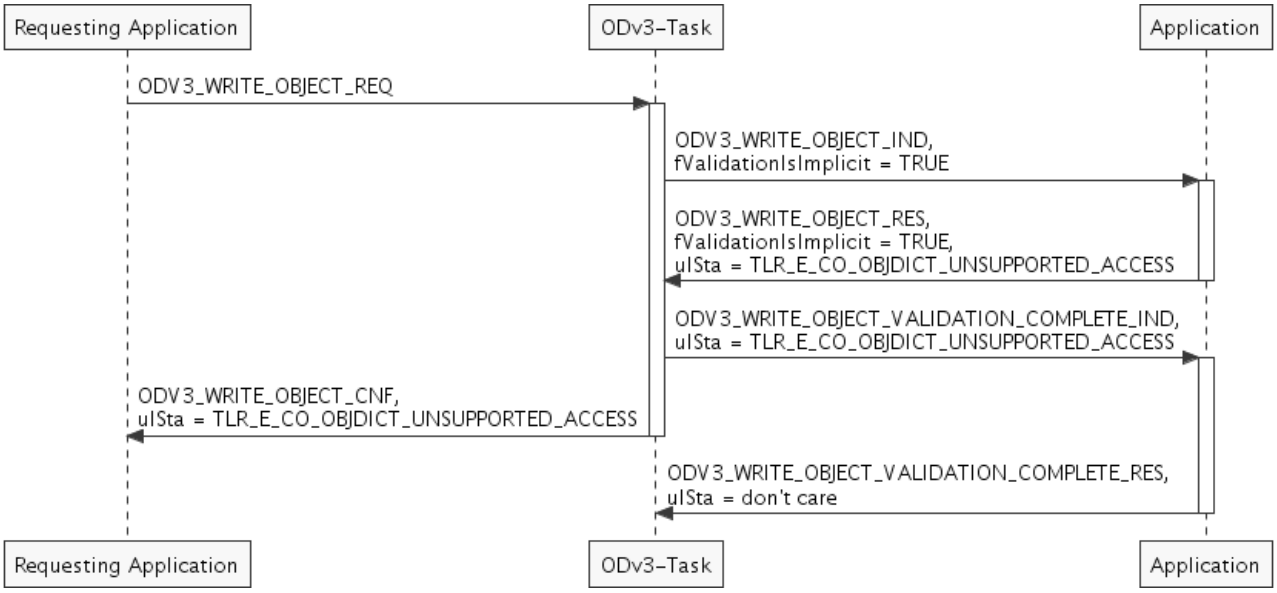

Figure 16: Sequence Diagrams - Single Application registered - Application unsuccessful

2.8.2 Multiple applications are registered for write indication

If multiple applications are registered for write indication of an object/subobject, the following diagrams apply.

2.8.2.1 All applications successful

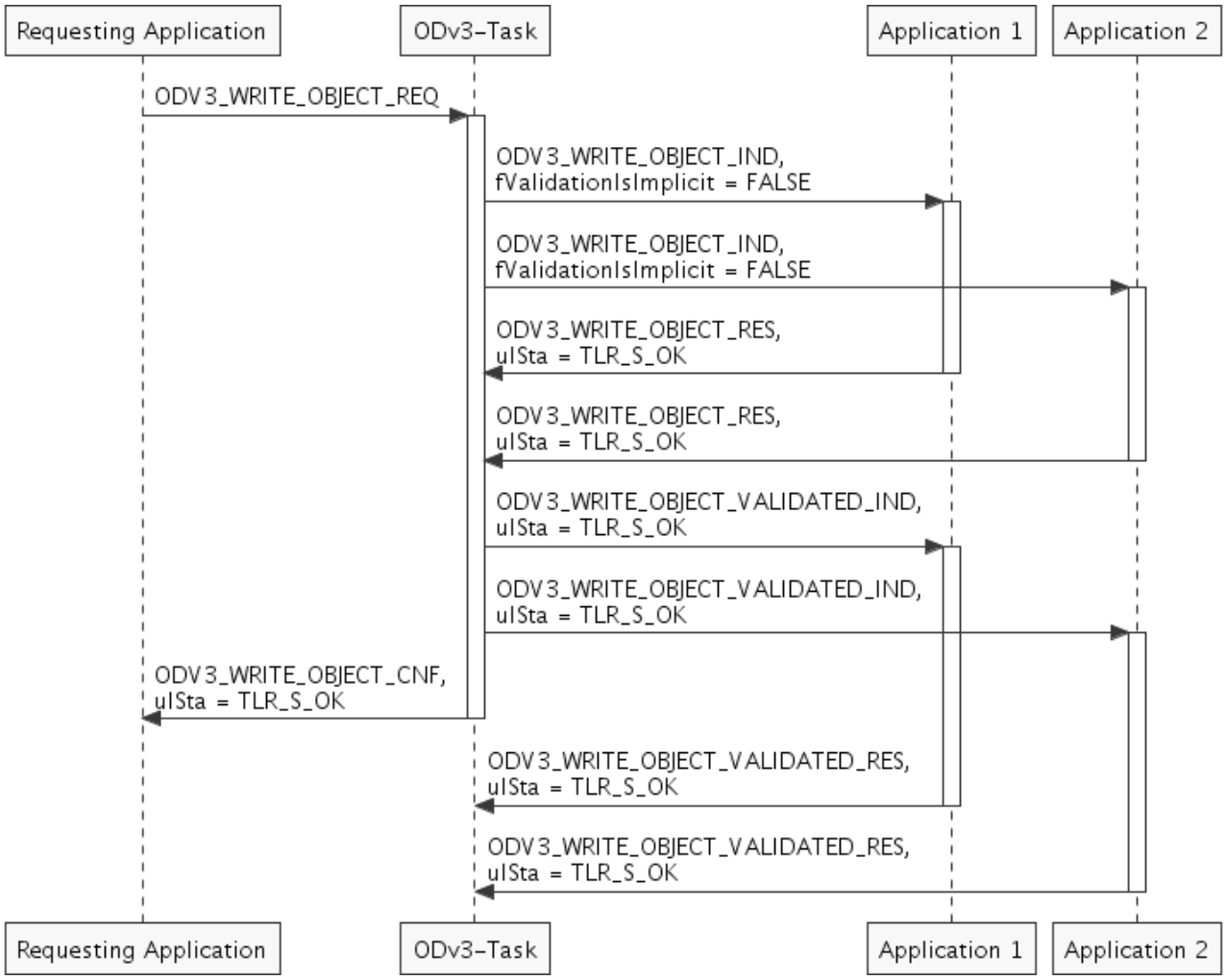

Figure 17: Sequence Diagrams - Multiple Applications registered - All Applications successful

2.8.2.2 One application unsuccessful

Application 1 is registered for `ODV3_WRITE_OBJECT_INVALIDATED_IND`.

Application 2 is not registered for `ODV3_WRITE_OBJECT_INVALIDATED_IND`.

Application 1 fails

Application 1 is returning `TLR_E_CO_OBJDICT_UNSUPPORTED_ACCESS`.

Application 2 is returning `TLR_S_OK`.

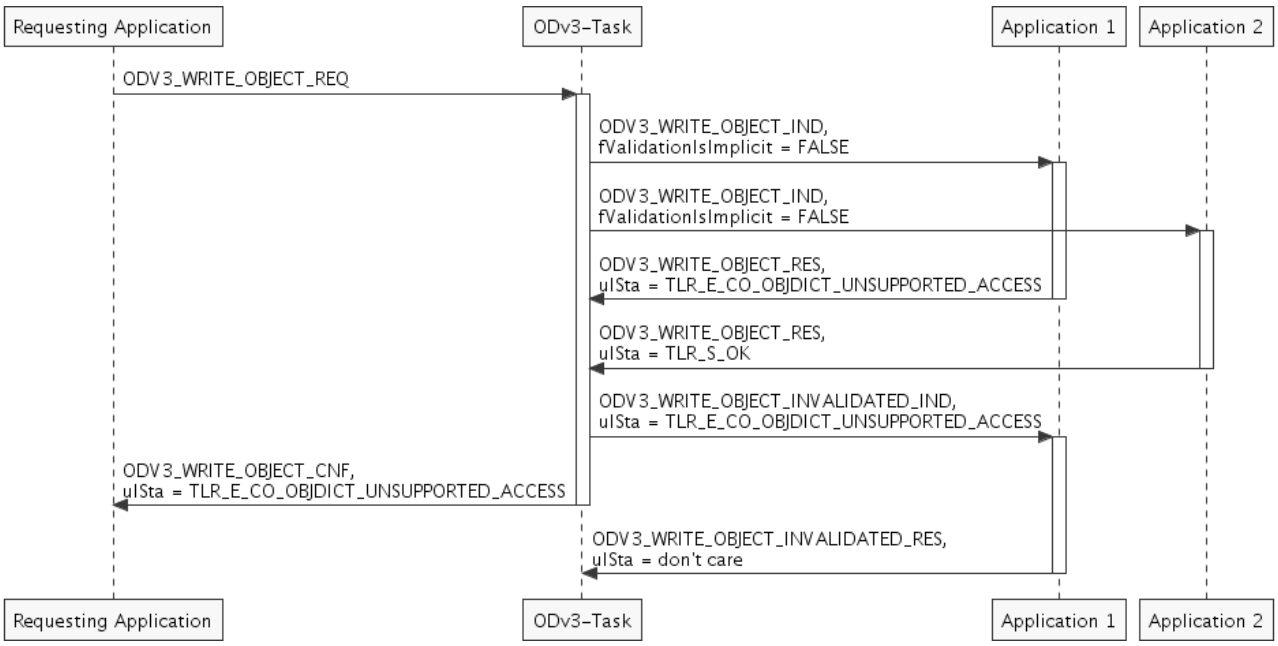

Figure 18: Sequence Diagrams - Multiple Applications registered - One application unsuccessful - Application 1 fails

Application 2 fails

Application 1 is returning `TLR_S_OK`.

Application 2 is returning `TLR_E_CO_OBJDICT_UNSUPPORTED_ACCESS`.

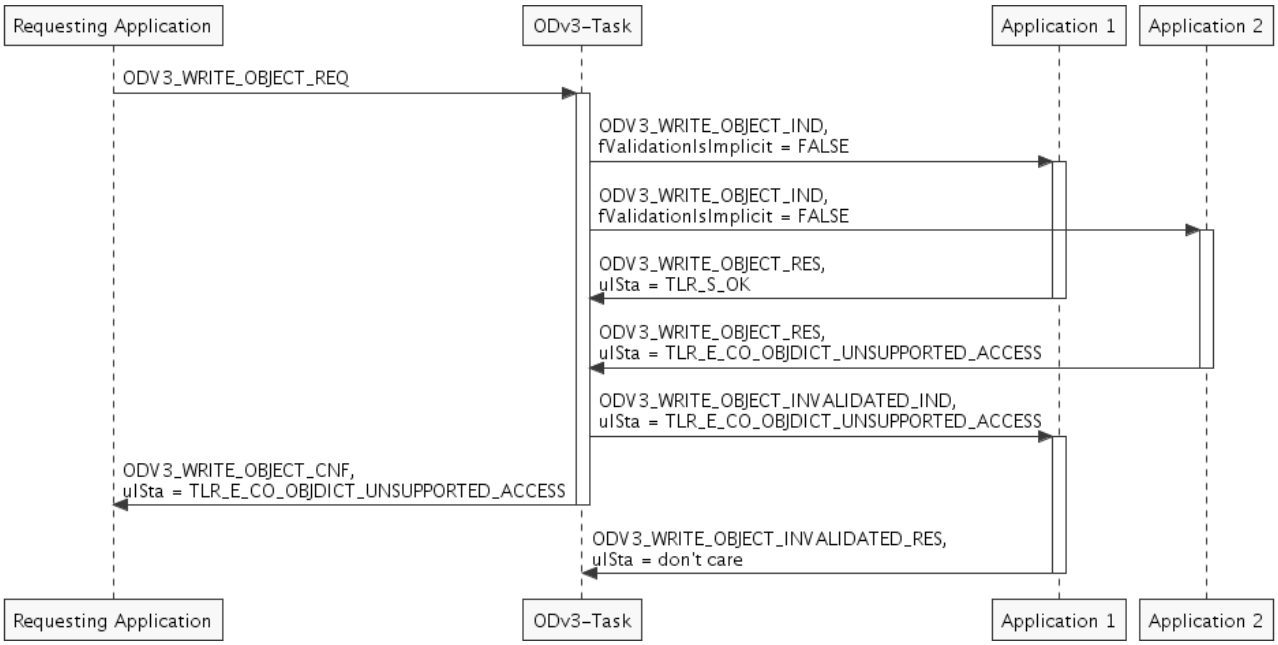

Figure 19: Sequence Diagrams - Multiple Applications registered - One application unsuccessful - Application 2 fails

2.8.2.3 Both applications unsuccessful

Application 1 is registered for ODV3_WRITE_OBJECT_INVALIDATED_IND.

Application 2 is not registered for ODV3_WRITE_OBJECT_INVALIDATED_IND.

Application 1 is returning TLR_E_CO_OBJDICT_UNSUPPORTED_ACCESS.

Application 2 is returning TLR_E_CO_OBJDICT_GENERAL_ERROR.

The first application's response will be used by the ODv3.

Note: If both applications return an error, it can result into unexpected behaviour since the error codes may change intermittently depending on the applications receiving the indications. It should **be avoided** but is listed here for showing what happens in that case

Application 1 is faster on response

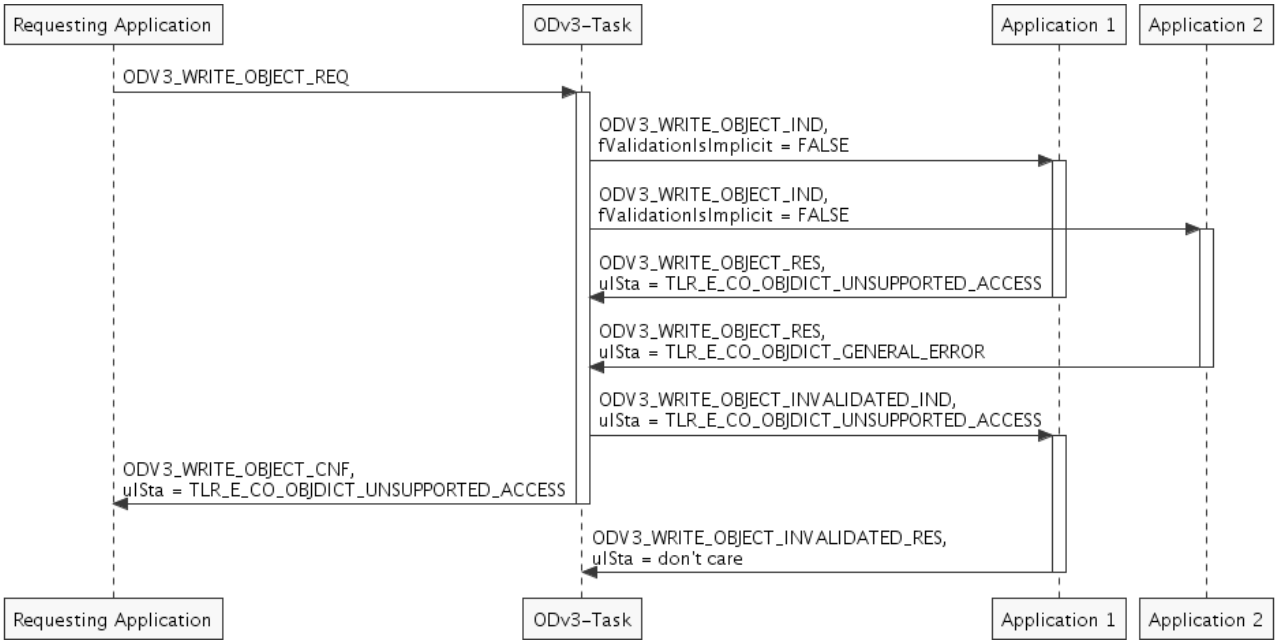

Figure 20: Sequence Diagrams - Multiple Applications registered - Both applications unsuccessful - Application 1 is faster on response

Application 2 is faster on response

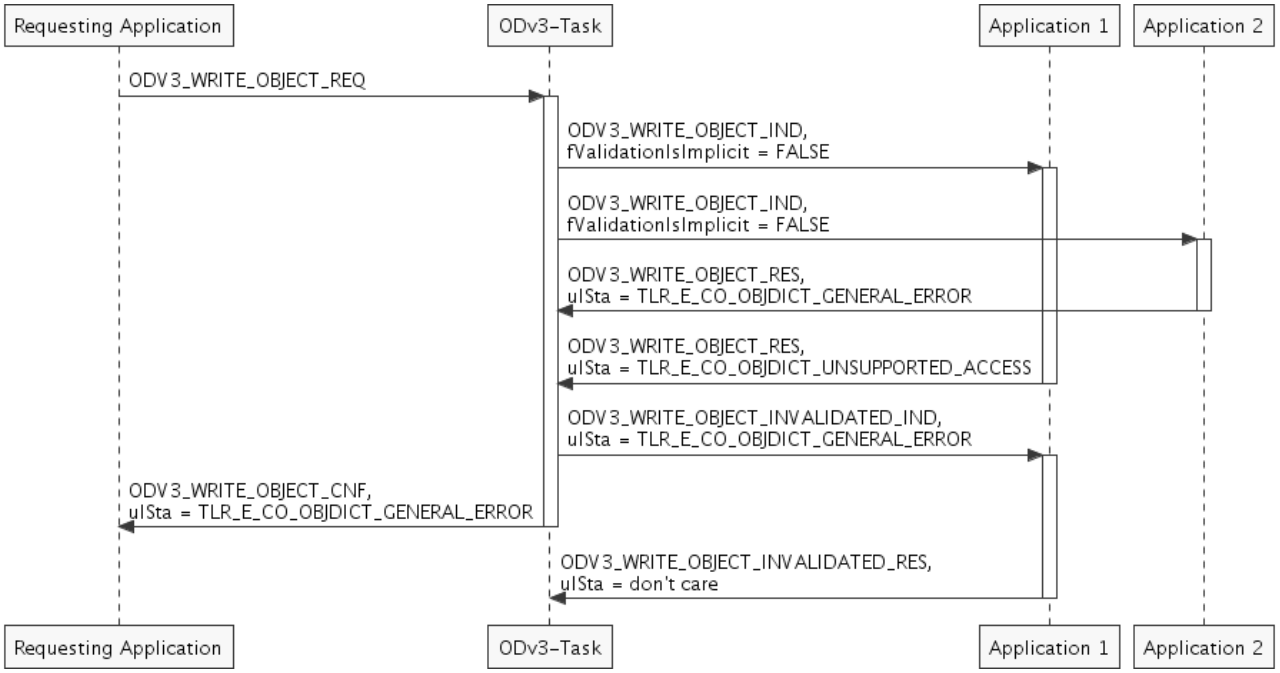

Figure 21: Sequence Diagrams - Multiple Applications registered - Both applications unsuccessful - Application 2 is faster on response

3 Basic services

This chapter describes the basic services of the object dictionary.

Table 13: Overview: Packets for basic services

3.1 ODV3_READ_OBJECT_REQ/CNF – Read Object from Dictionary

This packet is used for reading an object identified by its index (parameter `usIndex`) and subindex (parameter `bSubIndex`) from the object dictionary.

The packet supports fragmentation of read data and allows reading larger data than the actual maximum size of the packets. If the data read fits into a single segment, it will transfer the data without fragmentation.

If `usMaxSegLength` is set to a value unequal zero, it will override the MTU detection and will allow setting up custom fragmentation sizes. This is particularly useful if the accessing task needs to have larger internal packets than it is actually willing to transfer with this request.

The parameters `usIndex` and `bSubIndex` used identically to the fieldbus specification. If a simple variable is accessed, the value for `bSubIndex` is 0.

If any packet in a particular transfer contains `ulSta` unequal 0, it will abort the transfer and distribute the error accordingly.

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_READ_OBJECT_REQ/ODV3_READ_OBJECT_CNF
*
/* request packet */
typedef struct ODV3_READ_OBJECT_REQ_DATA_Ttag
{
  TLR_UINT16                                            usIndex;
  TLR_UINT8                                             bSubIndex;
  TLR_UINT16                                            usMaxSegLength;
} ODV3_READ_OBJECT_REQ_DATA_T;

typedef struct ODV3_READ_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                   tHead;
  ODV3_READ_OBJECT_REQ_DATA_T                           tData;
} ODV3_READ_OBJECT_REQ_T;
```

Packet description

Structure ODV3_READ_OBJECT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. ODv3 will allocate a fragmentation key and return it within this field on the first fragment. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A00	ODV3_READ_OBJECT_REQ – Command
ulExt	UINT32	0	Extension used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_OBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to be read
bSubIndex	UINT8	0..255	Subindex of object to be read
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes (or 0 if automatically detected MTU should be used)

Table 14: ODV3_READ_OBJECT_REQ - Read Object from Dictionary Request

Packet structure reference

```
 /* response/confirmation packet */
typedef struct ODV3_READ_OBJECT_CNF_DATA_Ttag
{
  /* unfragmentable part, part of every fragment */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT16                                          usMaxSegLength;
  TLR_UINT32                                          ulTotalDataBytes;

  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
} ODV3_READ_OBJECT_CNF_DATA_T;

typedef struct ODV3_READ_OBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_READ_OBJECT_CNF_DATA_T                         tData;
} ODV3_READ_OBJECT_CNF_T;
```

Packet description

Structure ODV3_READ_OBJECT_CNF_T			**Type:Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet ODv3 will allocate a fragmentation key and return it within this field on the first fragment.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	9+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A01	ODV3_READ_OBJECT_CNF – Command
ulExt	UINT32	0	Extension used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_OBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of read object
bSubIndex	UINT8	0..255	Subindex of read object
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes (or 0 if automatically detected MTU should be used)
ulTotalDataBytes	UINT32	0..1024	Total number of data bytes having been read over all fragments
abData[1024]	UINT8[]		Data having been read (the actual data length is determined by ulLen – 9) *If needed, this field will be fragmented.*

Table 15: ODV3_READ_OBJECT_CNF – Confirmation to Read Object from Dictionary Request

3.2 ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary

This packet is used for indicating a read of an object identified by its index (parameter `usIndex`) and subindex (parameter `bSubIndex`) from a registered application via the object dictionary. The indication is typically invoked by an ODV3_READ_OBJECT_REQ to the object dictionary where an application is registered for read indications.

The packet supports fragmentation of read data and allows reading larger data than the actual maximum size of the packets. If the data read fits into a single segment, it will transfer the data without fragmentation.

If `usMaxSegLength` is set to a value unequal zero, it will override the MTU detection and will allow setting up custom fragmentation sizes. This is particularly useful if the accessing task needs to have larger internal packets than it is actually willing to transfer with this request.

The parameters `usIndex` and `bSubIndex` used identically to the fieldbus specification. If a simple variable is accessed, the value for `bSubIndex` is 0.

If any packet in a particular transfer contains ulSta unequal 0, it will abort the transfer and distribute the error accordingly.

Packet structure reference

```
typedef struct ODV3_READ_OBJECT_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT16                                          usMaxSegLength;     /* max seg
length in bytes or 0 for field not used */
} ODV3_READ_OBJECT_IND_DATA_T;

typedef struct ODV3_READ_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_READ_OBJECT_IND_DATA_T                         tData;
} ODV3_READ_OBJECT_IND_T;
```

Packet description

Structure ODV3_READ_OBJECT_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. ODv3 will assign a fragmentation key and place it into this field. For fragmentation, see section *Fragmentation* on page 21.
ulLen	UINT32	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A00	ODV3_READ_OBJECT_IND - Command
ulExt	UINT32	0	Extension used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_OBJECT_IND_DATA_T			
usIndex	UINT16	0..65535	Index of object to be read
bSubIndex	UINT8	0..255	Subindex of object to be read
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes (or 0 if automatically detected MTU should be used by the requestee)

Table 16: ODV3_READ_OBJECT_IND - Read Object Indication

Packet structure reference

```
typedef struct ODV3_READ_OBJECT_CNF_DATA_Ttag
{
  /* unfragmentable part, part of every fragment */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT16                                          usMaxSegLength;
  TLR_UINT32                                          ulTotalDataBytes;

  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
} ODV3_READ_OBJECT_RES_DATA_T;

typedef struct ODV3_READ_OBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_READ_OBJECT_RES_DATA_T                         tData;
} ODV3_READ_OBJECT_RES_T;
```

Packet description

Structure ODV3_READ_OBJECT_CNF_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. Must be identical to indication.
ulLen	UINT32	9+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A01	ODV3_READ_OBJECT_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_OBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of read object
bSubIndex	UINT8	0..255	Subindex of read object
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes (or 0 if automatically detected MTU should be used by the requestee)
ulTotalDataBytes	UINT32	0..1024	Total number of data bytes having been read over all fragments
abData[1024]	UINT8[]		Data having been read (the actual data length is determined by ulLen – 9) *If needed, this field will be fragmented.*

Table 17: ODV3_READ_OBJECT_RES – Response to Read Object Indication

3.3 ODV3_WRITE_OBJECT_REQ/CNF – Write Object to Dictionary

This packet is used for writing an object identified by its index (parameter `usIndex`) and subindex (parameter `bSubIndex`) to the object dictionary.

The packet supports fragmentation of write data and allows writing larger data than the actual maximum size (1024) of the packets. If the data fits into a single segment, it can be transfer the data without fragmentation. The fragment data size can be kept smaller than the actual packet size.

The parameters `usIndex` and `bSubIndex` used identically to the fieldbus specification. If a simple variable is accessed, the value for `bSubIndex` is 0.

If any packet in a particular transfer contains `ulSta` unequal 0, it will abort the transfer and distribute the error accordingly.

If ulTotalDataBytes is set to `ODV3_WRITE_OBJECT_REQ_TOTAL_DATA_BYTES_NOT_SPECIFIED` (0xFFFFFFFF), the length of all fragments is not predetermined. The actual length will be extracted from the sum of all transferred data fragments.

Packet structure reference

```
typedef struct ODV3_WRITE_OBJECT_REQ_DATA_Ttag
{
  /* unfragmentable part, part of every fragment */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT32                                          ulTotalDataBytes;
  TLR_BOOLEAN32                                       fValidationIsImplicit;
  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
} ODV3_WRITE_OBJECT_REQ_DATA_T;

typedef struct ODV3_WRITE_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_OBJECT_REQ_DATA_T                        tData;
} ODV3_WRITE_OBJECT_REQ_T;

/* special value for ulTotalDataBytes */
#define ODV3_WRITE_OBJECT_REQ_TOTAL_DATA_BYTES_NOT_SPECIFIED      0xFFFFFFFF
```

Packet description

Structure ODV3_WRITE_OBJECT_REQ_T			Type: Request/Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. ODv3 will allocate a fragmentation key and return it within this field on the first fragment. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	11+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A02	ODV3_WRITE_OBJECT_REQ – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_OBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to be written
bSubIndex	UINT8	0..255	Subindex of object to be written
ulTotalDataBytes	UINT32	0..1024	Total number of data bytes to be written, needed for fragmentation support
fValidationIsImplicit	BOOLEAN32	TRUE	For request: Always set to TRUE.
abData[ ]	UINT8[]		Data to be written The actual length of the data contained is ulLen – 11. *If needed, this field will be fragmented.*

Table 18: ODV3_WRITE_OBJECT_REQ - Write Object to Dictionary Request

Packet structure reference

```
/* confirmation packet */
typedef struct ODV3_WRITE_OBJECT_CNF_DATA_Ttag
{
  /* unfragmentable part, part of every fragment */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
} ODV3_WRITE_OBJECT_CNF_DATA_T;

typedef struct ODV3_WRITE_OBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_OBJECT_CNF_DATA_T                        tData;
} ODV3_WRITE_OBJECT_CNF_T;
```

Packet description

Structure ODV3_WRITE_OBJECT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A03	ODV3_WRITE_OBJECT_CNF – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_OBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex

Table 19: ODV3_WRITE_OBJECT_CNF – Confirmation to Write Object to Dictionary Request

3.4 ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary

This packet is used for indicating a write to an object identified by its index (parameter usIndex) and subindex (parameter bSubIndex) to one or more registered applications via the object dictionary.

The packet supports fragmentation of write data and allows writing larger data than the actual maximum size of the packets. If the data fits into a single segment, it can be transfer the data without fragmentation. The fragment data size can be kept smaller than the actual packet size.

The parameters usIndex and bSubIndex used identically to the fieldbus specification. If a simple variable is accessed, the value for bSubIndex is 0.

If any packet in a particular transfer contains ulSta unequal 0, it will abort the transfer and distribute the error accordingly.

If ulTotalDataBytes is set to ODV3_WRITE_OBJECT_IND_TOTAL_DATA_BYTES_NOT_SPECIFIED (0xFFFFFFFF), the length of all fragments is not predetermined. The actual length must be extracted from the sum of all transferred data fragments.

If fValidationIsImplicit is set to TRUE, there is only one registered application for write indications. Therefore, this flag signals that the full value validation has been completed by the registered application entirely and the value can be activated immediately.

If fValidationIsImplicit is set to FALSE, there are multiple registered applications for write indications.

An ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES indication is always sent independent of the flag fValidationIsImplicit.

If all applications must accept the newly written value, the registered applications will get an ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES with ulSta = 0 and the requesting application a success response.

If at least one application denied the newly written value, all applications will get an ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES with ulSta showing the actual error code and the requesting application will get an error response.

Packet structure reference

```
typedef struct ODV3_WRITE_OBJECT_REQ_DATA_Ttag
{
  /* unfragmentable part, part of every fragment */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT32                                          ulTotalDataBytes;
  TLR_BOOLEAN32                                       fValidationIsImplicit;
  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
} ODV3_WRITE_OBJECT_IND_DATA_T;

typedef struct ODV3_WRITE_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_OBJECT_IND_DATA_T                        tData;
} ODV3_WRITE_OBJECT_IND_T;

/* special value for ulTotalDataBytes */
#define ODV3_WRITE_OBJECT_IND_TOTAL_DATA_BYTES_NOT_SPECIFIED      0xFFFFFFFF
```

Packet description

Structure ODV3_WRITE_OBJECT_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process ODv3 will assign a fragmentation key and place it into this field. For fragmentation, see section *Fragmentation* on page 21.
ulLen	UINT32	11+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A02	ODV3_WRITE_OBJECT_IND - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_OBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to be written
bSubIndex	UINT8	0..255	Subindex of object to be written
ulTotalDataBytes	UINT32	0..1024	Total number of data bytes to be written, needed for fragmentation support
fValidationIsImplicit	BOOLEAN32	TRUE, FALSE	See description of packet for details
abData[1024]	UINT8[]		Data to be written The actual length of the field is ulLen – 11. *If needed, this field will be fragmented.*

Table 20: ODV3_WRITE_OBJECT_IND - Write Object Indication

Packet structure reference

```
/* response/confirmation packet */
typedef struct ODV3_WRITE_OBJECT_CNF_DATA_Ttag
{
  /* unfragmentable part, part of every fragment */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
} ODV3_WRITE_OBJECT_RES_DATA_T;

typedef struct ODV3_WRITE_OBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_OBJECT_RES_DATA_T                        tData;
} ODV3_WRITE_OBJECT_RES_T;
```

Packet description

Structure ODV3_WRITE_OBJECT_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes*
ulCmd	UINT32	0x6A03	ODV3_WRITE_OBJECT_RES - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_OBJECT_RES_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex

Table 21: ODV3_WRITE_OBJECT_RES – Response to Write Object Indication

3.5 ODV3_GET_OBJECT_LIST_REQ/CNF – Get Object List matching the provided Access Mask

This packet is used for retrieving a list of objects matching the provided `usObjAccessMask` and `usObjAccessCompare`.

The parameters are evaluated according the following pseudo code:

```
FOREACH Object IN Objects DO
  IF (usObjAccessMask & Object->AccessMask) == usObjAccessCompare THEN
    APPEND Object TO ObjectList
  END IF
END FOREACH
```

The following parameters represent typical lists to be retrieved:

usObjAccessMask	usObjAccessCompare	Description
0x0000	0x0000	All objects The list contains all available objects
ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	Objects which are used to store settings These objects hold parameters which can be written during startup
ODV3_ACCESS_FLAGS_BACKUP (0x4000)	ODV3_ACCESS_FLAGS_BACKUP (0x4000)	Objects which are required to be stored for device replacement
ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	Objects which can be mapped into TxPDOs
ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	Objects which can be mapped into RxPDOs

Table 22: Typical parameters for usObjAccessMask and usObjAccessCompare

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_LIST_REQ_DATA_Ttag
{
  TLR_UINT16                                          usObjAccessMask;
  TLR_UINT16                                          usObjAccessCompare;
} ODV3_GET_OBJECT_LIST_REQ_DATA_T;

typedef struct ODV3_GET_OBJECT_LIST_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_LIST_REQ_DATA_T                     tData;
} ODV3_GET_OBJECT_LIST_REQ_T;
```

Packet description

Structure ODV3_GET_OBJECT_LIST_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A10	ODV3_GET_OBJECT_LIST_REQ – Command
ulExt	UINT32		Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_LIST_REQ_DATA_T			
usObjAccessMask	UINT16	0..65535 (Bit mask)	Object Access Mask
usObjAccessCompare	UINT16	0..65535 (Bit mask)	Object Access Compare Value

Table 23: ODV3_GET_OBJECT_LIST_REQ - Get Object List Request

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_LIST_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usObjAccessMask;
  TLR_UINT16                                          usObjAccessCompare;
  TLR_UINT32                                          ulTotalDataBytes;

  /* fragmentable part */
  TLR_UINT16                                          ausIndexes[512];
} ODV3_GET_OBJECT_LIST_CNF_DATA_T;

typedef struct ODV3_GET_OBJECT_LIST_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_LIST_CNF_DATA_T                     tData;
} ODV3_GET_OBJECT_LIST_CNF_T;
```

Packet description

Structure ODV3_GET_OBJECT_LIST_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	8+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A11	ODV3_GET_OBJECT_LIST_CNF – Command
ulExt	UINT32		Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_LIST_CNF_DATA_T			
usObjAccessMask	UINT16	0..65535	Object Access Mask
usObjAccessCompare	UINT16	0..65535	Object Access Compare Value
ulTotalDataBytes	UINT32		Total number of data bytes having been transferred over all fragments
ausIndexes[512]	UINT16[]		Index values having been returned *This field can be transferred in a fragmented manner.*

Table 24: ODV3_GET_OBJECT_LIST_CNF/RES - Confirmation to Get Object List Request

3.6 ODV3_GET_OBJECT_LIST_IND/RES – Get Object List matching the provided Access Mask Indication

This packet is used for retrieving a list of objects matching the provided `usObjAccessMask` and `usObjAccessCompare` from the registered application.

The parameters have to be evaluated according the following pseudo code:

```
FOREACH Object IN Objects DO
  IF (usObjAccessMask & Object->AccessMask) == usObjAccessCompare THEN
    APPEND Object TO ObjectList
  END IF
END FOREACH
```

The following parameters represent typical lists to be retrieved:

usObjAccessMask	usObjAccessCompare	Description
0x0000	0x0000	All objects The list contains all available objects
ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	Objects which are used to store settings These objects hold parameters which can be written during startup
ODV3_ACCESS_FLAGS_BACKUP (0x4000)	ODV3_ACCESS_FLAGS_BACKUP (0x4000)	Objects which are required to be stored for device replacement
ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	Objects which can be mapped into TxPDOs
ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	Objects which can be mapped into RxPDOs

Table 25: Typical parameters for usObjAccessMask and usObjAccessCompare

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_LIST_REQ_DATA_Ttag
{
  TLR_UINT16                                          usObjAccessMask;
  TLR_UINT16                                          usObjAccessCompare;
} ODV3_GET_OBJECT_LIST_IND_DATA_T;

typedef struct ODV3_GET_OBJECT_LIST_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_LIST_IND_DATA_T                     tData;
} ODV3_GET_OBJECT_LIST_IND_T;
```

Packet description

Structure ODV3_GET_OBJECT_LIST_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A10	ODV3_GET_OBJECT_LIST_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_LIST_IND_DATA_T			
usObjAccessMask	UINT16	0..65535 (Bit mask)	Object Access Mask
usObjAccessCompare	UINT16	0..65535 (Bit mask)	Object Access Compare Value

Table 26: ODV3_GET_OBJECT_LIST_IND - Get Object List Indication

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_LIST_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usObjAccessMask;
  TLR_UINT16                                          usObjAccessCompare;
  TLR_UINT32                                          ulTotalDataBytes;

  /* fragmentable part */
  TLR_UINT16                                          ausIndexes[512];
} ODV3_GET_OBJECT_LIST_RES_DATA_T;

typedef struct ODV3_GET_OBJECT_LIST_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_LIST_RES_DATA_T                     tData;
} ODV3_GET_OBJECT_LIST_RES_T;
```

Packet description

Structure ODV3_GET_OBJECT_LIST_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	8+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A11	ODV3_GET_OBJECT_LIST_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_LIST_RES_DATA_T			
usObjAccessMask	UINT16	0..65535	Object Access Mask
usObjAccessCompare	UINT16	0..65535	Object Access Compare Value
ulTotalDataBytes	UINT32		Total number of data bytes having been transferred over all fragments
ausIndexes[512]	UINT16[]		Index values having been returned This field can be transferred in a fragmented manner.

Table 27: ODV3_GET_OBJECT_LIST_RES - Response to Get Object List Indication

3.7 ODV3_GET_OBJECT_INFO_REQ/CNF – Get Object Info

This packet is used for retrieving the object description i.e. structure information of an object from the object dictionary.

An object description contains the following data:

- name of the object
- CANopen data type
- object access flags
- the maximum number of supported subindices
- object code

Object access flags (`usAccessFlags`)

The object access flags describe to what lists an object belongs. See the following table for the actual definition of the bit mask:

Bit	Name
D15	ODV3_ACCESS_FLAGS_SETTINGS If set, the object belongs to the Settings list. The list contains all objects which can be downloaded as startup parameter.
D14	ODV3_ACCESS_FLAGS_BACKUP If set, the object belongs to the Backup list. The Backup list is used for listing all objects which are needed for device replacement.
D13	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE If set, the object is listed in the list of all objects which can be mapped into TxPDOs.
D12	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE If set, the object is listed in the list of all objects which can be mapped into RxPDOs.

Table 28: Object access flags

Object code (bObjectCode)

The object code is defined as follows:

Value	Object Code (Type of accessed object)
0x00	NULL A dictionary entry with no data fields
0x02	DOMAIN Large variable amount of data e.g. executable program code
0x05	DEFTYPE Basic Type Definition
0x06	DEFSTRUCT Structure Type Definition
0x07	VAR Simple Variable
0x08	ARRAY Object with a set of subobjects of same data type
0x09	RECORD Object with a set of subobjects of any i.e. mixed data type
0x28	ENUM Enumerated definition (EtherCAT only)

Table 29: Object Code

Note: Additional object codes may exist in the protocol specification.

Maximum number of sub objects (bMaxNumOfSubObjs)

This parameter specifies the maximum number of sub objects the object can contain.

The following table shows which value ranges are used in relation to the type:

Object Type	Value range of bMaxNumOfSubObjs
Simple Variable	0
Object with subobjects	0 ... 255

Table 30: Maximum number of sub objects in relation to object type

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_INFO_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_GET_OBJECT_INFO_REQ_DATA_T;

typedef struct ODV3_GET_OBJECT_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_INFO_REQ_DATA_T                     tData;
} ODV3_GET_OBJECT_INFO_REQ_T;
```

Packet description

Structure ODV3_GET_OBJECT_INFO_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A12	ODV3_GET_OBJECT_INFO_REQ – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_INFO_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of the object whose description is to be retrieved

Table 31: ODV3_GET_OBJECT_INFO_REQ - Get Object Info Request

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_INFO_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usIndex;
  TLR_UINT16                                          usDatatype;
  TLR_UINT16                                          usAccessFlags;
  TLR_UINT8                                           bMaxNumOfSubObjs;
  TLR_UINT8                                           bObjectCode;
  TLR_UINT32                                          ulTotalDataBytes;
  /* fragmentable part */
  TLR_UINT8                                           abName[1024];
} ODV3_GET_OBJECT_INFO_CNF_DATA_T;

typedef struct ODV3_GET_OBJECT_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_INFO_CNF_DATA_T                     tData;
} ODV3_GET_OBJECT_INFO_CNF_T;
```

Packet description

Structure ODV3_GET_OBJECT_INFO_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	12+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A13	ODV3_GET_OBJECT_INFO_CNF – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_INFO_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of the object whose description is returned
usDatatype	UINT16	0x0001 … 0x0FFF	Reference to data type list given in EtherCAT Spec. Part 6, *Table 63 – Basic Data Type Area* and *Table 64 – Extended Data Type Area*
usAccessFlags	UINT16	0..65535	Access flags
bMaxNumOfSubObjs	UINT8	0..255	Maximum number of subindices of the object
bObjectCode	UINT8	7..9	Object Code, see section *Object* codes on page 14.
ulTotalDataBytes	UINT32		Total number of data bytes having been transferred over all fragments
abName[1024]	UINT8[]		Descriptive name retrieved from the object The actual length of the string is derived from ulLen – 12. It does not contain any terminating character.

Table 32: ODV3_GET_OBJECT_INFO_CNF - Confirmation to Get Object Info Request

3.8 ODV3_GET_OBJECT_INFO_IND/RES – Get Object Info

This packet is used for retrieving the object description i.e. structure information of an object from a registered application via the object dictionary.

An object description contains the following data:

- name of the object
- CANopen data type
- object access flags
- the maximum number of supported subindices
- object code

Object access flags (`usAccessFlags`)

The object access flags describe to what lists an object belongs. See the following table for the actual definition of the bit mask:

Bit	Name	Description
D15	ODV3_ACCESS_FLAGS_SETTINGS	If set, the object belongs to the Settings list. The list contains all objects which can be downloaded as startup parameter.
D14	ODV3_ACCESS_FLAGS_BACKUP	If set, the object belongs to the Backup list. The Backup list is used for listing all objects which are needed for device replacement.
D13	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE	If set, the object is listed in the list of all objects which can be mapped into TxPDOs.
D12	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE	If set, the object is listed in the list of all objects which can be mapped into RxPDOs.

Table 33: Object access flags

Object code (bObjectCode)

The object code is defined as follows:

Value	Object Code (Type of accessed object)
2	Domain
5	DEFTYPE Basic Type Definition
6	DEFSTRUCT Structure Type Definition
7	VAR Simple Variable
8	ARRAY Object with a set of subobjects of same data type
9	RECORD Object with a set of subobjects of any i.e. mixed data type

Table 34: Object Code

Note: Additional object codes may exist in protocol specification.

Maximum number of sub objects (bMaxNumOfSubObjs)

This parameter specifies the maximum number of sub objects the object can contain.

The following table shows which value ranges are used in relation to the type:

Object Type	Value range of bMaxNumOfSubObjs
Simple Variable	0
Object with subobjects	0 ... 255

Table 35: Maximum number of sub objects in relation to object type

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_INFO_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_GET_OBJECT_INFO_IND_DATA_T;

typedef struct ODV3_GET_OBJECT_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_INFO_IND_DATA_T                     tData;
} ODV3_GET_OBJECT_INFO_IND_T;
```

Packet description

Structure ODV3_GET_OBJECT_INFO_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A12	ODV3_GET_OBJECT_INFO_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_INFO_IND_DATA_T			
usIndex	UINT16	0..65535	Index of the object whose description is to be retrieved

Table 36: ODV3_GET_OBJECT_INFO_IND - Get Object Info Indication

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_INFO_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usIndex;
  TLR_UINT16                                          usDatatype;
  TLR_UINT16                                          usAccessFlags;
  TLR_UINT8                                           bMaxNumOfSubObjs;
  TLR_UINT8                                           bObjectCode;
  TLR_UINT32                                          ulTotalDataBytes;   /* needed
for fragmentation support */
  /* fragmentable part */
  TLR_UINT8                                           abName[1024];
} ODV3_GET_OBJECT_INFO_RES_DATA_T;

typedef struct ODV3_GET_OBJECT_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_INFO_RES_DATA_T                     tData;
} ODV3_GET_OBJECT_INFO_RES_T;
```

Packet description

Structure ODV3_GET_OBJECT_INFO_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	12+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A13	ODV3_GET_OBJECT_INFO_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_INFO_RES_DATA_T			
usIndex	UINT16	0..65535	Index of the object whose description is returned
usDatatype	UINT16	1..27,33..607	Reference to data type list given in EtherCAT Spec. Part 6, *Table 63 – Basic Data Type Area* and *Table 64 – Extended Data Type Area*
usAccessFlags	UINT16	0..65535	Access flags
bMaxNumOfSubObjs	UINT8	0..255	Maximum number of subindices of the object
bObjectCode	UINT8	7..9	Object Code, see section *Object* codes on page 14.
ulTotalDataBytes	UINT32		Total number of data bytes having been transferred, needed for fragmentation support
abName[1024]	UINT8[]		Descriptive name retrieved from the object The actual length of the string is derived from ulLen – 12. It does not contain any terminating character.

Table 37: ODV3_GET_OBJECT_INFO_RES - Response to Get Object Info Indication

3.9 ODV3_GET_SUBOBJECT_INFO_REQ/CNF – Get Subobject Info

This packet is used for retrieving the subobject description i.e. structure information of a subobject within an object from the object dictionary.

A subobject description contains the following data:

- unit type (EtherCAT only)
- default value
- minimum value
- maximum value
- name

bRequestedValueInfo and bValueInfo

`bRequestedValueInfo` specifies what data should be included in the confirmation. Whereas `bValueInfo` in the confirmation specifies what data is actually included.

The following table shows what bits can be selected within `bRequestedValueInfo`:

Bit	Name
D7	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MAXIMUM_VALUE Maximum value should be included in confirmation
D6	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MINIMUM_VALUE Minimum value should be included
D5	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_DEFAULT_VALUE Default value should be included
D4	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_ECAT_UNIT Unit should be included (EtherCAT only)
D0	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_NAME Name should be included

Table 38: Bit mask for bRequestedValueInfo

The following table shows what is available in the confirmation specified by `bValueInfo`:

Bit	Name
D7	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MAXIMUM_VALUE Maximum value is included in confirmation
D6	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MINIMUM_VALUE Minimum value is included in confirmation
D5	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_DEFAULT_VALUE Default value is included in confirmation
D4	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_ECAT_UNIT Unit is included in confirmation (EtherCAT only)
D0	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_NAME Name is included in confirmation

Table 39: Bit mask for bValueInfo

ulDataBitLen, usDatatype and usFieldLen

`ulDataBitLen` gives the actual length of the data contained within the subobject in total number of bits needed for the data. `usFieldLen` specifies the actual length in number of data type units. `usDatatype` refers to the actual data type whose bit length is used for a single data type unit.

`ulDataBitLen` = `usFieldLen` × data type bit length

Data field (abData)

`abData` contains the descriptive data of the subobject. This field can be transferred fragmented if its size exceeds the maximum available packet size. The included content is controlled by `bValueInfo`.

The following order is used for generating `abData` (1. and ascending):

1. unit type (EtherCAT only)
2. default value
3. minimum value
4. maximum value
5. name

Elements which are not selected within `bValueInfo` are skipped.

Packet structure reference

```
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_NAME           0x01
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_ECAT_UNIT      0x10
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_DEFAULT_VALUE  0x20
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MINIMUM_VALUE  0x40
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MAXIMUM_VALUE  0x80

typedef struct ODV3_GET_SUBOBJECT_INFO_REQ_DATA_Ttag
{
  TLR_UINT16                                                  usIndex;
  TLR_UINT8                                                   bSubIndex;
  TLR_UINT8                                                   bRequestedValueInfo;
} ODV3_GET_SUBOBJECT_INFO_REQ_DATA_T;

typedef struct ODV3_GET_SUBOBJECT_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                         tHead;
  ODV3_GET_SUBOBJECT_INFO_REQ_DATA_T                          tData;
} ODV3_GET_SUBOBJECT_INFO_REQ_T;
```

Packet description

Structure ODV3_GET_SUBOBJECT_INFO_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A14	ODV3_GET_SUBOBJECT_INFO_REQ – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_SUBOBJECT_INFO_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	SubIndex
bRequestedValueInfo	UINT8	0..255	Requested value info

Table 40: ODV3_GET_SUBOBJECT_INFO_REQ - Get Subobject Info Request

Packet structure reference

```
typedef struct ODV3_GET_SUBOBJECT_INFO_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT8                                           bValueInfo;
  TLR_UINT32                                          ulTotalDataBytes;
  TLR_UINT16                                          usAccessRights;
  TLR_UINT16                                          usDatatype;
  TLR_UINT32                                          ulDataBitLen;
  TLR_UINT16                                          usFieldLen;
  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
  /* order of fragmentable part is:
   * Unit Type, Default Value, Minimum Value, Maximum Value, Name
   */
} ODV3_GET_SUBOBJECT_INFO_CNF_DATA_T;

#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_NAME           0x01
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_ECAT_UNIT      0x10
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_DEFAULT_VALUE  0x20
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MINIMUM_VALUE  0x40
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MAXIMUM_VALUE  0x80

typedef struct ODV3_GET_SUBOBJECT_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_SUBOBJECT_INFO_CNF_DATA_T                  tData;
} ODV3_GET_SUBOBJECT_INFO_CNF_T;
```

Packet description

Structure ODV3_GET_SUBOBJECT_INFO_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	18+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A15	ODV3_GET_SUBOBJECT_INFO_CNF – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_SUBOBJECT_INFO_CNF_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex
bValueInfo	UINT8	0..255 Bit mask	Value Info
ulTotalDataBytes	UINT32		Total Data Bytes
usAccessRights	UINT16	0..65535	Access Rights
usDatatype	UINT16	0..65535	Data type
ulDataBitLen	UINT32		Data Bit Length
usFieldLen	UINT16	0..65535	Field Length (Number of data type units)
abData[1024]	UINT8[]		Data

Table 41: ODV3_GET_SUBOBJECT_INFO_CNF – Confirmation to Get Subobject Info Request

3.10 ODV3_GET_SUBOBJECT_INFO_IND/RES – Get Subobject Info Indication

This packet is used for retrieving the subobject description i.e. structure information of a subobject within an object from a registered application via the object dictionary.

A subobject description contains the following data:

- unit type (EtherCAT only)
- default value
- minimum value
- maximum value
- name

bRequestedValueInfo and bValueInfo

`bRequestedValueInfo` specifies what data should be included in the confirmation. Whereas `bValueInfo` in the confirmation specifies what data is actually included.

The following table shows what bits can be selected within `bRequestedValueInfo`:

Bit	Name and description
D7	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MAXIMUM_VALUE Maximum value should be included in confirmation
D6	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MINIMUM_VALUE Minimum value should be included
D5	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_DEFAULT_VALUE Default value should be included
D4	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_ECAT_UNIT Unit should be included (EtherCAT only)
D3	ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_NAME Name should be included

Table 42: Bit mask for bRequestedValueInfo

The following table shows what is available in the confirmation specified by `bValueInfo`:

Bit	Name and description
D7	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MAXIMUM_VALUE Maximum value is included in confirmation
D6	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MINIMUM_VALUE Minimum value is included in confirmation
D5	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_DEFAULT_VALUE Default value is included in confirmation
D4	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_ECAT_UNIT Unit is included in confirmation (EtherCAT only)
D3	ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_NAME Name is included in confirmation

Table 43: Bit mask for bValueInfo

ulDataBitLen, usDatatype and usFieldLen

`ulDataBitLen` gives the actual length of the data contained within the subobject in total number of bits needed for the data. `usFieldLen` specifies the actual length in number of data type units. `usDatatype` refers to the actual data type whose bit length is used for a single data type unit.

`ulDataBitLen` = `usFieldLen` × data type bit length

Data field (abData)

`abData` contains the descriptive data of the subobject. This field can be transferred fragmented if its size exceeds the maximum available packet size. The included content is controlled by `bValueInfo`.

The following order is used for generating `abData` (1. and ascending):

1. unit type (EtherCAT only)
2. default value
3. minimum value
4. maximum value
5. name

Elements which are not selected within `bValueInfo` are skipped.

Packet structure reference

```
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_NAME           0x01
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_ECAT_UNIT      0x10
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_DEFAULT_VALUE  0x20
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MINIMUM_VALUE  0x40
#define ODV3_GET_SUBOBJECT_INFO_REQ_VALUE_INFO_MAXIMUM_VALUE  0x80

typedef struct ODV3_GET_SUBOBJECT_INFO_REQ_DATA_Ttag
{
  TLR_UINT16                                                  usIndex;
  TLR_UINT8                                                   bSubIndex;
  TLR_UINT8                                                   bRequestedValueInfo;
} ODV3_GET_SUBOBJECT_INFO_IND_DATA_T;

typedef struct ODV3_GET_SUBOBJECT_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                         tHead;
  ODV3_GET_SUBOBJECT_INFO_IND_DATA_T                          tData;
} ODV3_GET_SUBOBJECT_INFO_IND_T;
```

Packet description

Structure ODV3_GET_SUBOBJECT_INFO_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A14	ODV3_GET_SUBOBJECT_INFO_IND - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_SUBOBJECT_INFO_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	SubIndex
bRequestedValueInfo	UINT8	0..255	Requested value info

Table 44: ODV3_GET_SUBOBJECT_INFO_IND - Get Subobject Info Indication

Packet structure reference

```
typedef struct ODV3_GET_SUBOBJECT_INFO_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT8                                           bValueInfo;
  TLR_UINT32                                          ulTotalDataBytes;
  TLR_UINT16                                          usAccessRights;
  TLR_UINT16                                          usDatatype;
  TLR_UINT32                                          ulDataBitLen;
  TLR_UINT16                                          usFieldLen;
  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
  /* order of fragmentable part is
   * Unit Type, Default Value, Minimum Value, Maximum Value, Name
   */
} ODV3_GET_SUBOBJECT_INFO_RES_DATA_T;

#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_NAME           0x01
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_ECAT_UNIT      0x10
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_DEFAULT_VALUE  0x20
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MINIMUM_VALUE  0x40
#define ODV3_GET_SUBOBJECT_INFO_CNF_VALUE_INFO_MAXIMUM_VALUE  0x80

typedef struct ODV3_GET_SUBOBJECT_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_SUBOBJECT_INFO_RES_DATA_T                  tData;
} ODV3_GET_SUBOBJECT_INFO_RES_T;
```

Packet description

Structure ODV3_GET_SUBOBJECT_INFO_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	18+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A15	ODV3_GET_SUBOBJECT_INFO_RES - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_SUBOBJECT_INFO_CNF_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex
bValueInfo	UINT8	0..255 Bit mask	Value Info
ulTotalDataBytes	UINT32		Total Data Bytes
usAccessRights	UINT16	0..65535	Access Rights
usDatatype	UINT16	0..65535	Data type
ulDataBitLen	UINT32		Data Bit Length
usFieldLen	UINT16	0..65535	Field Length (Number of data type units)
abData[1024]	UINT8[]		Data

Table 45: ODV3_GET_SUBOBJECT_INFO_RES –Response to Get Subobject Info Indication

3.11 ODV3_GET_OBJECT_ACCESS_INFO_REQ/CNF – Get Object Access Info

This packet is used for retrieving the access rights for a number of subindices of an object identified by its index (parameter `usIndex`).

The actual start subindex and number of subindices is given by `bStartSubIndex` and `bNumSubIndex`. If `bNumSubIndex` is set to 0, it is interpreted as 256.

The value of `bNumSubIndex` in the response can be smaller than in the actual request. Therefore, if the application needs the next subindices starting from the one following, it has to issue a following request starting from that subindex.

The parameter `bMaxSubIndex` of the response gives the maximum available subindices on the specified object.

The parameter `ausAccessRights[256]` in the response contains the actually retrieved access rights. The index 0 corresponds to the subindex selected by `bStartSubIndex`.

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_ACCESS_INFO_REQ_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bStartSubIndex;
  TLR_UINT8                                               bNumSubIndex; /* 0 ~ 256
entries */
} ODV3_GET_OBJECT_ACCESS_INFO_REQ_DATA_T;

typedef struct ODV3_GET_OBJECT_ACCESS_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_GET_OBJECT_ACCESS_INFO_REQ_DATA_T                  tData;
} ODV3_GET_OBJECT_ACCESS_INFO_REQ_T;
```

Packet description

Structure ODV3_GET_OBJECT_ACCESS_INFO_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A16	ODV3_GET_OBJECT_ACCESS_INFO_REQ – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_ACCESS_INFO_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
bStartSubIndex	UINT8	0..255	Start Subindex
bNumSubIndex	UINT8	0..255	Number of Subindices

Table 46: ODV3_GET_OBJECT_ACCESS_INFO_REQ - Get Object Access Info Request

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_ACCESS_INFO_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bStartSubIndex;
  TLR_UINT8                                               bNumSubIndex;
  TLR_UINT8                                               bMaxSubIndex;
  TLR_UINT8                                               abReserved[3];
  TLR_UINT16                                              ausAccessRights[256];
} ODV3_GET_OBJECT_ACCESS_INFO_CNF_DATA_T;

typedef struct ODV3_GET_OBJECT_ACCESS_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_GET_OBJECT_ACCESS_INFO_CNF_DATA_T                  tData;
} ODV3_GET_OBJECT_ACCESS_INFO_CNF_T;

#define ODV3_GET_OBJECT_ACCESS_INFO_RES_MIN_DATA_SIZE     \
                            (3 * sizeof(TLR_UINT16) + 4 * sizeof(TLR_UINT8))
```

Packet description

Structure ODV3_GET_OBJECT_ACCESS_INFO_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	8+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A17	ODV3_GET_OBJECT_ACCESS_INFO_CNF – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_ACCESS_INFO_CNF_DATA_T			
usIndex	UINT16	0..65535	Index
bStartSubIndex	UINT8	0..255	Start Subindex
bNumSubIndex	UINT8	0..255	Number of Subindices
bMaxSubIndex	UINT8	0..255	Maximum Subindex
abReserved[3]	UINT8[3]	0..255	Reserved
ausAccessRights [256]	UINT16[]		Array containing access rights of the requested range of subindices of the chosen object

Table 47: ODV3_GET_OBJECT_ACCESS_INFO_CNF - Confirmation to Get Object Access Info Request

3.12 ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info

This packet is used for retrieving the access rights for a number of subindices of an object identified by its index (parameter `usIndex`) from a registered application.

The actual start subindex and number of subindices is given by `bStartSubIndex` and `bNumSubIndex`. If `bNumSubIndex` is set to 0, it is interpreted as 256.

The value of `bNumSubIndex` in the response can be smaller than in the actual request. If done so, the object dictionary will continue with a new indication if further subindices are needed.

The parameter `bMaxSubIndex` of the response has to give the maximum available subindices on the specified object. This value is the relevant parameter for the object dictionary to determin the numer of subobjects. The maximum number of subobjects in bMaxNumOfSubObjs on object creation will be overwritten.

The parameter `ausAccessRights[256]` in the response has to contain the actually retrieved access rights. The index 0 corresponds to the subindex selected by `bStartSubIndex`.

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_ACCESS_INFO_REQ_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bStartSubIndex;
  TLR_UINT8                                               bNumSubIndex;
} ODV3_GET_OBJECT_ACCESS_INFO_IND_DATA_T;

typedef struct ODV3_GET_OBJECT_ACCESS_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_GET_OBJECT_ACCESS_INFO_IND_DATA_T                  tData;
} ODV3_GET_OBJECT_ACCESS_INFO_IND_T;
```

Packet description

Structure ODV3_GET_OBJECT_ACCESS_INFO_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A16	ODV3_GET_OBJECT_ACCESS_INFO_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_ACCESS_INFO_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
bStartSubIndex	UINT8	0..255	Start Subindex
bNumSubIndex	UINT8	0..255	Number of Subindices

Table 48: ODV3_GET_OBJECT_ACCESS_INFO_IND - Get Object Access Info Indication

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_ACCESS_INFO_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bStartSubIndex;
  TLR_UINT8                                               bNumSubIndex;
  TLR_UINT8                                               bMaxSubIndex;
  TLR_UINT8                                               bReserved;
  TLR_UINT16                                              ausAccessRights[256];
} ODV3_GET_OBJECT_ACCESS_INFO_RES_DATA_T;

typedef struct ODV3_GET_OBJECT_ACCESS_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_GET_OBJECT_ACCESS_INFO_RES_DATA_T                  tData;
} ODV3_GET_OBJECT_ACCESS_INFO_RES_T;

#define ODV3_GET_OBJECT_ACCESS_INFO_RES_MIN_DATA_SIZE     \
                           (3 * sizeof(TLR_UINT16) + 4 * sizeof(TLR_UINT8))
```

Packet description

Structure ODV3_GET_OBJECT_ACCESS_INFO_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	8+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A17	ODV3_GET_OBJECT_ACCESS_INFO_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_ACCESS_INFO_RES_DATA_T			
usIndex	UINT16	0..65535	Index
bStartSubIndex	UINT8	0..255	Start Subindex
bNumSubIndex	UINT8	0..255	Number of Subindices
bMaxSubIndex	UINT8	0..255	Maximum Subindex
bReserved	UINT8	0..255	Reserved
ausAccessRights [256]	UINT16[]		Array containing access rights of the requested range of subindices of the chosen object

Table 49: ODV3_GET_OBJECT_ACCESS_INFO_RES - Response to Get Object Access Info Indication

3.13 ODV3_GET_OBJECT_SIZE_REQ/CNF – Get Object Size

This packet is used for retrieving the subobject sizes of subobjects for a number of subindices of an object identified by its index (parameter `usIndex`).

The actual start subindex and number of subindices is given by `bStartSubIndex` and `bNumSubIndex`. If `bNumSubIndex` is set to 0, it is interpreted as 256.

The value of `bNumSubIndex` in the response can be smaller than in the actual request. Therefore, if the application needs the next subindices starting from the one following, it has to issue a following request starting from that subindex.

The parameter `bMaxSubIndex` of the response gives the maximum available subindices on the specified object.

The parameter `aulSubObjDataBitSize[256]` in the confirmation has to contain the the bit sizes of the requested subindices. The index 0 corresponds to the subindex selected by `bStartSubIndex`.

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_SIZE_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bStartSubIndex;
  TLR_UINT8                                           bNumSubIndex;
} ODV3_GET_OBJECT_SIZE_REQ_DATA_T;

typedef struct ODV3_GET_OBJECT_SIZE_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_SIZE_REQ_DATA_T                     tData;
} ODV3_GET_OBJECT_SIZE_REQ_T;
```

Packet description

Structure ODV3_GET_OBJECT_SIZE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A18	ODV3_GET_OBJECT_SIZE_REQ – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_SIZE_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
bStartSubIndex	UINT8	0..255	Start Subindex
bNumSubIndex	UINT8	0..255	Number of Subindices to be processed

Table 50: ODV3_GET_OBJECT_SIZE_REQ - Get Object Size Request/Indication

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_SIZE_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bStartSubIndex;
  TLR_UINT8                                           bNumSubIndex;
  TLR_UINT32                                          aulSubObjDataBitSize[256];
} ODV3_GET_OBJECT_SIZE_CNF_DATA_T;

typedef struct ODV3_GET_OBJECT_SIZE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_SIZE_CNF_DATA_T                     tData;
} ODV3_GET_OBJECT_SIZE_CNF_T;
#define ODV3_GET_OBJECT_SIZE_CNF_DATA_MIN_REQ_SIZE    \
                                (sizeof(TLR_UINT16) + 2 * sizeof(TLR_UINT8))
```

Packet description

Structure ODV3_GET_OBJECT_SIZE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A19	ODV3_GET_OBJECT_SIZE_CNF – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_SIZE_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of chosen object
bStartSubIndex	UINT8	0..255	Start Subindex to be applied
bNumSubIndex	UINT8	0..255	Number of Subindices
aulSubObjDataBitSize[256]	UINT32[]		Array containing subobject data bit size of requested subindices of chosen object

Table 51: ODV3_GET_OBJECT_SIZE_CNF/RES - Confirmation/Response to Get Object Size Request/Indication

3.14 ODV3_GET_OBJECT_SIZE_IND/RES – Get Object Size

This packet is used for retrieving the subobject sizes for a number of subindices of an object identified by its index (parameter `usIndex`) from a registered application.

The actual start subindex and number of subindices is given by `bStartSubIndex` and `bNumSubIndex`. If `bNumSubIndex` is set to 0, it is interpreted as 256.

The value of `bNumSubIndex` in the response can be smaller than in the actual request. If done so, the object dictionary will continue with a new indication if further subindices are needed.

The parameter `bMaxSubIndex` of the response has to give the maximum available subindices on the specified object.

The parameter `aulSubObjDataBitSize[256]` in the response has to contain the the bit sizes of the requested subindices. The index 0 corresponds to the subindex selected by `bStartSubIndex`.

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_SIZE_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bStartSubIndex;
  TLR_UINT8                                           bNumSubIndex;
} ODV3_GET_OBJECT_SIZE_IND_DATA_T;

typedef struct ODV3_GET_OBJECT_SIZE_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_SIZE_IND_DATA_T                     tData;
} ODV3_GET_OBJECT_SIZE_IND_T;
```

Packet description

Structure ODV3_GET_OBJECT_SIZE_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A18	ODV3_GET_OBJECT_SIZE_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_SIZE_IND_DATA_T			
usIndex	UINT16	0..65535	Index
bStartSubIndex	UINT8	0..255	Start Subindex
bNumSubIndex	UINT8	0..255	Number of Subindices to be processed

Table 52: ODV3_GET_OBJECT_SIZE_IND - Get Object Size Indication

Packet structure reference

```
/* response/confirmation packet */
typedef struct ODV3_GET_OBJECT_SIZE_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bStartSubIndex;
  TLR_UINT8                                               bNumSubIndex;
  TLR_UINT32                                              aulSubObjDataBitSize[256];
} ODV3_GET_OBJECT_SIZE_RES_DATA_T;

typedef struct ODV3_GET_OBJECT_SIZE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_GET_OBJECT_SIZE_RES_DATA_T                         tData;
} ODV3_GET_OBJECT_SIZE_RES_T;

#define ODV3_GET_OBJECT_SIZE_RES_DATA_MIN_REQ_SIZE        \
                                  (sizeof(TLR_UINT16) + 2 * sizeof(TLR_UINT8))
```

Packet description

Structure ODV3_GET_OBJECT_SIZE_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A19	ODV3_GET_OBJECT_SIZE_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_SIZE_RES_DATA_T			
usIndex	UINT16	0..65535	Index of chosen object
bStartSubIndex	UINT8	0..255	Start Subindex to be applied
bNumSubIndex	UINT8	0..255	Number of Subindices
aulSubObjDataBitSize[256]	UINT32[]		Array containing subobject data bit size of requested subindices of chosen object

Table 53: ODV3_GET_OBJECT_SIZE_RES - Response to Get Object Size Indication

3.15 ODV3_READ_OBJECT_NO_IND_REQ/CNF – Read Object No Indication

This packet is used for reading an object identified by its index (parameter `usIndex`) and subindex (parameter `bSubIndex`) from the object dictionary.

Note: There will be no indication being invoked with this request. Therefore, the object has to reside within the object dictionary if it is accessed by the application with this request.

The packet supports fragmentation of read data and allows reading larger data than the actual maximum size of the packets. If the data read fits into a single segment, it will transfer the data without fragmentation.

If `usMaxSegLength` is set to a value unequal zero, it will override the MTU detection and will allow setting up custom fragmentation sizes. This is particularly useful if the accessing task needs to have larger internal packets than it is actually willing to transfer with this request.

The parameters `usIndex` and `bSubIndex` are used identically to the fieldbus specification. If a simple variable is accessed, the value for `bSubIndex` is 0.

If any packet in a particular transfer contains `ulSta` unequal 0, it will abort the transfer and distribute the error accordingly.

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_READ_OBJECT_NO_IND_REQ/ODV3_READ_OBJECT_NO_IND_CNF
*/

/* request packet */
typedef struct ODV3_READ_OBJECT_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_UINT16                                          usMaxSegLength;
} ODV3_READ_OBJECT_NO_IND_REQ_DATA_T;

typedef struct ODV3_READ_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_READ_OBJECT_NO_IND_REQ_DATA_T                  tData;
} ODV3_READ_OBJECT_NO_IND_REQ_T;
```

Packet description

Structure ODV3_READ_OBJECT_NO_IND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet ODv3 will assign a fragmentation key and place it into this field. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A1C	ODV3_READ_OBJECT_NO_IND_REQ - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_OBJECT_NO_IND_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes (or 0 if automatically detected MTU should be used by the requestee)

Table 54: ODV3_READ_OBJECT_NO_IND_REQ - Read Object No Ind.Request

Packet structure reference

```
/* confirmation packet */
typedef ODV3_READ_OBJECT_CNF_DATA_T ODV3_READ_OBJECT_NO_IND_CNF_DATA_T;
typedef ODV3_READ_OBJECT_CNF_T ODV3_READ_OBJECT_NO_IND_CNF_T;
```

Packet description

Structure ODV3_READ_OBJECT_NO_IND_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process Must be identical to indication.
ulLen	UINT32	9+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A1D	ODV3_READ_OBJECT_NO_IND_CNF - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_OBJECT_NO_IND_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of read object
bSubIndex	UINT8	0..255	Subindex of read object
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes (or 0 if automatically detected MTU should be used by the requestee)
ulTotalDataBytes	UINT32	0..1024	Total number of data bytes having been read over all fragments
abData[1024]	UINT8[]		Data having been read (the actual data length is determined by ulLen – 9) *If needed, this field will be fragmented.*

Table 55: ODV3_READ_OBJECT_NO_IND_CNF – Confirmation to Read Object No Ind.Request

3.16 ODV3_GET_OBJECT_COUNT_REQ/CNF – Get Object Count

This packet is used for retrieving the number of objects belonging to the lists specified by the supplied access masks in the array `atEntries[20]`.

The request parameter `atEntries[20]` contains multiple sets of parameters to compare the object access masks with. The array element `atEntries[n]` corresponds to the confirmation parameter `ausCounts[n]`.

The parameters `usObjAccessMask` and `usObjAccessCompare` in each array element are evaluated according the following pseudo code:

```
ObjectCount = 0
FOREACH Object IN Objects DO
  IF (usObjAccessMask & Object->AccessMask) == usObjAccessCompare THEN
    ObjectCount = ObjectCount + 1
  END IF
END FOREACH
```

The following parameters represent typical lists to be retrieved:

usObjAccessMask	usObjAccessCompare	Description
0x0000	0x0000	Number of all objects The list contains all available objects
ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	Number of objects which are used to store settings These objects hold parameters which can be written during startup
ODV3_ACCESS_FLAGS_BACKUP (0x4000)	ODV3_ACCESS_FLAGS_BACKUP (0x4000)	Number of objects which are required to be stored for device replacement
ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	Number of objects which can be mapped into TxPDOs
ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	Number of objects which can be mapped into RxPDOs

Table 56: Typical parameters for usObjAccessMask and usObjAccessCompare

Packet structure reference

```
#define ODV3_GET_OBJECT_COUNT_MAX_ENTRIES     20

/* request/indication packet */
typedef struct ODV3_GET_OBJECT_COUNT_REQ_DATA_ENTRY_Ttag
{
  TLR_UINT16                                          usObjAccessMask;
  TLR_UINT16                                          usObjAccessCompare;
} ODV3_GET_OBJECT_COUNT_REQ_DATA_ENTRY_T;

typedef struct ODV3_GET_OBJECT_COUNT_REQ_DATA_Ttag
{
  ODV3_GET_OBJECT_COUNT_REQ_DATA_ENTRY_T              atEntries[20];
} ODV3_GET_OBJECT_COUNT_REQ_DATA_T;

typedef struct ODV3_GET_OBJECT_COUNT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_COUNT_REQ_DATA_T                    tData;
} ODV3_GET_OBJECT_COUNT_REQ_T;
```

Packet description

Structure ODV3_GET_OBJECT_COUNT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4 * n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A1E	ODV3_GET_OBJECT_COUNT_REQ – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_COUNT_REQ_DATA_T			
atEntries[20]	ODV3_GET_OBJECT_COUNT_REQ_DATA_ENTRY_T[]		Array containing 20 entries each of data type ODV3_GET_OBJECT_COUNT_REQ_DATA_ENTRY_T n is the actual number of entries used for requesting the data.

Table 57: ODV3_GET_OBJECT_COUNT_REQ - Get Object Count Request

Packet structure reference

```
/* response/confirmation packet */
typedef struct ODV3_GET_OBJECT_COUNT_CNF_DATA_Ttag
{
  TLR_UINT16                                          ausCounts[20];
} ODV3_GET_OBJECT_COUNT_CNF_DATA_T;

typedef struct ODV3_GET_OBJECT_COUNT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_COUNT_CNF_DATA_T                    tData;
} ODV3_GET_OBJECT_COUNT_CNF_T;
```

Packet description

Structure ODV3_GET_OBJECT_COUNT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2 * n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A1F	ODV3_GET_OBJECT_COUNT_CNF – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_COUNT_CNF_DATA_T			
ausCounts[20]	UINT16[]		Array containing the requested count values n is the actual number of entries used for requesting the data.

Table 58: ODV3_GET_OBJECT_COUNT_CNF – Confirmation/Response to Get Object Count Request

3.17 ODV3_GET_OBJECT_COUNT_IND/RES – Get Object Count

This packet is used for retrieving the number of objects belonging to the lists specified by the supplied access masks in the array `atEntries[20]` from a registered application.

The request parameter `atEntries[20]` contains multiple sets of parameters to compare the object access masks with. The array element `atEntries[n]` corresponds to the confirmation parameter `ausCounts[n]`.

The parameters `usObjAccessMask` and `usObjAccessCompare` in each array element have to be evaluated according the following pseudo code:

```
ObjectCount = 0
FOREACH Object IN Objects DO
  IF (usObjAccessMask & Object->AccessMask) == usObjAccessCompare THEN
    ObjectCount = ObjectCount + 1
  END IF
END FOREACH
```

The following parameters represent typical lists to be retrieved:

usObjAccessMask	usObjAccessCompare	Description
0x0000	0x0000	Number of all objects The list contains all available objects
ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	ODV3_ACCESS_FLAGS_SETTINGS (0x8000)	Number of objects which are used to store settings These objects hold parameters which can be written during startup
ODV3_ACCESS_FLAGS_BACKUP (0x4000)	ODV3_ACCESS_FLAGS_BACKUP (0x4000)	Number of objects which are required to be stored for device replacement
ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE (0x2000)	Number of objects which can be mapped into TxPDOs
ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE (0x1000)	Number of objects which can be mapped into RxPDOs

Table 59: Typical parameters for usObjAccessMask and usObjAccessCompare

Packet structure reference

```
#define ODV3_GET_OBJECT_COUNT_MAX_ENTRIES     20

typedef struct ODV3_GET_OBJECT_COUNT_REQ_DATA_ENTRY_Ttag
{
  TLR_UINT16                                              usObjAccessMask;
  TLR_UINT16                                              usObjAccessCompare;
} ODV3_GET_OBJECT_COUNT_IND_DATA_ENTRY_T;

typedef struct ODV3_GET_OBJECT_COUNT_REQ_DATA_Ttag
{
  ODV3_GET_OBJECT_COUNT_IND_DATA_ENTRY_T                  atEntries[20];
} ODV3_GET_OBJECT_COUNT_IND_DATA_T;

typedef struct ODV3_GET_OBJECT_COUNT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_GET_OBJECT_COUNT_IND_DATA_T                        tData;
} ODV3_GET_OBJECT_COUNT_IND_T;
```

Packet description

Structure ODV3_GET_OBJECT_COUNT_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4 * n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A1E	ODV3_GET_OBJECT_COUNT_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_COUNT_IND_DATA_T			
atEntries[20]	ODV3_GET_OBJECT_COUNT_IND_DATA_ENTRY_T[]		Array containing 20 entries each of data type ODV3_GET_OBJECT_COUNT_IND_DATA_ENTRY_T n is the actual number of entries used for requesting the data.

Table 60: ODV3_GET_OBJECT_COUNT_IND - Get Object Count Indication

Packet structure reference

```
typedef struct ODV3_GET_OBJECT_COUNT_CNF_DATA_Ttag
{
  TLR_UINT16                                          ausCounts[20];
} ODV3_GET_OBJECT_COUNT_RES_DATA_T;

typedef struct ODV3_GET_OBJECT_COUNT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_COUNT_RES_DATA_T                    tData;
} ODV3_GET_OBJECT_COUNT_RES_T;
```

Packet description

Structure ODV3_GET_OBJECT_COUNT_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2 * n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A1F	ODV3_GET_OBJECT_COUNT_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_COUNT_RES_DATA_T			
ausCounts[20]	UINT16[]		Array containing the requested count values n is the actual number of entries used for requesting the data.

Table 61: ODV3_GET_OBJECT_COUNT_RES –Response to Get Object Count Indication

3.18 ODV3_REQUEST_ABORTED_IND/RES – Request Aborted Indication

This packet is used to indicate that request has been aborted. This request may be any of the requests supporting fragmentation and all indication packets.

The error code in `ulSta` refers to the actual reason why the abort happened. Protocol stacks will translate these into the SDO Abort Code.

In the response packet, the parameter `ulSta` is not evaluated by ODv3.

The fragmentation key - to which this indication belongs - is referenced by ulSrcId.

On requests, the related fragmentation key is placed into `ulDestId` of those packets.

On indications, the related fragmentation key is placed into `ulSrcId` of those packets.

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_REQUEST_ABORTED_IND/ODV3_REQUEST_ABORTED_RES
 */

/* indication packet */
typedef struct ODV3_REQUEST_ABORTED_IND_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_REQUEST_ABORTED_IND_T;
```

Packet description

Structure ODV3_REQUEST_ABORTED_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process ODv3 has assigned a fragmentation key for the actual transfer before and will place it into this field again. For fragmentation, see section *Fragmentation* on page 21.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A2E	ODV3_REQUEST_ABORTED_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 62: ODV3_REQUEST_ABORTED_IND - Request Aborted Indication

Packet structure reference

```
/* response packet */
typedef struct ODV3_REQUEST_ABORTED_RES_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_REQUEST_ABORTED_RES_T;
```

Packet description

Structure ODV3_REQUEST_ABORTED_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process ODv3 has assigned a fragmentation key for the actual transfer before and will place it into this field again. For fragmentation, see section *Fragmentation* on page 21.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes*
ulCmd	UINT32	0x6A2F	ODV3_REQUEST_ABORTED_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 63: ODV3_REQUEST_ABORTED_RES – Response to Request Aborted Indication

3.19 ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication

This packet is used for signaling applications of the validity status of a recently transferred value with ODV3_WRITE_OBJECT_IND/RES. The field ulSrcId contains the same fragmentation key as in the ODV3_WRITE_OBJECT_IND/RES packet.

If the field fValidationIsImplicit on ODV3_WRITE_OBJECT_IND/RES is set, this indication can be ignored on simple applications. If in doubt, a user can always implement the flow based on ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES.

Packet structure reference

```
typedef struct ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
  TLR_BOOLEAN8                                            fSuccessful;
} ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_DATA_T;

typedef struct ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_DATA_T        tData;
} ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_T;
```

Packet description

Structure ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process ODv3 will assign a fragmentation key and place it into this field. For fragmentation, see section *Fragmentation* on page 21.
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		This value is TLR_S_OK (0) on success. If an error occured, the actual error code is filled in here and fSuccessful is FALSE.
ulCmd	UINT32	0x6A2A	ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex
fSuccessful	BOOL8	0 , 1	TRUE (1) if validation was successful FALSE (0) if validation was unsuccessful. The actual error code is reported in ulSta.

Table 64: ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND - Write Object Validation Complete Indication

Packet structure reference

```
/* confirmation packet */
typedef struct ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
} ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_DATA_T;

typedef struct ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_DATA_T        tData;
} ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_T;
```

Packet description

Structure ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process ODv3 will assign a fragmentation key and place it into this field. For fragmentation, see section *Fragmentation* on page 21.
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Always set to zero. The error code is not evaluated by ODv3.
ulCmd	UINT32	0x6A2B	ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing, do not touch
tData - Structure ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex

Table 65: ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_RES Response to Write Object Validation Complete Indication

3.20 ODV3_GET_OBJECT_PROPERTIES_IND/RES – Get Object Properties

This packet allows a registered application to answer incoming requests for the object properties of a specific object identified by its index (parameter `usIndex`). The application sets the `ulFlags` parameter of the response packet accordingly. The possible values for the `ulFlags` parameter are given in the following table:

Bit	Properties Flag and description
D2	ODV3_OBJECT_PROP_FLAG_WRITE_SI0_TO_0_FIRST_REQUIRED 0 = no 1 = yes
D1	ODV3_OBJECT_PROP_FLAG_IS_INDEXED 0 = simple object variable. 1 = object is not a simple variable.
D0	ODV3_OBJECT_PROP_FLAG_ALL_INDEXES_ARE_FIXED_SIZE 0 = no 1 = all indexes are fixed size: Object requires `ODV3_OBJECT_PROP_FLAG_WRITE_SI0_TO_0_FIRST_REQUIRED`. Setting this bit will set the actual transfer control bit (Necessary bit for using EtherCAT Complete Access).

Table 66: Possible values for the `ulFlags` parameter

The `usIndex` parameter of the response packet should be set to the value of the `usIndex` parameter of the indication packet.

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_GET_OBJECT_PROPERTIES_IND/ODV3_GET_OBJECT_PROPERTIES_RES
 *
 */

/* indication packet */

typedef struct ODV3_GET_OBJECT_PROPERTIES_IND_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_GET_OBJECT_PROPERTIES_IND_DATA_T;

typedef struct ODV3_GET_OBJECT_PROPERTIES_IND_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_GET_OBJECT_PROPERTIES_IND_DATA_T               tData;
} ODV3_GET_OBJECT_PROPERTIES_IND_T;
```

Packet description

Structure ODV3_GET_OBJECT_PROPERTIES_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A30	ODV3_GET_OBJECT_PROPERTIES_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_PROPERTIES_IND_DATA_T			
usIndex	UINT16	0..65535	Index of object whose properties have been requested

Table 67: ODV3_GET_OBJECT_PROPERTIES_IND - Get Object Properties Indication

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_GET_OBJECT_PROPERTIES_IND/ODV3_GET_OBJECT_PROPERTIES_RES
 *
 */

#define ODV3_OBJECT_PROP_FLAG_ALL_INDEXES_ARE_FIXED_SIZE    0x00000001
#define ODV3_OBJECT_PROP_FLAG_IS_INDEXED                    0x00000002
/* is not a simple variable */
#define ODV3_OBJECT_PROP_FLAG_WRITE_SI0_TO_0_FIRST_REQUIRED 0x00000004

/* response packet */
typedef struct ODV3_GET_OBJECT_PROPERTIES_RES_DATA_Ttag
{
  TLR_UINT16                                            usIndex;
  TLR_UINT32                                            ulFlags;
} ODV3_GET_OBJECT_PROPERTIES_RES_DATA_T;

typedef struct ODV3_GET_OBJECT_PROPERTIES_RES_Ttag
{
  TLR_PACKET_HEADER_T                                   tHead;
  ODV3_GET_OBJECT_PROPERTIES_RES_DATA_T                 tData;
} ODV3_GET_OBJECT_PROPERTIES_RES_T;
```

Packet description

Structure ODV3_GET_OBJECT_SIZE_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	6	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section „Status/Error Codes”
ulCmd	UINT32	0x6A31	ODV3_GET_OBJECT_PROPERTIES_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_GET_OBJECT_SIZE_RES_DATA_T			
usIndex	UINT16	0..65535	Index of requested object
ulFlags	UINT32	Bit mask	Property flag

Table 68: ODV3_GET_OBJECT_PROPERTIES_RES - Response to Get Object Properties Indication

4 Compound services

This chapter describes the compound services of the object dictionary.

No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
4.1	ODV3_WRITE_ALL_BY_INDEX_REQ/CNF – Write All by Index	0x6A20/ 0x6A21	106
4.2	ODV3_READ_ALL_BY_INDEX_REQ/CNF – Read All by Index	0x6A22/ 0x6A23	110
4.3	ODV3_RESET_OBJECTS_REQ/CNF – Set objects to their default values	0x6A28/ 0x6A29	114
4.4	ODV3_RESET_OBJECTS_IND/RES – Reset objects to their default values	0x6A28/ 0x6A29	117

Table 69: Overview: Packets for compound services

4.1 ODV3_WRITE_ALL_BY_INDEX_REQ/CNF – Write All by Index

This packet is used for writing an object identified by its index (parameter `usIndex`) and subindex (parameter `bSubIndex`) to the object dictionary.

The packet supports fragmentation of write data and allows writing larger data than the actual maximum size of the packets. If the data fits into a single segment, it can transfer the data without fragmentation. The fragment data size can be kept smaller than the actual packet size.

The parameters `usIndex` and `bSubIndex` used identically to the fieldbus specification. If a simple variable is accessed, the value for `bSubIndex` is 0.

If any packet in a particular transfer contains `ulSta` unequal 0, this will abort the transfer and distribute the error accordingly.

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_WRITE_ALL_BY_INDEX_REQ
 *
 * Fragmentation:
 *                            Request/Indication  Response/Confirmation
 * TLR_PACKET_SEQ_FIRST       ulDestId = 0        ulDestId = X
 * TLR_PACKET_SEQ_MIDDLE      ulDestId = X        ulDestId == X
 * TLR_PACKET_SEQ_LAST        ulDestId = X        ulDestId == X
 *
 * X is a handle to the running transfer
 *
*/

/* request packet */
typedef struct ODV3_WRITE_ALL_BY_INDEX_REQ_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bStartSubIndex; /* 0 or 1 valid
*/
  TLR_UINT8                                           bMultipleParaAccessFlags; /*
access flags for multiple parameter access */
  TLR_UINT32                                          ulTotalDataBytes;
  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
} ODV3_WRITE_ALL_BY_INDEX_REQ_DATA_T;

typedef struct ODV3_WRITE_ALL_BY_INDEX_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_ALL_BY_INDEX_REQ_DATA_T                  tData;
} ODV3_WRITE_ALL_BY_INDEX_REQ_T;

/* special value for ulTotalDataBytes */
#define ODV3_WRITE_ALL_BY_INDEX_REQ_TOTAL_DATA_BYTES_NOT_SPECIFIED      0xFFFFFFFF
```

Packet description

Structure ODV3_WRITE_ALL_BY_INDEX_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. ODv3 will allocate a fragmentation key and return it within this field on the first fragment. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	8+n	Packet Data Length in bytes (n= ulTotalDataBytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort!)
ulCmd	UINT32	0x6A20	ODV3_WRITE_ALL_BY_INDEX_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_ALL_BY_INDEX_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to be written
bStartSubIndex	UINT8	0..1	Minimum value of subindex to be used
bMultipleParaAccessFlags	UINT8	Bit mask	Access flags for multiple parameter access
ulTotalDataBytes	UINT32	0.. 1024	Total number of data bytes
abData[1024]	UINT8[]		Data

Table 70: ODV3_WRITE_ALL_BY_INDEX_REQ - Write All by Index Request

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_WRITE_ALL_BY_INDEX_CNF
 *
 * Fragmentation:
 *                             Request/Indication  Response/Confirmation
 * TLR_PACKET_SEQ_FIRST        ulDestId = 0        ulDestId = X
 * TLR_PACKET_SEQ_MIDDLE       ulDestId = X        ulDestId == X
 * TLR_PACKET_SEQ_LAST         ulDestId = X        ulDestId == X
 *
 * X is a handle to the running transfer
 *
*/

/* confirmation packet */
typedef struct ODV3_WRITE_ALL_BY_INDEX_CNF_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bStartSubIndex; /* 0 or 1 valid
*/
} ODV3_WRITE_ALL_BY_INDEX_CNF_DATA_T;

typedef struct ODV3_WRITE_ALL_BY_INDEX_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_ALL_BY_INDEX_CNF_DATA_T                  tData;
} ODV3_WRITE_ALL_BY_INDEX_CNF_T;


/* packet union */
typedef union ODV3_WRITE_ALL_BY_INDEX_PCK_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_WRITE_ALL_BY_INDEX_REQ_T                       tReq;
  ODV3_WRITE_ALL_BY_INDEX_CNF_T                       tCnf;
} ODV3_WRITE_ALL_BY_INDEX_PCK_T;
```

Packet description

Structure ODV3_WRITE_ALL_BY_INDEX_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A21	ODV3_WRITE_ALL_BY_INDEX_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_WRITE_ALL_BY_INDEX_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of object to be written
bStartSubIndex	UINT8	0..1	Minimum value of subindex to be used

Table 71: ODV3_WRITE_ALL_BY_INDEX_CNF - Confirmation to Write All by Index Request

4.2 ODV3_READ_ALL_BY_INDEX_REQ/CNF – Read All by Index

This packet is used for reading an object identified by its index (parameter `usIndex`) and subindex (parameter `bSubIndex`) from the object dictionary.

The packet supports fragmentation of read data and allows reading larger data than the actual maximum size of the packets. If the data read fits into a single segment, it will transfer the data without fragmentation.

The parameters `usIndex` and `bSubIndex` used identically to the fieldbus specification. If a simple variable is accessed, the value for `bSubIndex` is 0.

If any packet in a particular transfer contains `ulSta` unequal 0, it will abort the transfer and distribute the error accordingly.

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_READ_ALL_BY_INDEX_REQ
 *
 * Fragmentation:
 *                          Request/Indication  Response/Confirmation
 * TLR_PACKET_SEQ_FIRST     ulDestId = 0        ulDestId = X
 * TLR_PACKET_SEQ_MIDDLE    ulDestId = X        ulDestId == X
 * TLR_PACKET_SEQ_LAST      ulDestId = X        ulDestId == X
 *
 * X is a handle to the running transfer
 *
*/

/* request packet */
typedef struct ODV3_READ_ALL_BY_INDEX_REQ_DATA_Ttag
{
  /* unfragmentable part */
  /* first five bytes must be identical to ODV3_READ_OBJECT_REQ_DATA_T */
  TLR_UINT16 usIndex;
  TLR_UINT8  bStartSubIndex; /* 0 or 1 valid */
  TLR_UINT8  bMultipleParaAccessFlags; /* access flags for multiple parameter access */
  TLR_UINT16 usMaxSegLength;     /* max seg length in bytes or 0 for field not used */
} ODV3_READ_ALL_BY_INDEX_REQ_DATA_T;

typedef struct ODV3_READ_ALL_BY_INDEX_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  ODV3_READ_ALL_BY_INDEX_REQ_DATA_T                  tData;
} ODV3_READ_ALL_BY_INDEX_REQ_T;

/* packet union */
typedef union ODV3_READ_ALL_BY_INDEX_PCK_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  ODV3_READ_ALL_BY_INDEX_REQ_T                       tReq;
  ODV3_READ_ALL_BY_INDEX_CNF_T                       tCnf;
} ODV3_READ_ALL_BY_INDEX_PCK_T;
```

Packet description

Structure ODV3_READ_ALL_BY_INDEX_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. ODv3 will allocate a fragmentation key and return it within this field on the first fragment. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	6	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort!)
ulCmd	UINT32	0x6A22	ODV3_READ_ALL_BY_INDEX_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_ALL_BY_INDEX_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to be written
bStartSubIndex	UINT8	0..1	Minimum value of subindex to be used
bMultipleParaAccessFlags	UINT8	Bit mask	Access flags for multiple parameter access
usMaxSegLength	UINT16	0..65535	Maximum segment length in bytes or 0 for field not used

Table 72: ODV3_READ_ALL_BY_INDEX_REQ - Read All by Index Request

Packet structure reference

```
/*****************************************************************************
 * Packet:  ODV3_READ_ALL_BY_INDEX_CNF
 *
 * Fragmentation:
 *                             Request/Indication  Response/Confirmation
 * TLR_PACKET_SEQ_FIRST        ulDestId = 0        ulDestId = X
 * TLR_PACKET_SEQ_MIDDLE       ulDestId = X        ulDestId == X
 * TLR_PACKET_SEQ_LAST         ulDestId = X        ulDestId == X
 *
 * X is a handle to the running transfer
 *
*/

/* confirmation packet */
typedef struct ODV3_READ_ALL_BY_INDEX_CNF_DATA_Ttag
{
  /* unfragmentable part (must be identical to ODV3_READ_OBJECT_CNF_DATA_T) */
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bStartSubIndex; /* 0 or 1 valid
*/
  TLR_UINT32                                          ulTotalDataBytes;

  /* fragmentable part */
  TLR_UINT8                                           abData[1024];
} ODV3_READ_ALL_BY_INDEX_CNF_DATA_T;

typedef struct ODV3_READ_ALL_BY_INDEX_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_READ_ALL_BY_INDEX_CNF_DATA_T                   tData;
} ODV3_READ_ALL_BY_INDEX_CNF_T;

/* packet union */
typedef union ODV3_READ_ALL_BY_INDEX_PCK_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_READ_ALL_BY_INDEX_REQ_T                        tReq;
  ODV3_READ_ALL_BY_INDEX_CNF_T                        tCnf;
} ODV3_READ_ALL_BY_INDEX_PCK_T;
```

Packet description

Structure ODV3_READ_ALL_BY_INDEX_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet ODv3 will allocate a fragmentation key and return it within this field on the first fragment.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	7+n	Packet Data Length in bytes (n= ulTotalDataBytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A23	ODV3_READ_ALL_BY_INDEX_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_READ_ALL_BY_INDEX_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of object to be written
bStartSubIndex	UINT8	0..1	Minimum value of subindex to be used
ulTotalDataBytes	UINT32	0.. 1024	Total number of data bytes
abData[1024]	UINT8[]		Data

Table 73: ODV3_READ_ALL_BY_INDEX_CNF - Confirmation to Read All by Index Request

4.3 ODV3_RESET_OBJECTS_REQ/CNF – Set objects to their default values

This packet is used for setting the objects to their default values within the object dictionary.

Note: This packet is not accessible with all protocol stacks since it can be locked to ensure the operating condition of the actual protocol stack.

The field `atEntries` in the packet contains an array of description telling what range of objects to reset to default values. The parameter supports multiple entries which must be sorted in ascending order based on `usRangeStart`. If multiple entries of `atEntries` are used, their `usRangeStart` must be in ascending order.

Structure `ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_T`		
Variable	**Value**	**Description**
usRangeStart	0x0000 … 0xFFFF	Specifies on which index to start
usRangeCount	0x0000 … 0xFFFF	Specifies how many index to do from start (0 corresponds to 65536 objects)

Table 74: ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_T structure

Packet structure reference

```
typedef struct ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_Ttag
{
  TLR_UINT16                                          usRangeStart;
  TLR_UINT16                                          usRangeCount;
} ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_T;

typedef struct ODV3_RESET_OBJECTS_REQ_DATA_Ttag
{
  ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_T              atEntries[1];
} ODV3_RESET_OBJECTS_REQ_DATA_T;

typedef struct ODV3_RESET_OBJECTS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_RESET_OBJECTS_REQ_DATA_T                       tData;
} ODV3_RESET_OBJECTS_REQ_T;
```

Packet description

Structure ODV3_RESET_OBJECTS_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. ODv3 will allocate a fragmentation key and return it within this field on the first fragment. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	n*4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A28	ODV3_RESET_OBJECTS_REQ – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_RESET_OBJECTS_REQ_DATA_T			
atEntries[1]	ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_T[]		Array of range entries defining what to include in reset to default values

Table 75: ODV3_RESET_OBJECTS_REQ - Reset Objects to Default Values Request

Packet structure reference

```
typedef struct ODV3_RESET_OBJECTS_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_RESET_OBJECTS_CNF_T;
```

Packet description

Structure ODV3_RESET_OBJECTS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A29	ODV3_RESET_OBJECTS_CNF – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch

Table 76: ODV3_RESET_OBJECTS_CNF – Confirmation to Reset Objects to Default Values Request

4.4 ODV3_RESET_OBJECTS_IND/RES – Reset objects to their default values

This packet is used for notifying all applications registered for indication about what objects should be reset to their default values.

Note: This indication is sent only once per registered application. It is **not** repeated for every object.

The field `atEntries` in the packet contains an array of description telling what range of objects to reset to default values. The parameter supports multiple entries which must be sorted in ascending order based on `usRangeStart`. If multiple entries of `atEntries` are used, their `usRangeStart` must be in ascending order.

Structure `ODV3_RESET_OBJECTS_IND_RANGE_ENTRIES_T`		
Variable	**Value**	**Description**
usRangeStart	0x0000 … 0xFFFF	Specifies on which index to start
usRangeCount	0x0000 … 0xFFFF	Specifies how many index to do from start (0 corresponds to 65536 objects)

Table 77: ODV3_RESET_OBJECTS_IND_RANGE_ENTRIES_T structure

Packet structure reference

```
/* request packet */
typedef struct ODV3_RESET_OBJECTS_REQ_RANGE_ENTRIES_Ttag
{
  TLR_UINT16                                              usRangeStart;
  TLR_UINT16                                              usRangeCount;
} ODV3_RESET_OBJECTS_IND_RANGE_ENTRIES_T;

typedef struct ODV3_RESET_OBJECTS_REQ_DATA_Ttag
{
  ODV3_RESET_OBJECTS_IND_RANGE_ENTRIES_T                  atEntries[1];
} ODV3_RESET_OBJECTS_IND_DATA_T;

typedef struct ODV3_RESET_OBJECTS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_RESET_OBJECTS_IND_DATA_T                           tData;
} ODV3_RESET_OBJECTS_IND_T;
```

Packet description

Structure ODV3_RESET_OBJECTS_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. ODv3 will allocate a fragmentation key and return it within this field on the first fragment. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	n*4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A28	ODV3_RESET_OBJECTS_IND – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_RESET_OBJECTS_IND_DATA_T			
atEntries[1]	ODV3_RESET_OBJECTS_IND_RANGE_ENTRIES_T[]		Array of range entries defining what to include in reset to default values

Table 78: ODV3_RESET_OBJECTS_IND - Reset Objects to Default Values Indication

Packet structure reference

```
typedef struct ODV3_RESET_OBJECTS_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_RESET_OBJECTS_RES_T;
```

Packet description

Structure ODV3_RESET_OBJECTS_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A29	ODV3_RESET_OBJECTS_RES – Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch

Table 79: ODV3_RESET_OBJECTS_RES – Response to Reset Objects to Default Values Indication

5 Management services

This chapter describes the management services of the object dictionary.

No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
5.1	ODV3_CREATE_OBJECT_REQ/CNF – Create Object	0x6A80/ 0x6A81	120
5.2	ODV3_CREATE_SUBOBJECT_REQ/CNF – Create Subobject	0x6A82/ 0x6A83	132
5.3	ODV3_DELETE_OBJECT_REQ/CNF – Delete Object	0x6A84/ 0x6A85	139
5.4	ODV3_DELETE_SUBOBJECT_REQ /CNF – Delete Subobject	0x6A86/ 0x6A87	141
5.5	ODV3_REGISTER_OBJECT_NOTIFY_REQ/CNF – Register Object Notify	0x6A90/ 0x6A91	143
5.6	ODV3_UNREGISTER_OBJECT_NOTIFY_REQ/CNF – Unregister Object Notify	0x6A92/ 0x6A93	146
5.7	ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ /CNF – Register Subobject Notify	0x6A94/ 0x6A95	149
5.8	ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ/CNF – Unregister Subobject Notify	0x6A96/ 0x6A97	152
5.9	ODV3_REGISTER_UNDEFINED_NOTIFY_REQ/CNF – Register Undefined Notify	0x6AA0/ 0x6AA1	155
5.10	ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ/CNF – Unregister Undefined Notify	0x6AA2/ 0x6AA3	157
5.11	ODV3_REGISTER_OBJINFO_NOTIFY_REQ/CNF – Register Object Info Notify	0x6AA4/ 0x6AA5	159
5.12	ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ/CNF – Unregister Object Info Notify	0x6AA6/ 0x6AA7	161
5.13	ODV3_LOCK_OBJECT_DELETION_REQ/CNF – Lock Object Deletion	0x6AB0/ 0x6AB1	163
5.14	ODV3_UNLOCK_OBJECT_DELETION_REQ/CNF – Unlock Object Deletion	0x6AB2/ 0x6AB3	165
5.15	ODV3_SET_OBJECT_NAME_REQ/CNF – Set Object Name	0x6AB4/ 0x6AB5	167
5.16	ODV3_SET_SUBOBJECT_NAME_REQ/CNF – Set Subobject Name	0x6AB6/ 0x6AB7	169
5.17	ODV3_CREATE_DATATYPE_REQ/CNF – Create Data Type	0x6AC0/ 0x6AC1	171
5.18	ODV3_DELETE_DATATYPE_REQ/CNF – Delete Data Type	0x6AC2/ 0x6AC3	174

Table 80: Overview: Packets for management services

5.1 ODV3_CREATE_OBJECT_REQ/CNF – Create Object

This packet is used for creating a new object in the object dictionary. The service has two different types of operation:

- Creating an object containing a simple variable
- Creating an object with subindices

The index parameter `usIndex` specifies the index of the object to be created in the object dictionary.

Maximum number of Subobjects (`bMaxNumOfSubObjs`)

`bMaxNumOfSubObjs` specifies the number of subobjects the object should have.

The following rules apply for it:

Value	ODV3_ACCESS_FLAGS_FORCE_INDEXED set?	Result of operation
0	No	Simple Variable Subindex 0 will be created during the object creation.
0	Yes	Object with subobjects only element 0 exists
n	Do not care	Only valid with subobjects n elements exists after sub index 0 Subobjects must be created afterwards using ODV3_CREATE_SUBOBJECT_REQ/CNF – Create Subobject.

Table 81: Rules for `bMaxNumOfSubObjs` during Object Creation

Object code (bObjectCode)

The object code is defined as follows:

Value	Object Code (Type of accessed object)
0x00	NULL A dictionary entry with no data fields
0x02	DOMAIN Large variable amount of data e.g. executable program code
0x05	DEFTYPE Basic Type Definition
0x06	DEFSTRUCT Structure Type Definition
0x07	VAR Simple Variable
0x08	ARRAY Object with a set of subobjects of same data type
0x09	RECORD Object with a set of subobjects of any i.e. mixed data type
0x28	ENUM Enumerated definition (EtherCAT only)

Table 82: Object Code

Note: Additional object codes may exist in protocol specification.

Object access flags (usAccessFlags)

The object access flags describe to what lists an object belongs. See the following table for the actual definition of the bit mask:

Bit	Name and description
D15	ODV3_ACCESS_FLAGS_SETTINGS If set, the object belongs to the Settings list. The list contains all objects which can be downloaded as startup parameter.
D14	ODV3_ACCESS_FLAGS_BACKUP If set, the object belongs to the Backup list. The Backup list is used for listing all objects which are needed for device replacement.
D13	ODV3_ACCESS_FLAGS_TXPDO_MAPPABLE If set, the object is listed in the list of all objects which can be mapped into TxPDOs.
D12	ODV3_ACCESS_FLAGS_RXPDO_MAPPABLE If set, the object is listed in the list of all objects which can be mapped into RxPDOs.
D11	ODV3_ACCESS_FLAGS_FORCE_INDEXED If set during creation, the object will be created as an object with subobjects if `bMaxNumOfSubObjs` is set to 0.
D10	ODV3_ACCESS_FLAGS_CREATE_SUBINDEX_0 If set during creation of an object with subobjects, the subindex 0 will be created with data type 0x0005: UNSIGNED8. Its value has to be given with initial value. (it´s not set by bMaxNumOfSubobj).
D9	If set during creation of an object with subobjects, the subindex 0 will be written first. (only on indexed objects, uses simple var `bValueInfo` fields (uses data type ODV3_DATATYPE_UNSIGNED8))

Table 83: Object Access Flags during object creation

Value Info (bValueInfo)

bValueInfo specifies which fields are provided in abData[1024] and provides additional options on creation.

The following bits are defined for it:

Bit	Namea and description
D7	ODV3_VALUE_INFO_VARIABLE_SIZE_VALUE Only on VAR or Subindex 0 created: the data container will allow shorter data than specified by ulMaxFieldUnits. Only set if needed, otherwise one could e.g.write 2 bit to an UINT32 value. EtherCAT Complete Access will be denied when this bit is set.
D6	ODV3_VALUE_INFO_MAXIMUM If set, a maximum value data block is provided in abData. Otherwise, VAR or subindex 0 will not have a maximum value.
D5	ODV3_VALUE_INFO_MINIMUM If set, a minimum value data block is provided in abData. Otherwise, VAR or subindex 0 will not have a minimum value.
D4	ODV3_VALUE_INFO_DEFAULT_VALUE If set, a default value data block is provided in abData. Otherwise, VAR or subindex 0 will not have a default value.
D3	ODV3_VALUE_INFO_ECAT_UNIT Only used on EtherCAT: If set, an EtherCAT unit data block is provided in abData. Otherwise, VAR or subindex 0 will not have a unit.
D2	ODV3_VALUE_INFO_VIRTUAL If set, the VAR or subindex 0 will not have an associated data storage. Therefore, an application must register for read and write. Cannot be combined with ODV3_VALUE_INFO_INITIAL_VALUE
D1	ODV3_VALUE_INFO_INITIAL_VALUE If set, an initial value data block is provided in abData. Cannot be combined with ODV3_VALUE_INFO_VIRTUAL If neither ODV3_VALUE_INFO_INITIAL_VALUE nor ODV3_VALUE_INFO_VIRTUAL are set, the object is cleared to 0.
D0	ODV3_VALUE_INFO_NAME If set, a name data block is provided in abData. Otherwise, the object will have an empty name.

Table 84: Value Info Flags during Object Creation

Note: If ODV3_VALUE_INFO_VIRTUAL is used, it is necessary to register for read and write indications. Otherwise, any read or write to the object will result into an error.

Indication Registration Flags (`bIndicationFlags`)

`bIndicationFlags` controls whether specific indication options will be registered immediately on object creation. The parameters are also applied to the subobjects of the object. The following options exist for indication registration:

Bit	Name and description
D4	ODV3_INDICATION_FLAGS_ON_WRITE_INVALIDATED If this bit is set and multiple indication receivers are registered for write, the application registering will get the following indication: ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication. This bit must be combined with ODV3_INDICATION_FLAGS_ON_WRITE.
D3	ODV3_INDICATION_FLAGS_ON_INFO_UNDEFINED_SUBOBJ If this bit is set and a non-existing sub index is accessed, the application registering will get the following services: ODV3_GET_SUBOBJECT_INFO_IND/RES – Get Subobject Info Indication
D2	ODV3_INDICATION_FLAGS_ON_RW_UNDEFINED_SUBOBJ If this bit is set and a non-existing sub index is accessed, the application registering will get the following services: ▪ ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary ▪ ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary ▪ ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info ▪ ODV3_GET_OBJECT_SIZE_IND/RES – Get Object Size (Only send when Complete Acces is used)
D1	ODV3_INDICATION_FLAGS_ON_WRITE If this bit is set and a write to any subindex is executed, the application registering will get the following services: ▪ ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary ▪ ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication for any subobject not being registered on subobject-level.
D0	ODV3_INDICATION_FLAGS_ON_READ If this bit is set and a read of any subindex is executed, the application registering will get the following services if no one is registered on the subobject level: ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary

Table 85: Indication Flags during object creation

Note: If registered for write indications on object level, an additional registration on subobject level will result into two indications being sent to the application.

Multiple Parameter Read/Write Access Flags

The following parameters govern multiple parameter read/write access:

Bit	Name and description
D5	ODV3_MULTIPLE_PARA_ACCESS_CHECK_ACCESS_RIGHTS Reserved for future: first UINT16 in data contains access rights to be checked.
D4	ODV3_MULTIPLE_PARA_ACCESS_EVAL_SUBINDEX_0_ON_WRITE On writing all by indexes, subindex 0 will be evaluated if transferred.
D3	ODV3_MULTIPLE_PARA_ACCESS_SUBINDEX_0_PADDED_TO_U16 On reading and writing all indexes, the subindex 0 is padded to 16bit integer (when object is not SimpleVar).
D2	ODV3_MULTIPLE_PARA_ACCESS_WRITE_SI0_TO_0_FIRST Write subindex 0 to zero first and final value as last transfer, object properties can enable it without having been set here.
D1	ODV3_MULTIPLE_PARA_ACCESS_BIT_TYPES_BIT_ALIGNED Access to bit types will be placed at bit boundaries.
D0	ODV3_MULTIPLE_PARA_ACCESS_ALL_INDEXES Access all index starting at bSubIndex (0 or 1 allowed) (reserved for read/write multiple parameter).

Table 86: Multiple Parameter Read/Write Access Flags

Data type (`usDatatype`)

The field `usDatatype` specifies the actual data type to be set on the object. If a variable (VAR) is created, the subindex 0 will be assigned with the same data type.

If an object with subobjects is created, the data type is only assigned to the object. The data type is not passed into any subobject.

Access rights on VAR or subindex 0 creation (`usAccessRights`)

`usAccessRights` specifies the read/write access rights. The actual bit definition is protocol dependent (see 2.6).

Maximum field units on VAR or subindex 0 creation (usMaxFieldUnits)

ulMaxFieldUnits specifies the maximum number of data type units subindex 0 can contain if created. If ODV3_VALUE_INFO_VARIABLE_SIZE_VALUE is not set, ulMaxFieldUnits specifies the exact number of data type units subindex 0 will contain:

The following table shows the use cases for ulMaxFieldUnits:

CANopen data type	usDatatype	ODV3_VALUE_INFO_VARIABLE_SIZE_VALUE set in bValueInfo?	ulMaxFieldUnits
UNSIGNED32	0x0007	No	1
VISIBLE_STRING	0x0009	No	Specifies actual string length
VISIBLE_STRING	0x0009	Yes	Specifies max string length but actual string can be shorter

Table 87: Meaning of ulMaxFieldUnits during object creation

Fragmentable Data field (abData)

abData contains the additional data to be used for creation. The content is controlled by bValueInfo. In any packet, the actual length of the data contained within abData is ulLen – 20. ulTotalDataBytes refers to the entire data transfers i.e. the summed data length of all data fragments.

The following order from 1. upwards is filled into abData:

1. If ODV3_VALUE_INFO_ECAT_UNIT is set, the following structure is added (only EtherCAT)

Data Type	Value	Description
UINT32		For details see ETG.1004 EtherCAT Unit Specification

Table 88: abData content for EtherCAT unit

2. If ODV3_VALUE_INFO_NAME is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of name
UINT8[n]	String	Name to be used to create the object with

Table 89: abData content for Name

3. If ODV3_VALUE_INFO_MINIMUM is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of minimum value
UINT8[n]	Minimum value data	Minimum Value Data to be used to create the object with

Table 90: abData content for Minimum Value

4. If ODV3_VALUE_INFO_MAXIMUM is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of maximum value
UINT8[n]	Maximum value data	Maximum Value Data to be used to create the object with

Table 91: abData content for Maximum Value

5. If ODV3_VALUE_INFO_INITIAL_VALUE is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of initial value
UINT8[n]	Initial value data	Initial Value Data to be used to create the object with

Table 92: abData content for Initial Value

6. If ODV3_VALUE_INFO_DEFAULT_VALUE is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of default value
UINT8[n]	Default value data	Default Value Data to be used to create the object with

Table 93: abData content for Default Value

Packet structure reference

```
typedef struct ODV3_CREATE_OBJECT_REQ_DATA_Ttag
{
  /* non-fragmentable part */
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bMaxNumOfSubObjs;
  TLR_UINT8                                               bObjectCode;
  TLR_UINT16                                              usAccessFlags;
  TLR_UINT8                                               bValueInfo;
  TLR_UINT8                                               bIndicationFlags;
  TLR_UINT16                                              usDatatype;
  /* following two are only evaluated on SimpleVar { */
  TLR_UINT16                                              usAccessRights;
  TLR_UINT32                                              ulMaxFieldUnits;
  /* } */
  TLR_UINT32                                              ulTotalDataBytes;
  /* fragmentable part */
  /* bValueInfo determines what gets appended here */
  TLR_UINT8                                               abData[1024];
  /* order of fields: (no padding between)
   * - Unit (TLR_UINT32)
   * - Name (TLR_UINT32 ulNameLength,
   *         TLR_STR abName[ulNameLength])
   * - Minimum Value (TLR_UINT32 ulMinimumValueLength,
   *   TLR_UINT8 abData[ulMinimumValueLength])
   * - Maximum Value (TLR_UINT32 ulMaximumValueLength,
   *   TLR_UINT8 abData[ulMaximumValueLength])
   * - Initial Value (TLR_UINT32 ulInitialValueLength,
   *   TLR_UINT8 abData[ulInitialValueLength])
   * - Default Value (TLR_UINT32 ulDefaultValueLength,
   *   TLR_UINT8 abData[ulDefaultValueLength])
   */
} ODV3_CREATE_OBJECT_REQ_DATA_T;

typedef struct ODV3_CREATE_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_CREATE_OBJECT_REQ_DATA_T                           tData;
} ODV3_CREATE_OBJECT_REQ_T;
```

Packet description

Structure ODV3_CREATE_OBJECT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32		Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A80	ODV3_CREATE_OBJECT_REQ - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_CREATE_OBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object container to be created
bMaxNumOfSubObjs	UINT8	0..255	Maximum number of subobjects to be contained in object container to be created.
bObjectCode	UINT8	0..255	Object code
usAccessFlags	UINT16	0..65535	Access flags
bValueInfo	UINT8	0..255	Value Info
bIndicationFlags	UINT8	0..255	Indication flags
usDatatype	UINT16	0..65535	Data type
usAccessRights	UINT16	0..65535	Access rights, see section *2.5 “Object access masks”*
ulMaxFieldUnits	UINT32		Maximum field units
ulTotalDataBytes	UINT32		Total number of data bytes, needed for fragmentation support
abData[1024]	UINT8[]		Object data For structure of these data see above.

Table 94: ODV3_CREATE_OBJECT_REQ - Create Object Request

Packet structure reference

```
typedef struct ODV3_CREATE_OBJECT_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_CREATE_OBJECT_CNF_DATA_T;

typedef struct ODV3_CREATE_OBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_CREATE_OBJECT_CNF_DATA_T                       tData;
} ODV3_CREATE_OBJECT_CNF_T;
```

Packet description

Structure ODV3_CREATE_OBJECT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A81	ODV3_CREATE_OBJECT_CNF - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_CREATE_OBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index

Table 95: ODV3_CREATE_OBJECT_CNF - Confirmation to Create Object Request

5.2 ODV3_CREATE_SUBOBJECT_REQ/CNF – Create Subobject

This packet is used for creating a new subobject on an existing object in the object dictionary.

Value Info (bValueInfo)

bValueInfo specifies which fields are provided in abData[1024] and provides additional options on creation.

The following bits are defined for it:

Bit	Name and description
D7	ODV3_VALUE_INFO_VARIABLE_SIZE_VALUE Allows shorter data than specified by ulMaxFieldUnits
D6	ODV3_VALUE_INFO_MAXIMUM If set, a maximum value data block is provided in abData. Otherwise, the subindex will not have a maximum value.
D5	ODV3_VALUE_INFO_MINIMUM If set, a minimum value data block is provided in abData. Otherwise, the subindex will not have a minimum value.
D4	ODV3_VALUE_INFO_DEFAULT_VALUE If set, a default value data block is provided in abData. Otherwise, the subindex will not have a default value.
D3	ODV3_VALUE_INFO_ECAT_UNIT Only used on EtherCAT: If set, an EtherCAT unit data block is provided in abData. Otherwise, the subindex will not have a unit.
D2	ODV3_VALUE_INFO_VIRTUAL If set, the subindex will not have an associated data storage. Therefore, an application must register for read and write. Cannot be combined with ODV3_VALUE_INFO_INITIAL_VALUE
D1	ODV3_VALUE_INFO_INITIAL_VALUE If set, an initial value data block is provided in abData. Cannot be combined with ODV3_VALUE_INFO_VIRTUAL. If neither ODV3_VALUE_INFO_INITIAL_VALUE nor ODV3_VALUE_INFO_VIRTUAL are set, the object is cleared to 0.
D0	ODV3_VALUE_INFO_NAME If set, a name data block is provided in abData. Otherwise, the object will have an empty name.

Table 96: Value Info Flags during subobject creation

Note: If ODV3_VALUE_INFO_VIRTUAL is used, it is necessary to register for read and write indications. Otherwise, any read or write to the subobject will result into an error.

Indication Registration Flags (`bIndicationFlags`)

`bIndicationFlags` controls whether specific indication options will be registered immediately on object creation.

The following options exist for indication registration:

Bit	Name and description
D4	ODV3_INDICATION_FLAGS_ON_WRITE_INVALIDATED If this bit is set and multiple indication receivers are registered for write, the application registering will get the following indication: ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication This bit must be combined with ODV3_INDICATION_FLAGS_ON_WRITE.
D1	ODV3_INDICATION_FLAGS_ON_WRITE If this bit is set and a write to any subindex is executed, the application registering will get the following services: ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indicationfor any subobject not being registered on subobject-level.
D0	ODV3_INDICATION_FLAGS_ON_READ If this bit is set and a read of any subindex is executed, the application registering will get the following services if no one is registered on the subobject level: ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary

Table 97: Indication Flags during object creation

Data type (`usDatatype`)

The field `usDatatype` specifies the actual data type to be set on the object. If a variable (VAR) is created, the subindex 0 will be assigned with the same data type.

If an object with subobjects is created, the data type is only assigned to the object. The data type is not passed into any subobject.

Access rights on VAR or subindex 0 creation (`usAccessRights`)

`usAccessRights` specifies the read/write access rights. The actual bit definition is protocol dependent.

Maximum field units on VAR or subindex 0 creation (usMaxFieldUnits)

ulMaxFieldUnits specifies the maximum number of data type units subindex 0 can contain if created. If DV3_VALUE_INFO_VARIABLE_SIZE_VALUE is not set, ulMaxFieldUnits specifies the exact number of data type units subindex 0 will contain:

The following table shows the use cases for ulMaxFieldUnits:

CANopen data type	usDatatype	ODV3_VALUE_INFO_VARIABLE_SIZE_VALUE set in bValueInfo?	ulMaxFieldUnits
UNSIGNED32	0x0007	No	1
VISIBLE_STRING	0x0009	No	Specifies actual string length
VISIBLE_STRING	0x0009	Yes	Specifies max string length but actual string can be shorter

Table 98: Meaning of ulMaxFieldUnits during object creation

Fragmentable Data field (abData)

abData contains the additional data to be used for creation. The content is controlled by bValueInfo. In any packet, the actual length of the data contained within abData is ulLen – 20. ulTotalDataBytes refers to the entire data transfers i.e. the summed data length of all data fragments.

The following order from 1. upwards is filled into abData:

1. If ODV3_VALUE_INFO_ECAT_UNIT is set, the following structure is added (only EtherCAT)

Data Type	Value	Description
UINT32		For details see ETG.1004 EtherCAT Unit Specification

Table 99: abData content for EtherCAT unit

2. If ODV3_VALUE_INFO_NAME is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of name
UINT8[n]	String	Name to be used to create the object with

Table 100: abData content for Name

3. If ODV3_VALUE_INFO_MINIMUM is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of minimum value
UINT8[n]	Minimum value data	Minimum Value Data to be used to create the object with

Table 101: abData content for Minimum Value

4. If ODV3_VALUE_INFO_MAXIMUM is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of maximum value
UINT8[n]	Maximum value data	Maximum Value Data to be used to create the object with

Table 102: abData content for Maximum Value

5. If ODV3_VALUE_INFO_INITIAL_VALUE is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of initial value
UINT8[n]	Initial value data	Initial Value Data to be used to create the object with

Table 103: abData content for Initial Value

6. If ODV3_VALUE_INFO_DEFAULT_VALUE is set, the following structure is added

Data Type	Value	Description
UINT32	n	Length of default value
UINT8[n]	Default value data	Default Value Data to be used to create the object with

Table 104: abData content for Default Value

Packet structure reference

```
typedef struct ODV3_CREATE_SUBOBJECT_REQ_DATA_Ttag
{
  /* non-fragmentable part */
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
  TLR_UINT8                                               bValueInfo;
  TLR_UINT8                                               bIndicationFlags;
  TLR_UINT16                                              usAccessRights;
  TLR_UINT16                                              usDatatype;
  TLR_UINT32                                              ulMaxFieldUnits;
  TLR_UINT32                                              ulTotalDataBytes;
  /* fragmentable part */
  TLR_UINT8                                               abData[1024];
  /* order of fields: (no padding between)
   * - Unit (TLR_UINT32)
   * - Name (TLR_UINT32 ulNameLength,
   *         TLR_STR abName[ulNameLength]
   * - Minimum Value (TLR_UINT32 ulMinimumValueLength,
   *                  TLR_UINT8 abData[ulMinimumValueLength])
   * - Maximum Value (TLR_UINT32 ulMaximumValueLength,
   *                  TLR_UINT8 abData[ulMaximumValueLength])
   * - Initial Value (TLR_UINT32 ulInitialValueLength,
   *                  TLR_UINT8 abData[ulInitialValueLength])
   * - Default Value (TLR_UINT32 ulDefaultValueLength,
   *                  TLR_UINT8 abData[ulDefaultValueLength])
   */
} ODV3_CREATE_SUBOBJECT_REQ_DATA_T;

typedef struct ODV3_CREATE_SUBOBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_CREATE_SUBOBJECT_REQ_DATA_T                        tData;
} ODV3_CREATE_SUBOBJECT_REQ_T;
```

Packet description

Structure ODV3_CREATE_SUBOBJECT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet. For fragmentation, see section *Fragmentation* on page 21.
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	17+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A82	ODV3_CREATE_SUBOBJECT_REQ - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_CREATE_SUBOBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to be extended with new subobject
bSubIndex	UINT8	0..255	Subindex of newly created subobject
bValueInfo	UINT8	0..255	Value Info
bIndicationFlags	UINT8	0..255	Indication flags
usAccessRights	UINT16	0..65535	Access rights
usDatatype	UINT16	0..65535	Data type
ulMaxFieldUnits	UINT32		Maximum field units
ulTotalDataBytes	UINT32		Total number of data bytes, needed for fragmentation support
abData[1024]	UINT8[]		Subobject data. For structure of these data see above.

Table 105: ODV3_CREATE_SUBOBJECT_REQ - Create Subobject Request

Packet structure reference

```
typedef struct ODV3_CREATE_SUBOBJECT_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
} ODV3_CREATE_SUBOBJECT_CNF_DATA_T;

typedef struct ODV3_CREATE_SUBOBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_CREATE_SUBOBJECT_CNF_DATA_T                    tData;
} ODV3_CREATE_SUBOBJECT_CNF_T;
```

Packet description

Structure ODV3_CREATE_SUBOBJECT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A83	ODV3_CREATE_SUBOBJECT_CNF - Command
ulExt	UINT32	0	Used for fragmentation purposes
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_CREATE_SUBOBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of object extended with new subobject
bSubIndex	UINT8	0..255	Subindex of newly created subobject

Table 106: ODV3_CREATE_SUBOBJECT_CNF - Confirmation to Create Subobject Request

5.3 ODV3_DELETE_OBJECT_REQ/CNF – Delete Object

This packet is used for deletion (i.e. erasing) an existing object including its subobjects. The object is specified with the parameter `usIndex`.

Packet structure reference

```
typedef struct ODV3_DELETE_OBJECT_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_DELETE_OBJECT_REQ_DATA_T;

typedef struct ODV3_DELETE_OBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_DELETE_OBJECT_REQ_DATA_T                       tData;
} ODV3_DELETE_OBJECT_REQ_T;
```

Packet description

Structure ODV3_DELETE_OBJECT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A84	ODV3_DELETE_OBJECT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_DELETE_OBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of the object to be deleted including subobjects.

Table 107: ODV3_DELETE_OBJECT_REQ - Delete Object Request

Packet structure reference

```
typedef struct ODV3_DELETE_OBJECT_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_DELETE_OBJECT_CNF_DATA_T;

typedef struct ODV3_DELETE_OBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_DELETE_OBJECT_CNF_DATA_T                       tData;
} ODV3_DELETE_OBJECT_CNF_T;
```

Packet description

Structure ODV3_DELETE_OBJECT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A85	ODV3_DELETE_OBJECT_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_DELETE_OBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of the object having been deleted.

Table 108: ODV3_DELETE_OBJECT_CNF - Confirmation to Delete Object Request

5.4 ODV3_DELETE_SUBOBJECT_REQ /CNF – Delete Subobject

This packet is used for deletion (i.e. erasing) an existing subobject within an object of the object dictionary. The subobject is specified by `usIndex` and `bSubIndex`.

Packet structure reference

```
typedef struct ODV3_DELETE_SUBOBJECT_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
} ODV3_DELETE_SUBOBJECT_REQ_DATA_T;

typedef struct ODV3_DELETE_SUBOBJECT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_DELETE_SUBOBJECT_REQ_DATA_T                    tData;
} ODV3_DELETE_SUBOBJECT_REQ_T;
```

Packet description

Structure ODV3_DELETE_SUBOBJECT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A86	ODV3_DELETE_SUBOBJECT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_DELETE_SUBOBJECT_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of the object containing the subobject to be deleted.
bSubIndex	UINT8	0..255	Subindex of the subobject to be deleted

Table 109: ODV3_DELETE_SUBOBJECT_REQ - Delete Subobject Request

Packet structure reference

```
typedef struct ODV3_DELETE_SUBOBJECT_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
} ODV3_DELETE_SUBOBJECT_CNF_DATA_T;

typedef struct ODV3_DELETE_SUBOBJECT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_DELETE_SUBOBJECT_CNF_DATA_T                    tData;
} ODV3_DELETE_SUBOBJECT_CNF_T;
```

Packet description

Structure ODV3_DELETE_SUBOBJECT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A87	ODV3_DELETE_SUBOBJECT_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_DELETE_SUBOBJECT_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of the object formerly containing the subobject having been deleted
bSubIndex	UINT8	0..255	Subindex of the subobject having been deleted

Table 110: ODV3_DELETE_SUBOBJECT_CNF - Confirmation to Delete Subobject Request

5.5 ODV3_REGISTER_OBJECT_NOTIFY_REQ/CNF – Register Object Notify

This packet is used for registering the queue of an AP-task for indications of an object. The application can register for different types of indications with the same packet.

Indication Registration Flags (`bIndicationFlags`)

`bIndicationFlags` controls which indication options will be registered. The following options exist for indication registration:

Bit	Name and description
D4	ODV3_INDICATION_FLAGS_ON_WRITE_INVALIDATED If this bit is set and multiple indication receivers are registered for write, the application registering will get the following indication: ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication This bit must be combined with ODV3_INDICATION_FLAGS_ON_WRITE.
D3	ODV3_INDICATION_FLAGS_ON_INFO_UNDEFINED_SUBOBJ If this bit is set and a non-existing sub index is accessed, the application registering will get the following services: ODV3_GET_SUBOBJECT_INFO_IND/RES – Get Subobject Info Indication Necessary for the SDO Info Service to get information on undefined objects. (Set when undefined objects exist and SDO Info active)
D2	ODV3_INDICATION_FLAGS_ON_RW_UNDEFINED_SUBOBJ If this bit is set and a non-existing sub index is accessed, the application registering will get the following services: ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info ODV3_GET_OBJECT_SIZE_IND/RES – Get Object Size
D1	ODV3_INDICATION_FLAGS_ON_WRITE If this bit is set and a write to any subindex is executed, the application registering will get the following services: ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication for any subobject not being registered on subobject-level.
D0	ODV3_INDICATION_FLAGS_ON_READ If this bit is set and a read of any subindex is executed, the application registering will get the following services if no one is registered on the subobject level: ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary

Table 111: Indication Flags during Object Indication Registration

Note: If registered for write indications on object level, an additional registration on subobject level will result in two indications being sent to the application.

Packet structure reference

```
typedef struct ODV3_REGISTER_OBJECT_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bIndicationFlags;
} ODV3_REGISTER_OBJECT_NOTIFY_REQ_DATA_T;

typedef struct ODV3_REGISTER_OBJECT_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_REGISTER_OBJECT_NOTIFY_REQ_DATA_T              tData;
} ODV3_REGISTER_OBJECT_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_REGISTER_OBJECT_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A90	ODV3_REGISTER_OBJECT_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_REGISTER_OBJECT_NOTIFY_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to register from object notifications
bIndicationFlags	UINT8	0..255	Indication Flags

Table 112: ODV3_REGISTER_OBJECT_NOTIFY_REQ - Register Object Notify Request

Packet structure reference

```
typedef struct ODV3_REGISTER_OBJECT_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
} ODV3_REGISTER_OBJECT_NOTIFY_CNF_DATA_T;

typedef struct ODV3_REGISTER_OBJECT_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_REGISTER_OBJECT_NOTIFY_CNF_DATA_T                  tData;
} ODV3_REGISTER_OBJECT_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_REGISTER_OBJECT_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A91	ODV3_REGISTER_OBJECT_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_REGISTER_OBJECT_NOTIFY_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of object having been registered.

Table 113: ODV3_REGISTER_OBJECT_NOTIFY_CNF - Confirmation to Register Object Notify Request

5.6 ODV3_UNREGISTER_OBJECT_NOTIFY_REQ/CNF – Unregister Object Notify

This packet is used for de-registering the queue of an AP-task from receiving any indications related to a particular object specified with `usIndex` and `bSubIndex`.

The related indications are:

- ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary
- ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary
- ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info (if it was requested before)
- ODV3_GET_OBJECT_SIZE_IND/RES – Get Object Size (if it was requested before)

Note: If multiple application queues have been registered, the other queues will continue to receive indications.
Only the task, which has registered before, is able to de-register its indication wish. Other tasks have their own independent registration context and have to deregister on their own.

Packet structure reference

```
typedef struct ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
} ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_DATA_T;

typedef struct ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_DATA_T                tData;
} ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A92	ODV3_UNREGISTER_OBJECT_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_UNREGISTER_OBJECT_NOTIFY_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object to unregister from object notifications

Table 114: ODV3_UNREGISTER_OBJECT_NOTIFY_REQ - Unregister Object Notify Request

Packet structure reference

```
typedef struct ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
} ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_DATA_T;

typedef struct ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_DATA_T                tData;
} ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A93	ODV3_UNREGISTER_OBJECT_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_UNREGISTER_OBJECT_NOTIFY_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of object having been unregistered.

Table 115: ODV3_UNREGISTER_OBJECT_NOTIFY_CNF - Confirmation to Unregister Object Notify Request

5.7 ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ /CNF – Register Subobject Notify

This packet is used for registering the queue of an AP-task for indications of a subobject. The application can register for different types of indications with the same packet.

Indication Registration Flags (`bIndicationFlags`)

`bIndicationFlags` controls whether specific indication options will be registered immediately on object creation.

The following options exist for indication registration:

Bit	Name and description
D4	ODV3_INDICATION_FLAGS_ON_WRITE_INVALIDATED If this bit is set and multiple indication receivers are registered for write, the application registering will get the following indication: ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indication This bit must be combined with ODV3_INDICATION_FLAGS_ON_WRITE.
D1	ODV3_INDICATION_FLAGS_ON_WRITE If this bit is set and a write to any subindex is executed, the application registering will get the following services: ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND/RES – Write Object Validation Complete Indicationfor any subobject not being registered on subobject-level.
D0	ODV3_INDICATION_FLAGS_ON_READ If this bit is set and a read of any subindex is executed, the application registering will get the following services if no one is registered on the subobject level: ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary

Table 116: Indication Flags during Subobject Indication Registration

Packet structure reference

```
typedef struct ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
  TLR_UINT8                                               bIndicationFlags;
} ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_DATA_T;

typedef struct ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_DATA_T               tData;
} ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A94	ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object containing subobject to register
bSubIndex	UINT8	0..255	Subindex of subobject to register
bIndicationFlags	UINT8	0..255	Indication Flags

Table 117: ODV3_REGISTER_SUBOBJECT_NOTIFY_REQ - Register Subobject Notify Request

Packet structure reference

```
typedef struct ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
} ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_DATA_T;

typedef struct ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_DATA_T               tData;
} ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A95	ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of object containing subobject having been registered
bSubIndex	UINT8	0..255	Subindex of subobject having been registered

Table 118: ODV3_REGISTER_SUBOBJECT_NOTIFY_CNF - Confirmation to Register Subobject Notify Request

5.8 ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ/CNF – Unregister Subobject Notify

This packet is used for de-registering the queue of an AP-task from receiving any indications related to a particular subobject specified with `usIndex` and `bSubIndex`.

The related indications are:

- ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary
- ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary

Note: If multiple application queues have been registered, the other queues will continue to receive indications.
Only the task, which has registered before, is able to de-register its indication wish. Other tasks have their own independent registration context and have to deregister on their own.

Packet structure reference

```
typedef struct ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
} ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_DATA_T;

typedef struct ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_DATA_T             tData;
} ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6A96	ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of object containing subobject to unregister
bSubIndex	UINT8	0..255	Subindex of subobject to unregister

Table 119: ODV3_UNREGISTER_SUBOBJECT_NOTIFY_REQ - Unregister Subobject Notify Request

Packet structure reference

```
typedef struct ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT16                                              usIndex;
  TLR_UINT8                                               bSubIndex;
} ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_DATA_T;

typedef struct ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                     tHead;
  ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_DATA_T             tData;
} ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6A97	ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF_DATA_T			
usIndex	UINT16	0..65535	Index
bSubIndex	UINT8	0..255	Subindex

Table 120: ODV3_UNREGISTER_SUBOBJECT_NOTIFY_CNF - Confirmation to Unregister Subobject Notify Request

5.9 ODV3_REGISTER_UNDEFINED_NOTIFY_REQ/CNF – Register Undefined Notify

This packet is used for registering from the following indications if an object is accessed that does not exist:

- ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info
- ODV3_GET_OBJECT_SIZE_REQ/CNF – Get Object Size
- ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary
- ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary

Note: Only a single task can register for this service.

Packet structure reference

```
typedef struct ODV3_REGISTER_UNDEFINED_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_REGISTER_UNDEFINED_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_REGISTER_UNDEFINED_NOTIFY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AA0	ODV3_REGISTER_UNDEFINED_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 121: ODV3_REGISTER_UNDEFINED_NOTIFY_REQ - Register Undefined Notify Request

Packet structure reference

```
typedef struct ODV3_REGISTER_UNDEFINED_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_REGISTER_UNDEFINED_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_REGISTER_UNDEFINED_NOTIFY_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AA1	ODV3_REGISTER_UNDEFINED_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 122: ODV3_REGISTER_UNDEFINED_NOTIFY_CNF - Confirmation to Register Undefined Notify Request

5.10 ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ/CNF – Unregister Undefined Notify

This packet is used for de-registering from the following indications if any object is accessed that does not exist:

- ODV3_GET_OBJECT_ACCESS_INFO_IND/RES – Get Object Access Info
- ODV3_GET_OBJECT_SIZE_IND/RES – Get Object Size
- ODV3_READ_OBJECT_IND/RES – Read Object Indication from Dictionary
- ODV3_WRITE_OBJECT_IND/RES – Write Object Indication from Dictionary

Note: Only the task, which has registered before, is able to unregister.

Packet structure reference

```
typedef struct ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AA2	ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 123: ODV3_UNREGISTER_UNDEFINED_NOTIFY_REQ - Unregister Undefined Notify Request

Packet structure reference

```
typedef struct ODV3_UNREGISTER_UNDEFINED_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_UNREGISTER_UNDEFINED_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_UNREGISTER_UNDEFINED_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AA3	ODV3_UNREGISTER_UNDEFINED_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 124: ODV3_UNREGISTER_UNDEFINED_NOTIFY_CNF - Confirmation to Unregister Undefined Notify Request

5.11 ODV3_REGISTER_OBJINFO_NOTIFY_REQ/CNF – Register Object Info Notify

This packet is used for registering for the following indications:

- ODV3_GET_OBJECT_COUNT_IND/RES
- ODV3_GET_OBJECT_LIST_IND/RES
- ODV3_GET_OBJECT_INFO_IND/RES
- ODV3_GET_SUBOBJECT_INFO_IND/RES
- ODV3_GET_OBJECT_SIZE_IND/RES

The following conditions must be met for receiving indications whit this registration:

- Object does not exist
- Any of the related services is executed on the object dictionary

Note: Only a single task can register for this service.

Packet structure reference

```
typedef struct ODV3_REGISTER_OBJINFO_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_REGISTER_OBJINFO_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_REGISTER_OBJINFO_NOTIFY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AA4	ODV3_REGISTER_OBJINFO_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 125: ODV3_REGISTER_OBJINFO_NOTIFY_REQ - Register Object Info Notify Request

Packet structure reference

```
typedef struct ODV3_REGISTER_OBJINFO_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_REGISTER_OBJINFO_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_REGISTER_OBJINFO_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AA5	ODV3_REGISTER_OBJINFO_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 126: ODV3_REGISTER_OBJINFO_NOTIFY_CNF - Confirmation to Register Object Info Notify Request

5.12 ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ/CNF – Unregister Object Info Notify

This packet is used for de-registering from the following indications:

- ODV3_GET_OBJECT_COUNT_IND/RES – Get Object Count
- ODV3_GET_OBJECT_LIST_IND/RES – Get Object List matching the provided Access Mask Indication
- ODV3_GET_OBJECT_INFO_IND/RES – Get Object Info
- ODV3_GET_SUBOBJECT_INFO_IND/RES – Get Subobject Info Indication

Note: Only the task, which has registered before, is able to de-register.

Packet structure reference

```
typedef struct ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ_T;
```

Packet description

Structure ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AA6	ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 127: ODV3_UNREGISTER_OBJINFO_NOTIFY_REQ - Unregister Object Info Notify Request

Packet structure reference

```
typedef struct ODV3_UNREGISTER_OBJINFO_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_UNREGISTER_OBJINFO_NOTIFY_CNF_T;
```

Packet description

Structure ODV3_UNREGISTER_OBJINFO_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AA7	ODV3_UNREGISTER_OBJINFO_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 128: ODV3_UNREGISTER_OBJINFO_NOTIFY_CNF - Confirmation to Unregister Object Info Notify Request

5.13 ODV3_LOCK_OBJECT_DELETION_REQ/CNF – Lock Object Deletion

This packet is used for setting a lock preventing objects to be deleted at all.

Note: This service is not accessible on all protocol stacks since those stacks may use it to ensure their operating conditions.

Packet structure reference

```
typedef struct ODV3_LOCK_OBJECT_DELETION_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_LOCK_OBJECT_DELETION_REQ_T;
```

Packet description

Structure ODV3_LOCK_OBJECT_DELETION_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AB0	ODV3_LOCK_OBJECT_DELETION_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 129: ODV3_LOCK_OBJECT_DELETION_REQ - Lock Object Deletion Request

Packet structure reference

```
typedef struct ODV3_LOCK_OBJECT_DELETION_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_LOCK_OBJECT_DELETION_CNF_T;
```

Packet description

Structure ODV3_LOCK_OBJECT_DELETION_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AB1	ODV3_LOCK_OBJECT_DELETION_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 130: ODV3_LOCK_OBJECT_DELETION_CNF - Confirmation to Lock Object Deletion Request

5.14 ODV3_UNLOCK_OBJECT_DELETION_REQ/CNF – Unlock Object Deletion

This packet is used for releasing a lock preventing objects to be deleted.

Note: Only the task, which has locked the deletion previously, is able to unlock it again. This service is not accessible on all protocol stacks since those stacks may use it to ensure their operating conditions.

Packet structure reference

```
typedef struct ODV3_UNLOCK_OBJECT_DELETION_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_UNLOCK_OBJECT_DELETION_REQ_T;
```

Packet description

Structure ODV3_UNLOCK_OBJECT_DELETION_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AB2	ODV3_UNLOCK_OBJECT_DELETION_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 131: ODV3_UNLOCK_OBJECT_DELETION_REQ - Unlock Object Deletion Request

Packet structure reference

```
typedef struct ODV3_UNLOCK_OBJECT_DELETION_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} ODV3_UNLOCK_OBJECT_DELETION_CNF_T;
```

Packet description

Structure ODV3_UNLOCK_OBJECT_DELETION_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AB3	ODV3_UNLOCK_OBJECT_DELETION_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 132: ODV3_UNLOCK_OBJECT_DELETION_CNF - Confirmation to Unlock Object Deletion Request

5.15 ODV3_SET_OBJECT_NAME_REQ/CNF – Set Object Name

This packet is used for setting the name of an object to a new value. The object is selected with `usIndex`. The name is specified as a NUL-terminated string in `szName`.

Packet structure reference

```
typedef struct ODV3_SET_OBJECT_NAME_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_STR                                             szName[200];
} ODV3_SET_OBJECT_NAME_REQ_DATA_T;

typedef struct ODV3_SET_OBJECT_NAME_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_SET_OBJECT_NAME_REQ_DATA_T                     tData;
} ODV3_SET_OBJECT_NAME_REQ_T;
```

Packet description

Structure ODV3_SET_OBJECT_NAME_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AB4	ODV3_SET_OBJECT_NAME_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_SET_OBJECT_NAME_REQ_DATA_T			
usIndex	UINT16	0..65535	Index
szName[200]	STR		New name for the object must be a NUL-terminated string n is its length including the NUL terminator

Table 133: ODV3_SET_OBJECT_NAME_REQ - Set Object Name Request

Packet structure reference

```
typedef struct ODV3_SET_OBJECT_NAME_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
} ODV3_SET_OBJECT_NAME_CNF_DATA_T;

typedef struct ODV3_SET_OBJECT_NAME_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_SET_OBJECT_NAME_CNF_DATA_T                     tData;
} ODV3_SET_OBJECT_NAME_CNF_T;
```

Packet description

Structure ODV3_SET_OBJECT_NAME_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AB5	ODV3_SET_OBJECT_NAME_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_SET_OBJECT_NAME_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of the object

Table 134: ODV3_SET_OBJECT_NAME_CNF - Confirmation to Set Object Name Request

5.16 ODV3_SET_SUBOBJECT_NAME_REQ/CNF – Set Subobject Name

This packet is used for setting the name of a subobject to a new value. The subobject is selected with `usIndex` and `bSubIdx`. The name is specified as a NUL-terminated string in `szName`.

Packet structure reference

```
typedef struct ODV3_SET_SUBOBJECT_NAME_REQ_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
  TLR_STR                                             szName[200];
} ODV3_SET_SUBOBJECT_NAME_REQ_DATA_T;

typedef struct ODV3_SET_SUBOBJECT_NAME_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_SET_SUBOBJECT_NAME_REQ_DATA_T                  tData;
} ODV3_SET_SUBOBJECT_NAME_REQ_T;
```

Packet description

Structure ODV3_SET_SUBOBJECT_NAME_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AB6	ODV3_SET_SUBOBJECT_NAME_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_SET_SUBOBJECT_NAME_REQ_DATA_T			
usIndex	UINT16	0..65535	Index of the subobject
bSubIndex	UINT8	0..255	Subindex of the subobject
szName[200]	STR		New name for the subobject must be a NUL-terminated string n is its length including the NUL terminator

Table 135: ODV3_SET_SUBOBJECT_NAME_REQ - Set Subobject Name Request

Packet structure reference

```
typedef struct ODV3_SET_SUBOBJECT_NAME_CNF_DATA_Ttag
{
  TLR_UINT16                                          usIndex;
  TLR_UINT8                                           bSubIndex;
} ODV3_SET_SUBOBJECT_NAME_CNF_DATA_T;

typedef struct ODV3_SET_SUBOBJECT_NAME_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_SET_SUBOBJECT_NAME_CNF_DATA_T                  tData;
} ODV3_SET_SUBOBJECT_NAME_CNF_T;
```

Packet description

Structure ODV3_SET_SUBOBJECT_NAME_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	3	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AB7	ODV3_SET_SUBOBJECT_NAME_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_SET_SUBOBJECT_NAME_CNF_DATA_T			
usIndex	UINT16	0..65535	Index of the subobject
bSubIndex	UINT8	0..255	Subindex of the subobject

Table 136: ODV3_SET_SUBOBJECT_NAME_CNF - Confirmation to Set Subobject Name Request

5.17 ODV3_CREATE_DATATYPE_REQ/CNF – Create Data Type

This packet creates a new data type. These data type definitions are used for defining the data type unit size when creating an object/subobject.

Note: The data type definition storage uses an independent storage. Therefore, if needed an object-based data type description must be created explicitly e.g. an object within CANopen Data type area.

The coding of the data type (variable `usDatatype` of the request packet) is explained in *Table 7: Available data type definitions – Basic data type area* and *Table 8: Available data type definitions – Extended data type area*, see there.

Use the data type bit length parameter `ulDatatypeBitLength` to specify the bit length of the data type to be created.

Packet structure reference

```
typedef struct ODV3_CREATE_DATATYPE_REQ_DATA_Ttag
{
  TLR_UINT16                                          usDatatype;
  TLR_UINT32                                          ulDatatypeBitLength;
} ODV3_CREATE_DATA TYPE_REQ_DATA_T;

typedef struct ODV3_CREATE_DATATYPE_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_CREATE_DATA TYPE_REQ_DATA_T                     tData;
} ODV3_CREATE_DATA TYPE_REQ_T;
```

Packet description

Structure ODV3_CREATE_DATATYPE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	6	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AC0	ODV3_CREATE_DATATYPE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_CREATE_DATATYPE_REQ_DATA_T			
usDatatype	UINT16	0..65535	Data type to be created
ulDatatypeBitLength	UINT32		Bit Length of data type to be created

Table 137: ODV3_CREATE_DATATYPE_REQ - Create Data type Request

Packet structure reference

```
typedef struct ODV3_CREATE_DATATYPE_CNF_DATA_Ttag
{
  TLR_UINT16                                          usDatatype;
} ODV3_CREATE_DATATYPE_CNF_DATA_T;

typedef struct ODV3_CREATE_DATATYPE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_CREATE_DATATYPE_CNF_DATA_T                     tData;
} ODV3_CREATE_DATATYPE_CNF_T;
```

Packet description

Structure ODV3_CREATE_DATATYPE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AC1	ODV3_CREATE_DATATYPE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_CREATE_DATATYPE_CNF_DATA_T			
usDatatype	UINT16		Data type having been created

Table 138: ODV3_CREATE_DATATYPE_CNF - Confirmation to Create Data type Request

5.18 ODV3_DELETE_DATATYPE_REQ/CNF – Delete Data Type

This packet is used for deletion of a user-defined data type. Afterwards, it cannot be used for object/subobject creation anymore.

Note: Objects/subobjects which were created with that data type will continue to work. On object/subobject creation, the defined ulDatatypeBitLength will be copied into the object/subobject description and kept there. If you want to redefine such an object/subobject, it must be deleted and re-created afterwards.

The coding of the data type (variable `usDatatype` of the request packet) is explained in *Table 7: Available data type definitions – Basic data type area* and *Table 8: Available data type definitions – Extended data type area*, see there.

Do not delete default data type definitions.

Packet structure reference

```
typedef struct ODV3_DELETE_DATATYPE_REQ_DATA_Ttag
{
  TLR_UINT16                                          usDatatype;
} ODV3_DELETE_DATATYPE_REQ_DATA_T;

typedef struct ODV3_DELETE_DATATYPE_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_DELETE_DATATYPE_REQ_DATA_T                     tData;
} ODV3_DELETE_DATATYPE_REQ_T;
```

Packet description

Structure ODV3_DELETE_DATATYPE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Execute packet (any other value than 0 will be interpreted as an abort)
ulCmd	UINT32	0x6AC2	ODV3_DELETE_DATATYPE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_DELETE_DATATYPE_REQ_DATA_T			
usDatatype	UINT16		Data type to be deleted

Table 139: ODV3_DELETE_DATA TYPE_REQ - Delete Data type Request

Packet structure reference

```
typedef struct ODV3_DELETE_DATATYPE_CNF_DATA_Ttag
{
  TLR_UINT16                                          usDatatype;
} ODV3_DELETE_DATATYPE_CNF_DATA_T;

typedef struct ODV3_DELETE_DATATYPE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                 tHead;
  ODV3_DELETE_DATATYPE_CNF_DATA_T                     tData;
} ODV3_DELETE_DATATYPE_CNF_T;
```

Packet description

Structure ODV3_DELETE_DATATYPE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error Codes
ulCmd	UINT32	0x6AC3	ODV3_DELETE_DATATYPE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure ODV3_DELETE_DATA TYPE_CNF_DATA_T			
usDatatype	UINT16		Data type having been deleted

Table 140: ODV3_DELETE_DATATYPE_CNF - Confirmation to Delete Data type Request

6 Application holds object data and/or object descriptions

The object dictionary is a container or storage area for device parameter data structures. It stores the data itself and the properties of the related objects. For some devices, it is necessary to offer the possibility to realize a different behaviour and to shift object specific descriptions and/or data from the object dictionary component (stack) towards the user application. There are two main reasons to do this. The first reason is to reduce the memory consumption in the stack/firmware. The second reason is to decrease startup time, especially when the object dictionary is very large.

The ODV3 offers the following possibilities to handle object data and object descriptions.

- The object dictionary holds the complete object descriptions and data.
- The object dictionary holds the complete object descriptions but the data is stored in the application.
- The object dictionary holds the complete object descriptions and data but the subobject descriptions and data is stored in the application.
- The application holds the complete object descriptions and data. This means, that the object dictionary does not know anything about an object, even if it exists.

In all cases where object data and/or object descriptions are hold by the application, this has to be indicated to the object dictionary in order that the component can forward related requests to the application.

The following sections give an overview how to implement the handling of object descriptions and data in the application. The object dictionary allows you to separate object/subobject creation and registration. For details on the flags, it is necessary to read the information in the services parts of this documentation.

6.1 Application holds object data

Use case: object dictionary holds object and subobjects descriptions, application holds data

On object or subobject creation use the following flags:

- bValueInfo: ODV3_VALUE_INFO_VIRTUAL
- bIndicationFlags: ODV3_INDICATION_FLAGS_ON_WRITE (for writeable objects)
- bIndicationFlags: ODV3_INDICATION_FLAGS_ON_READ

Please note: If the flags are set during object creation they are also valid for all subobjects. If set on subobject level they are only valid for the specific subobject.

The actual value of the object/subobject has to be provided to the service user in the read response. Values provided in write requests have to be saved in the application.

6.2 Application holds object data and subobject descriptions

Use case: Object dictionary holds object descriptions, application holds subobject descriptions and data

On object creation use the following flags:

- bIndicationFlags: ODV3_INDICATION_FLAGS_ON_RW_UNDEFINED_SUBOBJ
- bIndicationFlags: ODV3_INDICATION_FLAGS_ON_INFO_UNDEFINED_SUBOBJ (flag is necessary if SDO Info is active)

This activates that the object dictionary sends indications to the application including read and write indications. The actual value of the object/subobject has to be provided to the service user in the read response. Values provided in write requests have to be saved in the application. Subobject descriptions are requested by the following packets:

- Get Object Size Indication,
- Get Object Access Info Indication, and
- Get Subobject Info Indication.

6.3 Application holds object data and all object descriptions

Use case: Application holds the complete object descriptions (including subobject descriptions) and data (the object dictionary does not know anything about object descriptions and data).

The object is not created with the create object request at all and thus not known by object dictionary. To induce the object dicionary to ask the application for information if an unknown object is accessed or a SDO Info Service is pending, the application has to register for services with the following requests: ODV3_REGISTER_UNDEFINED_NOTIFY_REQ

This activates that the object dictionary sends indications to the application including read and write indications. The actual value of the object/subobject has to be provided to the service user in the read response. Values provided in write requests have to be saved in the application. Object and subobject descriptions are requested by the following packets to the application:

- Get Object Size Indication and
- Get Object Access Info Indication.

ODV3_REGISTER_OBJINFO_NOTIFY_REQ: Activates the following indications to the application:

- Get Object Count Indication,
- Get Object List Indication,
- Get Object Info Indication,
- Get Subobject Info Indication, and
- Get Object Size Indication.

7 Protocol-specific Information

EtherCAT: Complete Access

Accessing a whole object with all its subobjects at once with one SDO read or SDO write access is called **Complete Access** in EtherCAT. The intention of the Complete Access is to speed up the acyclic access mainly to minimize the boot-up time of the device. A Complete Access can start with subindex 0 or subindex 1. Subobjects with dynamic entries are not allowed since the boundaries of subindices are unknown.

8 Status/error codes

8.1 Status/error codes Object Dictionary

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC09B0001	TLR_E_CO_OBJDICT_PROTOCOL_TIMEOUT SDO Protocol Timeout.
0xC09B0002	TLR_E_CO_OBJDICT_UNSUPPORTED_ACCESS Unsupported access.
0xC09B0003	TLR_E_CO_OBJDICT_OBJECT_IS_WRITE_ONLY Object is write only.
0xC09B0004	TLR_E_CO_OBJDICT_OBJECT_IS_READ_ONLY Object is read only.
0xC09B0005	TLR_E_CO_OBJDICT_OBJECT_DOES_NOT_EXIST Object does not exist.
0xC09B0006	TLR_E_CO_OBJDICT_OBJECT_CANNOT_BE_PDO_MAPPED Object cannot be mapped into PDO.
0xC09B0007	TLR_E_CO_OBJDICT_OBJECTS_WOULD_EXCEED_PDO_LENGTH The number and length of the objects to be mapped would exceed the PDO length.
0xC09B0008	TLR_E_CO_OBJDICT_GENERAL_PARAMETER_INCOMPATIBILITY General parameter incompatibility.
0xC09B0009	TLR_E_CO_OBJDICT_ACCESS_FAILED_DUE_TO_HW_ERROR Access failed due to hardware error.
0xC09B000A	TLR_E_CO_OBJDICT_DATATYPE_DOES_NOT_MATCH Data type does not match, length of service parameter does not match.
0xC09B000B	TLR_E_CO_OBJDICT_DATATYPE_LENGTH_IS_TOO_LONG Data type does not match, length of service parameter too high.
0xC09B000C	TLR_E_CO_OBJDICT_DATATYPE_LENGTH_IS_TOO_SHORT Data type does not match, length of service parameter too short.
0xC09B000D	TLR_E_CO_OBJDICT_SUBINDEX_DOES_NOT_EXIST Subindex does not exist.
0xC09B000E	TLR_E_CO_OBJDICT_RANGE_OF_PARAMETER_EXCEEDED Value range of parameter exceeded.
0xC09B000F	TLR_E_CO_OBJDICT_VALUE_OF_PARAMETER_WRITTEN_TOO_HIGH Value of parameter written too high.
0xC09B0010	TLR_E_CO_OBJDICT_VALUE_OF_PARAMETER_WRITTEN_TOO_LOW Value of parameter written too low.
0xC09B0011	TLR_E_CO_OBJDICT_MAXIMUM_VALUE_IS_LESS_THAN_MINIMUM_VALUE Maximum value is less than minimum value.
0xC09B0012	TLR_E_CO_OBJDICT_GENERAL_ERROR General error.
0xC09B0013	TLR_E_CO_OBJDICT_DATA_CANNOT_BE_TRANSFERRED_OR_STORED_TO_THE_APP Data cannot be transferred or stored to the application.
0xC09B0014	TLR_E_CO_OBJDICT_DATA_NO_TRANSFER_DUE_TO_LOCAL_CONTROL Data cannot be transferred or stored to the application because of local control.
0xC09B0015	TLR_E_CO_OBJDICT_DATA_NO_TRANSFER_DUE_TO_PRESENT_DEVICE_STATE Data cannot be transferred or stored to the application because of present device state.
0xC09B0016	TLR_E_CO_OBJDICT_NO_OBJECT_DICTIONARY_PRESENT Object dictionary dynamic generation fails or no object dictionary present.
0xC09B0017	TLR_E_CO_OBJDICT_GENERAL_INTERNAL_INCOMPATIBILITY General internal incompatibility.
0xC09B0018	TLR_E_CO_OBJDICT_ALL_BY_INDEX_UNSUPPORTED Access via AllByIndex unsupported.

Hex Value	Definition / Description
0xC09B8000	TLR_E_CO_OBJDICT_DELETION_LOCKED Deletion is locked
0xC09B8001	TLR_E_CO_OBJDICT_OTHER_TASK_HAS_LOCKED_DELETION Other task has locked deletion.
0xC09B8002	TLR_E_CO_OBJDICT_ONLY_ONE_READ_NOTIFY_ALLOWED Only one read notify allowed.
0xC09B8003	TLR_E_CO_OBJDICT_APPLICATION_NOT_REGISTERED Application task is not registered.
0xC09B8004	TLR_E_CO_OBJDICT_UNFRAGMENTABLE_PART_DOES_NOT_MATCH_SRCID Unfragmentable part of packet does not match SrcId.
0xC09B8005	TLR_E_CO_OBJDICT_UNFRAGMENTABLE_PART_DOES_NOT_MATCH_DESTID Unfragmentable part of packet does not match DestId.
0xC09B8006	TLR_E_CO_OBJDICT_SRCID_DOES_NOT_MATCH_ANY_FRAGMENTATION_BUFFER SrcId does not match any fragmentation buffer.
0xC09B8007	TLR_E_CO_OBJDICT_DESTID_DOES_NOT_MATCH_ANY_FRAGMENTATION_BUFFER DestId does not match any fragmentation buffer.
0xC09B8008	TLR_E_CO_OBJDICT_OBJECT_WAS_DELETED_IN_ACTION Object was deleted in action.
0xC09B8009	TLR_E_CO_OBJDICT_SUBOBJECT_WAS_DELETED_IN_ACTION Subobject was deleted in action.
0xC09B800A	TLR_E_CO_OBJDICT_REQUEST_ABORTED Request aborted.
0xC09B800B	TLR_E_CO_OBJDICT_VALUE_INFO_ONLY_SUPPORTED_ON_SIMPLE_VAR Given bValueInfo is only supported on SimpleVar.
0xC09B800C	TLR_E_CO_OBJDICT_DATATYPE_UNDEFINED Data type is undefined.
0xC09B800D	TLR_E_CO_OBJDICT_OTHER_APPLICATION_REGISTERED Other application is already registered.
0xC09B800E	TLR_E_CO_OBJDICT_DATATYPE_ALREADY_EXISTS CANopen Datatype already exists.
0xC09B800F	TLR_E_CO_OBJDICT_DATATYPE_DOES_NOT_EXIST CANopen Datatype does not exist.
0xC09B8010	TLR_E_CO_OBJDICT_VIRTUAL_OBJECT_CANNOT_BE_ACCESSED_WITHOUT_INDICATION Virtual object cannot be accessed without indication.
0xC09B8011	TLR_E_CO_OBJDICT_FRAGMENTATION_IMPOSSIBLE Fragmentation impossible.
0xC09B8012	TLR_E_CO_OBJDICT_ACCESS_VIA_UNDEFINED_NOTIFY_DENIED Access via undefined notify denied.
0xC09B8013	TLR_E_CO_OBJDICT_OBJECT_ALREADY_EXISTS Object already exists.
0xC09B8014	TLR_E_CO_OBJDICT_SUBOBJECT_ALREADY_EXISTS Subobject already exists.
0xC09B8015	TLR_E_CO_OBJDICT_CANNOT_BE_DELETED_NOT_OWNER Object/Subobject cannot be deleted. Requestor is not owner.
0xC09B8016	TLR_E_CO_OBJDICT_MAX_NUMBER_OF_SUBOBJECTS_EXCEEDED Maximum number of subobjects exceeded.
0xC09B8017	TLR_E_CO_OBJDICT_HAS_NO_DEFAULT_VALUE Has no default value.
0xC09B8018	TLR_E_CO_OBJDICT_INDICATION_FLAGS_NOT_ALLOWED Indication flags not allowed.
0xC09B8019	TLR_E_CO_OBJDICT_INDICATION_FLAGS_NOT_SUPPORTED Indication flags not supported.
0xC09B801A	TLR_E_CO_OBJDICT_ONLY_ONE_RW_UNDEFINED_SUBOBJ_NOTIFY_ALLOWED Only one read/write notify for non-existing subobject notify allowed.

Hex Value	Definition / Description
0xC09B801B	TLR_E_CO_OBJDICT_ONLY_ONE_INFO_UNDEFINED_SUBOBJ_NOTIFY_ALLOWED Only one info notify for non-existing subobject notify allowed.
0xC09B801C	TLR_E_CO_OBJDICT_CREATION_NOT_COMPLETED Running creation of object/subobjects not yet completed.
0xC09B801D	TLR_E_CO_OBJDICT_TOTAL_DATA_BYTES_AND_PKT_LEN_MISMATCH Dependency of total data bytes and packet length not fulfilled.
0xC09B801E	TLR_E_CO_OBJDICT_NO_MORE_SUBINDEXES No more subindexes.
0xC09B801F	TLR_E_CO_OBJDICT_VIRTUAL_CANNOT_BE_COMBINED_WITH_INITIAL_VALUE Virtual cannot be combined with initial value.

Table 141: Status/Error codes of the ODV3 task

8.2 SDO Abort Codes

SDO Abort Codes may be issued for instance within
`ODV3_WRITE_MULTIPLE_PARAMETER_BY_INDEX_CNF` or
`ODV3_READ_MULTIPLE_PARAMETER_BY_INDEX_CNF`.

Return codes are generally structured into the following elements:

- Error Class
- Error Code
- Additional Code

Error Class

The element Error Class (1 byte) generally classifies the kind of error, see table:

Class (hex)	Name	Description
1	vfd-state	Status error in virtual field device
2	application-reference	Error in application program
3	definition	Definition error
4	resource	Resource error
5	service	Error in service execution
6	access	Access error
7	od	Error in object dictionary
8	other	Other error

Table 142: Possible Values of Error Class

Error Code

The element Error Code (1 byte) accomplishes the more precise differentiation of the error cause within an Error Class. For Error Class = 8 (Other error) only Error Code = 0 (Other error code) is defined, for more detailing the Additional Code is available.

Additional Code

The additional code contains the detailed error description

The following table explains the relationship between SDO Abort Code on one hand and error class, error code and additional code on the other hand and gives a short description on the kind error:

SDO Abort Code	Error Class	Error Code	Additional Code	Description
0x00000000	0	0	0	No error
0x05030000	5	3	0	For CANopen and EtherCAT: Toggle bit not changed – Error in toggle bit at segmented transfer
0x05040000	5	4	0	SDO Protocol Timeout (at service execution)
0x05040001	5	4	1	Client/Server command specifier/ID not valid (i.e. unknown command specifier for SDO Service)
0x05040002				For CANopen: Invalid block size (only in block mode) For EtherCAT: Reserved
0x05040003				For CANopen: Invalid sequence number (only in block mode) For EtherCAT: Reserved
0x05040004				For CANopen: CRC error (only in block mode) For EtherCAT: Reserved
0x05040005	5	4	5	Out of memory - Memory overflow occurred at SDO Service execution
0x06010000	6	1	0	Unsupported access to an object
0x06010001	6	1	1	Attempt to read a write –only object (Index may only be written but not read)
0x06010002	6	1	2	Attempt to write a read –only object (Index may only be read but not written- parameter lock active)
0x06020000	6	2	0	Object does not exist in the object dictionary – for instance, wrong index.
0x06040041	6	4	41	Object cannot be mapped to the PDO
0x06040042	6	4	42	The number and length of objects to be mapped would exceed the length of the PDO
0x06040043	6	4	43	General parameter incompatibility reason. (i.e. the data format of the parameter is incompatible for the index)
0x06040044	6	4	44	For CANopen and EtherCAT: Reserved
0x06040047	6	4	47	General internal incompatibility in the device. (Device-internal error)
0x06060000	6	6	0	Access failed due to a hardware error. (Device-internal error)
0x06070010	6	7	10	Data type does not match, length of service parameter does not match (i.e. Parameter length error – data format for index has wrong size)
0x06070012	6	7	12	Data type does not match, length of service parameter too high – Data format to large for index
0x06070013	6	7	13	Data type does not match, length of service parameter too low – Data format to small for index
0x06090011	6	9	11	Subindex does not exist.
0x06090030	6	9	30	Value range of parameter exceeded (only for write access) i.e. value is invalid

SDO Abort Code	Error Class	Error Code	Additional Code	Description
0x06090031	6	9	31	Value of parameter written too high.
0x06090032	6	9	32	Value of parameter written too low.
0x06090036	6	9	36	Maximum value is less than minimum value
0x08000000	8	0	0	General error
0x08000020	8	0	20	Data cannot be transferred or stored to the application.
0x08000021	8	0	21	Data cannot be transferred or stored to the application because of local control.
0x08000022	8	0	22	Data cannot be transferred or stored to the application because of the present device state.
0x08000023	8	0	23	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of a file error).

Table 143: List of SDO Abort Codes

9 Glossary

Access Mask

An access mask is used to select groups of objects. See section „*Object access masks*“ on page 19 for more information.

Acyclic Data

Acyclic data are transferred only once or repeatedly, but not periodically. This may be done using Service Data Objects, see SDO.

CANopen

CANopen is a networking technology for automation based on the CAN bus and an own communication protocol which is located at the application layer (layer 7) within the OSI model of networking.

CAN and CANopen both have originally been developed by Bosch and are supported by the CiA. It has been standardized in EN 50325-4 and ISO 11898 (CAN).

CiA

CAN in Automation

The CiA is the user organization promoting CAN and CANopen and coordinating its development. It has been founded in 1992 and consists of device manufacturers, research institutes in the field of automation and users as independent organization.

The CiA defines the CANopen specifications, represents the interests of its members in international standardization organizations and certifies the CANopen conformance of devices.

It is located in Nuremberg, Germany.

CoE

CANopen over EtherCAT

CoE denominates a method for accessing an object dictionary which has been adopted for use in EtherCAT from CANopen.

Cyclic Data

Cyclic data are transferred periodically. This can be done with Process Data Objects, see PDO.

Complete Access

Accessing a whole object with all its subobjects at once with one SDO read or SDO write.

Entry

Storage area for a subobject containing a specific device parameter data structure (or a single parameter) which can be accessed and identified by index and subindex.

Its office is located in Berlin, Germany.

ETG

EtherCAT Technology Group

The ETG is the user organization promoting EtherCAT and coordinating its development. It consists of device manufacturers, research institutes in the field of automation and users as independent organization.

The ETG defines the EtherCAT specifications, represents the interests of its members in international standardization organizations and certifies the EtherCAT conformance of devices.

The ETG is located in Nuremberg, Germany.

EtherCAT

EtherCAT is a networking technology for automation based on the real-time Ethernet which has been developed and patented by Beckhoff Automation GmbH, Verl, Germany. The user organization ETG caters for EtherCAT.

Index

A value between 0 and 65535 identifying a specific object within an object dictionary.

IP Address

Internet Protocol Address

Address of a network adapter as defined in the Internet Protocol. It consists of 4 bytes and is usually written in decimal notation with separating dots.

MAC-Address

(only EtherCAT)

World-wide unique address of a network adapter (for Ethernet). It consists of 6 bytes and can thus be interpreted as an UNSIGNED48. The first 3 bytes encode the adapter's manufacturer.

Object

Named storage area for device parameter data structures (or single parameters) belonging together. An object is uniquely identified by its index which ranges from 0 to 65535.

An object may contains up to 255 parameter data structures (or single parameters) as subobjects (entries) which can be separately identified by their subindex. At subindex 0 usually the number of supported subindices of the object is stored.

Object Dictionary

An object dictionary is a storage area for device parameter data structures. All device parameters of a device are stored there. In detail the object dictionary contains objects and data types which have predefined by the standard, the manufacturer of the device and the device profile.

It acts as a link between the application and the CANopen, or EtherCAT protocol stack. It is accessed in standardized manner very similarly in CANopen, EtherCAT.

PDO

Process Data Object

Specific data object for cyclic data communication used in CANopen, EtherCAT. A PDO is used to transport real time data without any acknowledge.

PDO Mapping

PDO mapping is a process in which a device is adjusted to the format of PDOs for cyclic data communication of another device. There are two cases in general:

1. Fixed mapping: The order of process data of a (slave) device is fixed. The master must be adjusted to match this fixed mapping.
2. Variable mapping: Process data can be selected and ordered in the PDO as required. This is done by supplying index and subindex of the object to be mapped and the length of the data to be mapped to the mapping object.

Receive PDO

Process Data Object for input (receiving data)

RPDO

See Receive PDO

RxPDO

See Receive PDO

SDO

Service Data Object

Specific data object for acyclic data communication, i.e. mailbox-based communication. It is typically used for parameterization and accessing the object dictionary. Each device must provide at least one channel for SDO communication.

Subindex

A value between 0 and 255 identifying a specific subobject within an object.

SYNC

Synchronization Object

Sync Manager

Synchronization Manager (EtherCAT term)

According to the EtherCAT specification, a sync manager is a collection of control elements to coordinate access to concurrently used objects.

A sync manager synchronizes the data communication on a specific communication channel. It is configured for managing either input or output and either for cyclic or for acyclic communication. Configuration is done within the device definition file. Up to 32 sync managers can be configured

The following data are assigned to a sync manager:

- PDO Mappings (either only RPDOs or only TPDOs, but not mixed)
- Sync Manager length (specified in bytes)
- Sync Manager type (Inputs/ Outputs)
- Flags

Usually 4 sync managers numbered 0 to 3 are configured for:

- Mailbox receive (master to slave)
- Mailbox send (slave to master)
- Process data input (slave to master)
- Process data output (master to slave)

Transmit PDO

Process Data Object for output (transmitting data)

TPDO

See Transmit PDO

TxPDO

See Transmit PDO

10 Appendix

10.1 Detailed Description of Data Types

The following section describes the data types listed Table 8: Available data type definitions – Extended data type area more precisely.

10.1.1 0x0020: PDO_COMMUNICATION_PARAMETER

Only for CANopen			
Index	**Subindex**	**Field in PDO_COMMUNICATION_PARAMETER**	**Data Type**
0x0020	0	number of supported entries in the record	UNSIGNED8
	1	COB-ID	UNSIGNED32
	2	transmission type	UNSIGNED8
	3	inhibit time	UNSIGNED16
	4	reserved	UNSIGNED8
	5	event timer	UNSIGNED16

Table 144: Data type 0x0020: PDO_COMMUNICATION_PARAMETER

10.1.2 0x0021: PDO_MAPPING

Only for CANopen and EtherCAT			
Index	**Subindex**	**Field in PDO_MAPPING**	**Data Type**
0x0021	0	number of mapped objects in PDO	UNSIGNED8
	1	1st object to be mapped	UNSIGNED32
	2	2nd object to be mapped	UNSIGNED32
		: : : : : : : :	UNSIGNED32
	0x40	64th object to be mapped	UNSIGNED32

Table 145: Data type0x0021: PDO_MAPPING

10.1.3 0x0022: SDO_PARAMETER (CANopen only)

CANopen			
Index	**Subindex**	**Field in SDO_PARAMETER**	**Data Type**
0x0022	0	number of supported entries	UNSIGNED8
	1	COB-ID client -> server	UNSIGNED32
	2	COB-ID server -> client	UNSIGNED32
	3	node ID of SDOs client resp. server	UNSIGNED8

Table 146: Data type 0x0022: SDO_PARAMETER

10.1.4 0x0023: IDENTITY

CANopen, EtherCAT			
Index	**Subindex**	**Field in IDENTITY**	**Data Type**
0x0023	0	number of supported entries	UNSIGNED8
	1	Vendor-ID	UNSIGNED32
	2	Product code	UNSIGNED32
	3	Revision number	UNSIGNED32
	4	Serial number	UNSIGNED32

Table 147: Data type0x0023: IDENTITY

10.1.5 0x0800 – 0x0FFF: Enumerated Data Type Area of EtherCAT

EtherCAT provides the area from 0x0800 to 0x0FFF for defining enumerated data types.

According to the EtherCAT Specification, Part 6, each item has a data type that specifies the number of bits occupied (e.g. BIT5 or UNSIGNED32) and a list of entries that specifies integer value (data type is UNSIGNED32) and the enumeration visible string as shown in the table below.

These are optional and only for read access. They do not have a PDO mapping.

Only for EtherCAT			
Subindex	**Description**	**Data type**	**Value**
0	Number of entries	UNSIGNED8	Number of enumeration values n
	Padding	UNSIGNED8	Padding for obtaining even boundaries
1	Enum 1	OCTET STRING	UNSIGNED32 as integer value, VISIBLE STRING as ERNUMERATION STRINHG
	…		
1	Enum n	OCTET STRING	UNSIGNED32 as integer value, VISIBLE STRING as ERNUMERATION STRINHG

Table 148: 0x0800 – 0x0FFF : Enumerated Data Type Area

10.2 List of tables

10.3 List of figures

10.4 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com