PUBLICLY
AVAILABLE
SPECIFICATION

IEC
PAS 62407

First edition
2005-06

Real-time Ethernet control automation technology (EtherCAT^{TM})

Reference number
IEC/PAS 62407:2005(E)

PUBLICLY AVAILABLE SPECIFICATION

IEC PAS 62407

First edition
2005-06

Real-time Ethernet control automation technology (EtherCAT^{TM})

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CONTENTS

FOREWORD 18
1 Introduction 20
2 Scope 20
3 Normative References 20
4 Terms, Definitions and Abbreviations 21
4.1 IEC 61158 definitions 21
4.2 Definitions from other standards 24
4.3 EtherCAT Definitions 24
4.4 Abbreviations 26
5 Technology Overview 28
5.1 Operating principle 28
5.2 Topology 28
5.2.1 Telegram processing principles 28
5.3 Data Link Layer overview 29
5.4 Error Detection Overview 30
5.5 Parameter and Process Data Handling Introduction 30
5.6 Node Reference Model 30
5.6.1 Mapping onto OSI basic reference model 30
5.6.2 Physical Layer 31
5.6.3 Data Link Layer 31
5.6.4 Application Layer 31
5.7 Data types and encoding rules 32
5.7.1 General description of data types and encoding rules 32
5.7.2 Data type definitions 32
5.7.3 Bit sequences 33
5.7.3.1 Definition of bit sequences 33
5.7.3.2 Transfer syntax for bit sequences 33
5.7.4 Basic data types 34
5.7.4.1 Type Definition 34
5.7.4.2 NIL 34
5.7.4.3 Boolean 34
5.7.4.4 Void 34
5.7.4.5 Unsigned Integer 34
5.7.4.6 Signed Integer 35
5.7.4.7 Floating-Point Numbers 36
5.7.5 Extended data types 37
5.7.5.1 Octet String 37
5.7.5.2 Visible String 37
6 Contents of Part 2: Physical Layer 38
6.1 Overview 38
6.2 100BASE-TX 38
6.3 100BASE-FX 38
6.4 LVDS 38
7 Contents of Part 3: Data Link Layer Service definition 39
7.1 Overview 39

7.1.1 Relation to ISO/IEC8802-3 39
7.1.2 Frame Structure 39
7.1.3 EtherCAT Modes 39
7.1.3.1 Open Mode 39
7.1.3.2 Direct Mode 40
7.1.4 Segment represents logical ring 40
7.1.5 Addressing 41
7.1.5.1 Segment addressing 41
7.1.5.2 Device addressing 41
7.1.5.2.1 Position addressing 41
7.1.5.2.2 Node addressing 41
7.1.5.3 Logical Addressing 42
7.1.5.3.1 FMMU introduction 42
7.1.6 Slave Classification 42
7.1.6.1 Full Slave 42
7.1.6.2 Basic Slave 43
7.2 EtherCAT Services 43
7.2.1 Read 43
7.2.1.1 Auto Increment Physical Read (APRD) 43
7.2.1.2 Node-Addressed Physical Read (NPRD) 44
7.2.1.3 Logical Read (LRD) 45
7.2.1.4 Broadcast Read (BRD) 46
7.2.2 Write 46
7.2.2.1 Auto Increment Physical Write (APWR) 46
7.2.2.2 Node-Addressed Physical Write (NPWR) 47
7.2.2.3 Logical Write (LWR) 48
7.2.2.4 Broadcast Write (BWR) 49
7.2.3 ReadWrite 50
7.2.3.1 Logical ReadWrite (LRW) 50
7.2.3.2 Auto Increment Physical Read Multiple Write (ARMW) 51
7.3 EtherCAT Attributes 52
7.3.1 Register 53
7.3.1.1 Management 53
7.3.1.1.1 DL Information 53
7.3.1.1.2 Configured Station Address 53
7.3.1.1.3 DL Control 54
7.3.1.1.4 DL Status 54
7.3.1.1.5 AL Management Interface 54
7.3.1.1.5.1 AL Control 56
7.3.1.1.5.2 AL Status 56
7.3.1.1.6 PDI Control 57
7.3.1.1.7 PDI Configuration 57
7.3.1.1.8 AL Event 57
7.3.1.2 Statistics 58
7.3.1.2.1 Channel CRC Fault Counter 58
7.3.1.3 Watchdogs 58
7.3.1.3.1 Watchdog Divider 58
7.3.1.3.2 PDI Watchdog 58
7.3.1.3.3 Sync Manager Channel Watchdog 58

7.3.1.3.4 Sync Manager Watchdog Status 58
7.3.1.4 Slave Information Interface 59
7.3.1.4.1 Slave Information Interface Content 59
7.3.1.4.2 Slave Information Interface Size 60
7.3.1.4.3 Slave Information Interface Control/Status 60
7.3.1.4.4 Actual Slave Information Interface Address 60
7.3.1.4.5 Actual Slave Information Interface Data 61
7.3.1.5 Fieldbus Memory Management Unit (FMMU) 61
7.3.1.6 Sync Manager 62
7.3.1.7 Distributed Clock 63
7.3.2 Application Memory 64
7.3.2.1 Queued Access Type 64
7.3.2.1.1 Write Access from Master 64
7.3.2.1.2 Read Access from Master 66
7.3.2.2 Buffered Access Type 68
7.3.2.2.1 Write Access from Master 68
7.3.2.2.2 Read Access from Master 69
8 Contents of Part 4: Data Link Layer Protocol definition 70
8.1 Frame Structure 70
8.1.1 EtherCAT telegram inside an Ethernet frame 70
8.1.2 EtherCAT frame inside an UDP datagram 71
8.1.3 EtherCAT telegram structure 71
8.1.3.1 Read 72
8.1.3.1.1 Auto Increment Physical Read (APRD) 72
8.1.3.1.2 Node-Addressed Physical Read (NPRD) 72
8.1.3.1.3 Logical Read (LRD) 73
8.1.3.1.4 Broadcast Read (BRD) 73
8.1.3.2 Write 74
8.1.3.2.1 Auto Increment 74
8.1.3.2.2 Node-Addressed Physical Write (NPWR) 74
8.1.3.2.3 Logical Write (LWR) 75
8.1.3.2.4 Broadcast Write (BWR) 75
8.1.3.3 ReadWrite 75
8.1.3.3.1 Logical ReadWrite (LRW) 75
8.1.3.3.2 Auto Increment Physical Read Multiple Write (ARMW) 76
8.2 EtherCAT Attributes 77
8.2.1 Register 77
8.2.1.1 Management 77
8.2.1.1.1 DL Information 77
8.2.1.1.1.1 Coding 77
8.2.1.1.1.2 Description 77
8.2.1.1.2 Configured Station Address 77
8.2.1.1.2.1 Coding 77
8.2.1.1.2.2 Description 77
8.2.1.1.3 DL Control 78
8.2.1.1.3.1 Coding 78
8.2.1.1.3.2 Description 78
8.2.1.1.4 DL Status 78

8.2.1.1.4.1 Coding 78
8.2.1.1.4.2 Description....79
8.2.1.1.5 AL Management Interface 79
8.2.1.1.5.1 AL Control....79
8.2.1.1.5.2 AL Status 80
8.2.1.1.6 PDI Control....81
8.2.1.1.6.1 Coding 81
8.2.1.1.6.2 Description....81
8.2.1.1.7 PDI Configuration 81
8.2.1.1.7.1 8 Bit Micro Controller Interface 81
8.2.1.1.7.2 16 Bit Micro Controller Interface 82
8.2.1.1.8 AL Event 83
8.2.1.1.8.1 Coding 83
8.2.1.1.8.2 Description....83
8.2.1.2 Statistics 85
8.2.1.2.1 Channel CRC Fault Counter....85
8.2.1.2.1.1 Coding 85
8.2.1.2.1.2 Description....85
8.2.1.3 Watchdogs 85
8.2.1.3.1 Watchdog Divider 85
8.2.1.3.1.1 Coding 85
8.2.1.3.1.2 Description....85
8.2.1.3.2 PDI Watchdog 85
8.2.1.3.2.1 Coding 85
8.2.1.3.2.2 Description....85
8.2.1.3.3 Sync Manager Channel Watchdog....86
8.2.1.3.3.1 Coding 86
8.2.1.3.3.2 Description....86
8.2.1.3.4 Sync Manager Watchdog Status....87
8.2.1.3.4.1 Coding 87
8.2.1.3.4.2 Description....87
8.2.1.4 Slave Information Interface 88
8.2.1.4.1 Slave Information Interface Access Flow Charts 88
8.2.1.4.1.1 Read Operation....88
8.2.1.4.1.2 Write Operation....89
8.2.1.4.1.3 Reload Operation 90
8.2.1.4.2 Slave Information Interface Area 90
8.2.1.4.2.1 Slave Information Interface Size 91
8.2.1.4.2.2 Slave Information Interface Control/Status 92
8.2.1.4.2.3 Actual Slave Information Interface Address 93
8.2.1.4.2.4 Actual Slave Information Interface Data 93
8.2.1.5 Fieldbus Memory Management Unit (FMMU) 94
8.2.1.5.1 Coding....94
8.2.1.5.2 Description 94
8.2.1.6 Sync Manager 95
8.2.1.6.1 Coding....95

8.2.1.6.2 Description 95
8.2.1.7 Distributed Clock 97
8.2.1.7.1 Coding....97
8.2.1.7.2 Description 98
8.2.2 Application Memory....100
8.2.2.1 Queued Access Type Flow Charts....100
8.2.2.1.1 Write Access from Master (see Figures 22 and 23) 100
8.2.2.1.2 Read Access from Master (see Figures 24 and 25) 102
8.2.2.2 Buffered Access Type Flow Charts....104
8.2.2.2.1 Write Access from Master (see Figures 26 and 27) 104
8.2.2.2.2 Read Access from Master (see Figures 28 and 29) 106
9 Contents of Part 5: Application Layer Service definition 108
9.1 Communication Model Overview 108
9.2 Slave....108
9.2.1 Slave AL Classification....108
9.2.2 Simple Slave Device 108
9.2.3 Complex Slave Device 108
9.2.4 Management....109
9.2.4.1 Overview 109
9.2.4.2 EtherCAT States....111
9.2.4.2.1 Init....111
9.2.4.2.2 Pre-Operational 111
9.2.4.2.3 Safe-Operational 111
9.2.4.2.4 Operational....111
9.2.4.2.5 Bootstrap....111
9.2.4.3 EtherCAT State Services 111
9.2.4.3.1 AL Control Sequence 111
9.2.4.3.2 AL State Changed Sequence 113
9.2.4.3.3 AL Control Service....113
9.2.4.3.4 AL State Changed Service 114
9.2.4.3.5 AL State Acknowledge Service....114
9.2.4.4 Local Management services 115
9.2.4.4.1 Start Mailbox Communication....115
9.2.4.4.2 Stop Mailbox Communication 115
9.2.4.4.3 Start Input Update 115
9.2.4.4.4 Stop Input Update....116
9.2.4.4.5 Start Output Update....116
9.2.4.4.6 Stop Output Update 116
9.2.4.4.7 Start Bootstrap Mode 116
9.2.5 Mailbox....117
9.2.5.1 Overview 117
9.2.5.1.1 Communication from Master to Slave 117
9.2.5.1.2 Communication from Slave to Master 117
9.2.5.2 Mailbox Transmission Services 117
9.2.5.2.1 Mailbox Write....119

9.2.5.2.2 Mailbox Read 119
9.2.5.3 Mailbox protocols 120
9.2.6 EtherCAT Process Data 121
9.2.6.1 Overview 121
9.2.6.2 Process Output Data 121
9.2.6.3 Process Input Data 122
9.2.7 CANopen over EtherCAT 123
9.2.7.1 Overview 123
9.2.7.2 Object Dictionary 124
9.2.7.2.1 Overview 124
9.2.7.2.2 Data Type Area 124
9.2.7.2.3 CoE Communication Area 125
9.2.7.2.3.1 Device Type 125
9.2.7.2.3.2 Manufacturer Device Name 125
9.2.7.2.3.3 Hardware Version 125
9.2.7.2.3.4 Software Version 125
9.2.7.2.3.5 Identity 125
9.2.7.2.3.6 EtherCAT Fixed Station Address 126
9.2.7.2.3.7 Virtual MAC Address 126
9.2.7.2.3.8 IP Address Info 126
9.2.7.2.3.9 Receive PDO Mapping 126
9.2.7.2.3.10 Transmit PDO Mapping 126
9.2.7.2.3.11 Sync Manager Communication Type 127
9.2.7.2.3.12 Sync Manager PDO Assignment 127
9.2.7.2.3.13 EtherCAT Physical Memory Access 127
9.2.7.3 SDO Services 127
9.2.7.3.1 Overview 127
9.2.7.3.2 SDO Download 129
9.2.7.3.3 SDO Upload 134
9.2.7.3.4 Abort SDO Transfer 139
9.2.7.4 SDO Information Services 140
9.2.7.4.1 Overview 140
9.2.7.4.2 SDO Information Service 141
9.2.7.4.3 Get OD List 142
9.2.7.4.4 Get Object Description 143
9.2.7.4.5 Get Entry Description 144
9.2.7.5 Process Data 146
9.2.7.5.1 Overview 146
9.2.7.5.2 Process Data Objects (PDO) 148
9.2.7.5.3 Sync Manager Channel Objects (SMCO) 148
9.2.7.5.4 PDO transmission via Mailbox 148
9.2.7.5.4.1 RxPDO 149
9.2.7.5.4.2 TxPDO 150
9.2.7.5.4.3 RxPDO Remote Transmission Request 151
9.2.7.5.4.4 TxPDO Remote Transmission Request 152
9.2.7.6 Command 153

9.2.7.7 Emergency 153
9.2.8 Ethernet over EtherCAT (EoE) 155
9.2.8.1 Overview 155
9.2.8.2 EoE Services...155
9.2.8.3 Initiate Ethernet over EtherCAT...156
9.2.8.4 Ethernet over EtherCAT Fragment 157
9.2.9 File Access over EtherCAT...158
9.2.9.1 Overview 158
9.2.9.2 FoE Services...158
9.2.9.3 FoE Read Sequence 159
9.2.9.3.1 FoE Read with Success 159
9.2.9.3.2 FoE Read with Error 159
9.2.9.4 FoE Write Sequence 160
9.2.9.4.1 FoE Write with Success 160
9.2.9.4.2 FoE Write with Error 160
9.2.9.4.3 FoE Write with Busy..161
9.2.9.5 FoE Read Request 161
9.2.9.6 FoE Write Request..162
9.2.9.7 FoE Data Request 162
9.2.9.8 FoE Ack Request..163
9.2.9.9 FoE Busy Request..163
9.2.9.10 FoE Error Request..164
9.2.9.11 FoE Busy Request..164
9.3 Master..165
9.3.1 Overview 165
9.3.2 Management..165
9.3.2.1 Slave Handler...165
9.3.3 Router..166
10 Contents of Part 6: Application Layer Protocol definition.................................167
10.1 Management 167
10.1.1 EtherCAT State Services..167
10.1.1.1 AL Control Service..167
10.1.1.1.1 AL Control Request (Indication) 167
10.1.1.1.1.1 Coding 167
10.1.1.1.1.2 Description..167
10.1.1.1.2 AL Control Response (Confirmation) 167
10.1.1.1.2.1 Coding 167
10.1.1.1.2.2 Description..167
10.1.1.2 AL State Changed Service 168
10.1.1.2.1 AL State Changed Request (Indication)...................168
10.1.1.2.1.1 Coding 168
10.1.1.2.1.2 Description..168
10.1.1.3 AL State Acknowledge Service...168
10.1.1.3.1 AL State Acknowledge Request (Indication) 168
10.1.1.3.1.1 Coding 168
10.1.1.3.1.2 Description..169
10.1.1.3.2 AL State Response (Confirmation) 169
10.1.1.3.2.1 Coding 169

10.1.1.3.2.2 Description 169
10.1.2 EtherCAT State Machine 170
10.1.2.1 Start Mailbox Communication 170
10.1.2.2 Stop Mailbox Communication 171
10.1.2.3 Start Input Update 172
10.1.2.4 Stop Input Update 173
10.1.2.5 Start Output Update 173
10.1.2.6 Stop Output Update 174
10.1.2.7 Acknowledge State Change 174
10.1.3 Slave Handler 175
10.1.3.1 Start Mailbox Communication 175
10.1.3.2 Stop Mailbox Communication 176
10.1.3.3 Start Input Update 177
10.1.3.4 Stop Input Update 178
10.1.3.5 Start Output Update 179
10.1.3.6 Stop Output Update 180
10.1.3.7 Acknowledge State Change 181
10.2 Mailbox 181
10.2.1 General Mailbox Header 181
10.2.1.1 Coding 181
10.2.1.2 Description 181
10.3 CANopen over EtherCAT 182
10.3.1 Coding 182
10.3.2 Description 182
10.3.3 SDO 183
10.3.3.1 Initiate SDO Download Expedited 183
10.3.3.1.1 Initiate SDO Download Expedited Request 183
10.3.3.1.1.1 Coding 183
10.3.3.1.1.2 Description 183
10.3.3.1.2 Initiate SDO Download Expedited Response 185
10.3.3.1.2.1 Coding 185
10.3.3.1.2.2 Description 185
10.3.3.2 Initiate SDO Download Normal 185
10.3.3.2.1 Initiate SDO Download Normal Request 185
10.3.3.2.1.1 Coding 185
10.3.3.2.1.2 Description 186
10.3.3.2.2 Initiate SDO Download Normal Response 186
10.3.3.2.2.1 Coding 186
10.3.3.2.2.2 Description 187
10.3.3.3 Download SDO Segment 187
10.3.3.3.1 Download SDO Segment Request 187
10.3.3.3.1.1 Coding 187
10.3.3.3.1.2 Description 187
10.3.3.3.2 Download SDO Segment Response 188
10.3.3.3.2.1 Coding 188
10.3.3.3.2.2 Description 188
10.3.3.4 Initiate SDO Upload Expedited 189
10.3.3.4.1 Initiate SDO Upload Expedited Request 189
10.3.3.4.1.1 Coding 189

10.3.3.4.1.2 Description..189
10.3.3.4.2 Initiate SDO Upload Expedited Response................190
10.3.3.4.2.1 Coding 190
10.3.3.4.2.2 Description..190
10.3.3.5 Initiate SDO Upload Normal 191
10.3.3.5.1 Initiate SDO Upload Normal Request.......................191
10.3.3.5.1.1 Coding 191
10.3.3.5.1.2 Description..191
10.3.3.5.2 Initiate SDO Upload Normal Response 192
10.3.3.5.2.1 Coding 192
10.3.3.5.2.2 Description..192
10.3.3.6 Upload SDO Segment...193
10.3.3.6.1 Upload SDO Segment Request 193
10.3.3.6.1.1 Coding 193
10.3.3.6.1.2 Description..193
10.3.3.6.2 Upload SDO Segment Response 194
10.3.3.6.2.1 Coding 194
10.3.3.6.2.2 Description..194
10.3.3.7 Abort SDO Transfer 195
10.3.3.7.1 Abort SDO Transfer Request 195
10.3.3.7.1.1 Coding 195
10.3.3.7.1.2 Description..195
10.3.3.7.2 SDO Abort Codes 196
10.3.4 SDO Information 197
10.3.4.1 SDO Information Service..197
10.3.4.1.1 Coding...197
10.3.4.1.2 Description 197
10.3.4.2 Get OD List 198
10.3.4.2.1 Get OD List Request...198
10.3.4.2.1.1 Coding 198
10.3.4.2.1.2 Description..198
10.3.4.2.2 Get OD List Response 198
10.3.4.2.2.1 Coding 198
10.3.4.2.2.2 Description..199
10.3.4.3 Get Object Description...199
10.3.4.3.1 Get Object Description Request 199
10.3.4.3.1.1 Coding 199
10.3.4.3.1.2 Description..199
10.3.4.3.2 Get Object Description Response...........................200
10.3.4.3.2.1 Coding 200
10.3.4.3.2.2 Description..200
10.3.4.4 Get Entry Description..201
10.3.4.4.1 Get Entry Description Request 201
10.3.4.4.1.1 Coding 201
10.3.4.4.1.2 Description..201
10.3.4.4.2 Get Object Description Response...........................202
10.3.4.4.2.1 Coding 202
10.3.4.4.2.2 Description..202
10.3.5 Emergency 204

10.3.5.1 Emergency Request....204
10.3.5.2 Emergency Error Codes....204
10.3.5.2.1 EtherCAT State Machine Transition Error....205
10.3.5.2.1.1 Error Code....205
10.3.5.2.1.2 Diagnostic Data....205
10.3.6 Process Data....207
10.3.6.1 Process Data Objects (PDO)....207
10.3.6.1.1 PDO Mapping....207
10.3.6.2 Sync Manager Channel Objects (SMCO)....208
10.3.6.2.1 Sync Manager PDO Assign....208
10.3.6.3 PDO transmission via Mailbox....209
10.3.6.3.1 RxPDO....209
10.3.6.3.2 TxPDO....209
10.3.6.3.3 RxPDO Remote Transmission Request....209
10.3.6.3.4 TxPDO Remote Transmission Request....210
10.3.7 Command....210
10.3.8 Object Dictionary....212
10.3.8.1 Object Definitions....212
10.3.8.2 Data Type Area....212
10.3.8.3 CoE Communication Area....214
10.3.8.3.1 Device Type....215
10.3.8.3.2 Manufacturer Device Name....215
10.3.8.3.3 Manufacturer Hardware Version....216
10.3.8.3.4 Manufacturer Software Version....216
10.3.8.3.5 Identity Object....216
10.3.8.3.6 EtherCAT Configured Station Address....218
10.3.8.3.7 Virtual MAC Address....218
10.3.8.3.8 IP Address Info....218
10.3.8.3.9 Receive PDO Mapping....220
10.3.8.3.10 Transmit PDO Mapping....220
10.3.8.3.11 Sync Manager Communication Type....221
10.3.8.3.12 Sync Manager PDO Assignment....223
10.3.8.3.12.1 Sync Manager Channel 0 (Mailbox Receive)....223
10.3.8.3.12.2 Sync Manager Channel 1 (Mailbox Send)....223
10.3.8.3.12.3 Sync Manager Channel 2 (Process Data Output)....224
10.3.8.3.12.4 Sync Manager Channel 3 (Process Data Input)....224
10.3.8.3.12.5 Sync Manager Channel 4-32....225
10.3.8.3.13 EtherCAT Physical Memory Access....226
10.4 Ethernet over EtherCAT....227
10.4.1 Initiate EoE Request....227
10.4.1.1 Coding....227
10.4.1.2 Description....227
10.4.2 EoE Fragment Request....227
10.4.2.1 Coding....227
10.4.2.2 Description....228

10.5 File Access over EtherCAT 228
10.5.1 Read Request 228
10.5.1.1 Coding 228
10.5.1.2 Description 229
10.5.2 Write Request 229
10.5.2.1 Coding 229
10.5.2.2 Description 229
10.5.3 Data Request 230
10.5.3.1 Coding 230
10.5.3.2 Description 230
10.5.4 Ack Request 231
10.5.4.1 Coding 231
10.5.4.2 Description 231
10.5.5 Error Request 231
10.5.5.1 Coding 231
10.5.5.2 Description 232
10.5.6 Busy Request 232
10.5.6.1 Coding 232
10.5.6.2 Description 233

FIGURES

Figure 1 – Frame Structure 29
Figure 2 – Nodes map data directly in frame 29
Figure 3 – Slave Node Reference Model 31
Figure 4: EtherCAT Telegrams embedded in Ethernet Frame 39
Figure 5: EtherCAT Telegrams embedded in UDP/IP 39
Figure 6: EtherCAT Segments in Open Mode 40
Figure 7: EtherCAT Segment in Direct Mode 40
Figure 8: Addressing Mode Overview 41
Figure 9: Fieldbus Memory Management Unit Overview 42
Figure 10: Application Layer Model 52
Figure 11: Successful Write sequence to AL Control Register 55
Figure 12: Successful Read sequence to the AL Status Register 56
Figure 13: Successful Write sequence to application memory area of Queued Access Type 65
Figure 14: Bad Write sequence to application memory area of Queued Access Type 66
Figure 15: Successful Read sequence to application memory area of Queued Access Type 67
Figure 16: Bad read sequence to application memory area of Queued Access Type 68
Figure 17: Successful Write sequence to application memory area of Buffered Access Type 69
Figure 18: Successful read sequence to application memory area of Buffered Access Type 70
Figure 19: Slave Information Interface Read Operation 88
Figure 20: Slave Information Interface Write Operation 89
Figure 21: Slave Information Interface Reload Operation 90

Figure 22: Write Queued Access Type Memory from Master 100
Figure 23: Read Queued Access Type Memory from AL Controller 101
Figure 24: Write Queued Access Type Memory from AL Controller 102
Figure 25: Read Queued Access Type Memory from Master 103
Figure 26: Write Buffered Access Type Memory from Master 104
Figure 27: Read 3 Buffered Access Type Memory from AL Controller 105
Figure 28: Write Buffered Access Type Memory from AL Controller 106
Figure 29: Read Buffered Access Type Memory from Master 107
Figure 30: Simple Slave Device 108
Figure 31: Complex Slave Device 109
Figure 32: EtherCAT State Machine and services 110
Figure 33: Successful AL Control sequence 112
Figure 34: Unsuccessful AL Control sequence 112
Figure 35: AL State Changed sequence 113
Figure 36: Successful Mailbox Write sequence 118
Figure 37: Successful Mailbox Read sequence 118
Figure 38: Process Output Data Sequence 122
Figure 39: Process Input Data Sequence 123
Figure 40: Server Model (CANopen part) 124
Figure 41: Successful single SDO-Download sequence 129
Figure 42: Unsuccessful single SDO-Download sequence 130
Figure 43: Successful segmented SDO-Download sequence 130
Figure 44: Successful single SDO-Upload sequence 134
Figure 45: Unsuccessful single SDO-Upload sequence 135
Figure 46: Successful segmented SDO-Upload sequence 135
Figure 47: SDO Information service sequence 140
Figure 48: PDO Mapping Example 147
Figure 49: Sync Manager PDO Assigment 147
Figure 50: RxPDO service 149
Figure 51: TxPDO service 150
Figure 52: RxPDO Remote Transmission Request service sequence 151
Figure 53: TxPDO Remote Transmission Request service sequence 152
Figure 54: Command sequence 153
Figure 55: Emergency service 154
Figure 56: Ethernet over EtherCAT sequence 156
Figure 57: FoE Read sequence with success 159
Figure 58: FoE Read sequence with error 159
Figure 59: FoE Write sequence with success 160
Figure 60: FoE Read sequence with error 160
Figure 61: FoE Write sequence with busy 161
Figure 62: Master Functional Overview 165
Figure 63: Processing of Start Mailbox Communication in the Slave 170
Figure 64: Processing of Stop Mailbox Communication in the Slave 171

Figure 65: Processing of Start Input Update in the Slave 172
Figure 66: Processing of Stop Input Update in the Slave 173
Figure 67: Processing of Start Output Update in the Slave 173
Figure 68: Processing of Stop Output Update in the Slave 174
Figure 69: Processing of Acknowledge State Change in the Slave 174
Figure 70: Processing of Start Mailbox Communication in the Master 175
Figure 71: Processing of Stop Mailbox Communication in the Master 176
Figure 72: Processing of Start Input Update in the Master 177
Figure 73: Processing of Stop Input Update in the Master 178
Figure 74: Processing of Start Output Update in the Master 179
Figure 75: Processing of Stop Output Update in the Master 180
Figure 76: Processing of Acknowledge State Change in the Master 181
Figure 77: Writing of PDO Mapping objects by the master....207
Figure 78: Writing of Sync Manager PDO Assign objects by the master....208

TABLES

Table 1 – Abbreviations 26
Table 2 –Data Type Syntax 32
Table 3 – Transfer Syntax for bit sequences....34
Table 4 – Transfer syntax for data type UNSIGNEDn....35
Table 5 –Transfer syntax for data type INTEGERn....35
Table 6 – Transfer syntax for data type INTEGERn....36
Table 7 – Auto Increment Physical Read (APRD)....43
Table 8 – Node-Addressed Physical Read (NPRD) 44
Table 9 – Logical Read (LRD) 45
Table 10 – Broadcast Read (BRD) 46
Table 11 – Auto Increment Physical Write (APWR) 47
Table 12 – Node-Addressed Physical Write (NPWR)....47
Table 13 – Logical Write (LWR)....48
Table 14 – Broadcast Write (BWR)....49
Table 15 – Logical ReadWrite (LRW)....50
Table 16 – Auto Increment Physical Read Multiple Write (ARMW)....51
Table 17 – EtherCAT telegram inside an Ethernet frame....71
Table 18 – EtherCAT frame inside an UDP datagram....71
Table 19 – Auto Increment Physical Read (APRD)....72
Table 20 – Node-Addressed Physical Read (NPRD) 72
Table 21 – Logical Read (LRD) 73
Table 22 – Broadcast Read (BRD) 73
Table 23 – Auto Increment Physical Write (APWR) 74
Table 24 – Node-Addressed Physical Write (NPWR)....74
Table 25 – Logical Write (LWR)....75
Table 26 – Broadcast Write (BWR)....75
Table 27 – Logical ReadWrite (LRW)....76

Table 28 – Auto Increment Physical Read Multiple Write (ARMW).......................................76
Table 29 – DL Information...77
Table 30 – Configured Station Address 78
Table 31 – DL Control 78
Table 32 – DL Status 79
Table 33 – AL Control 80
Table 34 – AL Status 80
Table 35 – PDI Control 81
Table 36 – 8 Bit Micro Controller Interface...82
Table 37 – 16 Bit Micro Controller Interface...82
Table 38 – AL Event 83
Table 39 – Channel CRC Fault Counter..85
Table 40 – Watchdog Divider 85
Table 41 – PDI Watchdog 86
Table 42 – Sync Manager Channel Watchdog...86
Table 43 – Sync Manager Watchdog Status 87
Table 44 – Slave Information Interface Area 91
Table 45 – Slave Information Interface Size..91
Table 46 – Slave Information Interface Control/Status...92
Table 47 – Actual Slave Information Interface Address 93
Table 48 – Actual Slave Information Interface Data 93
Table 49 – Fieldbus Memory Management Unit (FMMU) Part 1 94
Table 50 – Fieldbus Memory Management Unit (FMMU) Part 2 95
Table 51 – Sync Manager Structure Part 1 96
Table 52 – Sync Manager Structure Part 2 97
Table 53 – Distributed Clock Structure Part 1 98
Table 54 – Distributed Clock Structure Part 2 99
Table 55 – State transitions and local management services..110
Table 56 – AL management and ESM service primitives 111
Table 57 – AL Control Service..113
Table 58 – AL State Changed Service 114
Table 59 – AL State Acknowledge Service...114
Table 60 – Mailbox Write 119
Table 61 – Mailbox Read 120
Table 62 – Initiate SDO Download Expedited...131
Table 63 – Initiate SDO Download Normal...132
Table 64 – Download SDO Segment 133
Table 65 – Initiate SDO Upload Expedited..136
Table 66 – Initiate SDO Upload Normal 137
Table 67 – Upload SDO Segment..138
Table 68 – Abort SDO Transfer 139
Table 69 – SDO Information Service 141
Table 70 – Get OD List 142

Table 71 – Get Object Description 143
Table 72 – Get Entry Description 144
Table 73 – RxPDO Service 149
Table 74 – TxPDO Service 150
Table 75 – RxPDO Remote Transmission Request 151
Table 76 – TxPDO Remote Transmission Request 152
Table 77 – Emergency Service 154
Table 78 – Initiate Ethernet over EtherCAT 156
Table 79 – Ethernet over EtherCAT Fragment 157
Table 80 – FoE Read Request 161
Table 81 – FoE Write Request 162
Table 82 – FoE Data Request 162
Table 83 – FoE Ack Request 163
Table 84 – FoE Busy Request 163
Table 85 – FoE Error Request 164
Table 86 – FoE Busy Request 164
Table 87 – AL Control Request Indication Description 167
Table 88 – AL Control Request Confirmation 168
Table 89 – AL State Changed Request Indication 168
Table 90 - AL State Acknowledge Request Indication 169
Table 91 - AL State Response Confirmation 169
Table 92 - General Mailbox Header 182
Table 93 – CANopen over EtherCAT 183
Table 94 – CANopen Initiate SDO Download Expedited Request 184
Table 95 – Initiate SDO Download Expedited 185
Table 96 – Initiate SDO Download Normal Request 186
Table 97 – Initiate SDO Download Normal Response 187
Table 98 – Download SDO Segment Request 188
Table 99 – Download SDO Segment Response 189
Table 100 – Initiate SDO Upload Expedited Request 190
Table 101 – Initiate SDO Upload Expedited Response 191
Table 102 – Initiate SDO Upload Normal Request 192
Table 103 – Initiate SDO Upload Normal Response 193
Table 104 – Upload SDO Segment Request 194
Table 105 – Upload SDO Segment Response 194
Table 106 – Abort SDO Transfer Request 195
Table 107 – SDO Abort Codes 196
Table 108 – SDO Information Service 197
Table 109 – Get OD List Request 198
Table 110 – Get OD List Response 199
Table 111 – Get Object Description Request 200
Table 112 – Get Object Description Response 201
Table 113 – Get Entry Description Request 202

Table 114 – Get Object Description Response 203
Table 115 – Emergency Request 204
Table 116 – Emergency Error Codes 204
Table 117 – Error Code 205
Table 118 – Diagnostic Data 205
Table 119 – Sync Manager Length Error 206
Table 120 – Sync Manager Address Error 206
Table 121 – Sync Manager Settings Error 206
Table 122 – RxPDO Transmission via Mailbox 209
Table 123 – TxPDO Transmission via Mailbox 209
Table 124 – RxPDO Remote Transmission Request 210
Table 125 – TxPDO Remote Transmission Request 210
Table 126 – TxPDO Remote Transmission Request Command 211
Table 127 – Object Dictionary Structure 212
Table 128 – Object Definitions 212
Table 129 – Data Type Area 212
Table 130 – CoE Communication Area 214
Table 131 – Device Type 215
Table 132 – Manufacturer Device Name 216
Table 133 – Manufacturer Hardware Version 216
Table 134 – Manufacturer Software Version 216
Table 135 – Identity Object 217
Table 136 – EtherCAT Configured Station Address 218
Table 137 – Virtual MAC Address 218
Table 138 – IP Address Info 218
Table 139 – Receive PDO Mapping 220
Table 140 – Transmit PDO Mapping 220
Table 141 – Sync Manager Communication Type 221
Table 142 – Sync Manager Channel 0 (Mailbox Receive) 223
Table 143 – Sync Manager Channel 1 (Mailbox Send) 223
Table 144 – Sync Manager Channel 2 (Process Data Output) 224
Table 145 – Sync Manager Channel 3 (Process Data Input) 224
Table 146 – Sync Manager Channel 4-32 225
Table 147 – EtherCAT Physical Memory Access 226
Table 148 – Initiate EoE Request 227
Table 149 – EoE Fragment Request 228
Table 150 – Read Request 229
Table 151 – Write Request 230
Table 152 – Data Request 230
Table 153 – Ack Request 231
Table 154 – Error Request 232
Table 155 – Busy Request 233

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

Real-time Ethernet control automation technology (EtherCAT™[1])

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patents concerning the Fieldbus Memory Management Unit (FMMU) given in subclauses 7.3.1.5 and 8.2.1.5.

IEC takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured the IEC that he is willing to negotiate licenses under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from:

EtherCAT Technology Group
Ostendstr. 196
D-90492 Nürnberg

Phone ++ 49 911 540 56 20
Fax ++49 911 540 56 29
email info@ethercat.org
Internet www.ethercat.org

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. The IEC shall not be held responsible for identifying any or all such patent rights.

———————

1 EtherCAT™ is the registered trade name of Beckhoff, Verl. This information is given for the convenience of users of this specification and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance to this PAS does not require use of the trade name EtherCAT™. Use of the trade name EtherCAT™ requires the permission of the trade name holder.

A PAS is a technical specification not fulfilling the requirements for a standard but made available to the public .

IEC-PAS 62407 has been processed by subcommittee 65C: Digital communications, of IEC technical committee 65: Industrial-process measurement and control.

The text of this PAS is based on the following document:	This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document
Draft PAS	**Report on voting**
65C/355/NP	65C/371/RVN

Following publication of this PAS, the technical committee or subcommittee concerned will transform it into an International Standard.

It is intended that the content of this PAS will be incorporated in the future new edition of the various parts of IEC 61158 series according to the structure of this series.

This PAS shall remain valid for an initial maximum period of three years starting from 2005-06. The validity may be extended for a single three-year period, following which it shall be revised to become another type of normative document or shall be withdrawn.

Real-time Ethernet control automation technology (EtherCAT^{TM})

1 Introduction

This PAS is provided by the EtherCAT Technology Group (ETG). This PAS follows the general structure and terms of IEC 61158 series in order to support a future migration to this standard. If required, the clauses 7 – 10 can be separated and turned into parts 2-6 according to the IEC 61158 structure.

2 Scope

EtherCAT is a real-time Ethernet technology especially suitable for communication between control systems and peripheral devices like I/O systems, drives, sensors and actuators.

This PAS is related to the ISO/IEC 7498-1.

This PAS gives an introductory overview on the EtherCAT technology and then specifies

- The Physical Layer selection from ISO/IEC 8802-3 and IEEE 802.3ae

- The Data Link Layer services and protocols in addition to those specified in ISO/IEC 8802-3.

NOTE: ISO/IEC 8802-3 is not replaced, but enhanced in order to allow for improved real time behaviour.

- The Application Layer services and protocols

3 Normative References

IEC 61131 (all parts), *Programmable controllers*

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

IEC 61158, *Digital data communications for measurement and control — Fieldbus for use in industrial control systems*

IEC 61158-2, *Physical layer specification and service definition*

IEC 61158-3, *Data link service definition*

IEC 61158-4, *Data link protocol specification*

IEC 61158-5, *Application layer service definition*

IEC 61158-6, *Application layer protocol specification*

IEC 61784-1, *Digital data communications for measurement and control – Part 1: Profile sets for continuous and discrete manufacturing relative to fieldbus use in Industrial control systems*

ISO/IEC 7498-1, *Information processing systems – Open Systems Interconnection – Basic Reference Model: The basic model (OSI)*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information between systems – Local and metropolitan area networks – Specific requirements – Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

IEEE 802.3ae-2002: *Information technology – Local and metropolitan area networks – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications – Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation.*

IEEE 1588-2002, *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*

ANSI/TIA/EIA-644-A-2001, *Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits*

RFC 768, *User Datagram Protocol Specification*

RFC 791, *Internet Protocol Specification*

RFC 793, *Transmission Control Protocol Specification*

EN 50325-4, *Industrial communications subsystem based on ISO 11898 (CAN) for controller-device interfaces – Part 4: CANopen*

4 Terms, Definitions and Abbreviations

For the purposes of this PAS, some of the following terms and definitions have been compiled from the referenced documents. The terms and definitions of ISO/IEC 7498-1, ISO/IEC 8802-3 and IEC 61588 series shall be fully valid, unless otherwise stated.

4.1 IEC 61158 definitions

For the purpose of this specification the following definitions of IEC 61158 apply

4.1.1
Application
Function or data structure for which data is consumed or produced [IEC 61158-5:2003]

4.1.2
Application Objects
Multiple object classes that manage and provide a run time exchange of messages across the network and within the network device [IEC 61158-5:2003]

4.1.3
Bit
Unit of information consisting of a 1 or a 0. This is the smallest data unit that can be transmitted [IEC 61158-4:2003]

4.1.4
Client
1) An object which uses the services of another (server) object to perform a task

2) An initiator of a message to which a server reacts [IEC 61158-4:2003]

4.1.5
Clock Synchronization
Represents a sequence of interactions to synchronize the clocks of all time receivers by a time master [IEC 61158-3:2003]

4.1.6
Communication Objects
Components that manage and provide a run time exchange of messages across the network [IEC 61158-5:2003]

4.1.7
Connection
Logical binding between two application objects within the same or different devices [IEC 61158-4:2003]

4.1.8
Connector
Coupling device employed to connect the medium of one circuit or communication element with that of another circuit or communication element [IEC 61158-2:2003]

4.1.9
Cyclic
Term used to describe events which repeat in a regular and repetitive manner [IEC 61158-3:2003]

4.1.10
Cyclic Redundancy Check (CRC)
Residual value computed from an array of data and used as a representative signature for the array [IEC 61158-4:2003]

4.1.11
Data
Generic term used to refer to any information carried over a Fieldbus [IEC 61158-3:2003]

4.1.12
Data Consistency
Means for coherent transmission and access of the input- or output-data object between and within client and server [IEC 61158-5:2003]

4.1.13
Device
Physical entity connected to the fieldbus composed of at least one communication element (the network element) and which may have a control element and/or a final element (transducer, actuator, etc.) [IEC 61158-2:2003]

4.1.14
Device Profile
A collection of device dependent information and functionality providing consistency between similar devices of the same device type [IEC 61158-5:2003]

4.1.15
Diagnosis Information
All data available at the server for maintenance purposes [IEC 61158-5:2003]

4.1.16
Error
Discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition [IEC 61158-4:2003]

4.1.17
Error Class
General grouping for related error definitions and corresponding error codes [IEC 61158-5:2003]

4.1.18
Error Code
Identification of a specific type of error within an error class [IEC 61158-5:2003]

4.1.19
Event
An instance of a change of conditions [IEC 61158-5:2003]

4.1.20
Frame
Denigrated synonym for DLPDU [IEC 61158-3:2003]

4.1.21
Index
Address of an object within an application process [IEC 61158-5:2003]

4.1.22
Interface
Shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate [IEC 61158-5:2003]

4.1.23
Master
A device that controls the data transfer on the network and initiates the media access of the slaves by sending messages and that constitutes the interface to the control system [IEC 61158-2:2003]

4.1.24
Medium
Cable, optical fibre, or other means by which communication signals are transmitted between two or more points [IEC 61158-2:2003]

NOTE In this specification "media" is used only as the plural of medium.

4.1.25
Network
A set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways [IEC 61158-6:2003]

4.1.26
node
end-point of a link in a network or a point at which two or more links meet [derived from IEC 61158-2]

4.1.27 None

4.1.28
Object
Abstract representation of a particular component within a device [IEC 61158-4:2003]

NOTE An object can be

1) an abstract representation of the capabilities of a device. Objects can be composed of any or all of the following components:

a) data (information which changes with time);

b) configuration (parameters for behavior);

c) methods (things that can be done using data and configuration).

2) a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior.

4.1.29
Server
Object which provides services to another (client) object [IEC 61158-4:2003]

4.1.30
Service
Operation or function than an object and/or object class performs upon request from another object and/or object class [IEC 61158-5:2003]

4.1.31
Slave
DL-entity accessing the medium only after being initiated by the preceding slave or master [IEC 61158-3:2003]

4.2 Definitions from other standards

For the purpose of this specification the following definitions apply

4.2.1
Frame
a unit of data transmission on an ISO/IEC8802-3 MAC (Media Access Control) that conveys a protocol data unit (PDU) between MAC Service users [IEEE Std. 802.1Q – 1998]

4.2.2
Message
Ordered series of octets intended to convey information [derived from ISO 2382-16.02.01]

NOTE Normally used to convey information between peers at the application layer.

4.2.3
switch
a MAC bridge as defined in IEEE 802.1D:1998

4.3 EtherCAT Definitions

For the purpose of this specification the following EtherCAT specific definitions apply

4.3.1
Application Object
Data structure in the object dictionary containing application data

4.3.2
Basic Slave
Slave device that supports only physical addressing of data

4.3.3
Communication Object
Data structure in the object dictionary containing communication parameters

4.3.4
Data Type
Relation between values and encoding for data of that type. For this specification the data type definitions of IEC 61131-3 shall be applied.

4.3.5
Data Type Object
Entry in the object dictionary indicating a data type

4.3.6
Fieldbus Memory Management Unit
Function that establishes one or several correspondences between logical addresses and physical memory

4.3.7
Fieldbus Memory Management Unit Channel
Single entity of the fieldbus memory management unit: one correspondence between a coherent logical address space and a coherent physical memory location.

4.3.8
Full Slave
Slave Device that supports both physical and logical addressing of data

4.3.9
Index
Address of an object within the object dictionary

4.3.10
Mapping
Correspondence between application objects and process data objects

4.3.11
Mapping Parameters
Set of values defining the correspondence between application objects and process data objects

4.3.12
Object Dictionary
Data structure addressed by Index and Sub-index that contains data type objects, communication objects and application objects

4.3.13
Process Data
Data object containing application objects designated to be transferred cyclically or acyclically for the purpose of processing

4.3.14
Process Data Object
Data structure described by mapping parameters containing one or several process data entities

4.3.15
Sub-index
Sub-Address of an object within the object dictionary

4.3.16
Sync Manager

4.3.17
Sync Manager Channel

4.3.18
Node
Single DL-entity

4.4 Abbreviations

Table 1 – Abbreviations

Term or Abbreviation	**Definition or meaning**
AL	Application Layer
APDU	Application Protocol Data Unit
APRD	Auto Increment Physical Read
APWR	Auto Increment Physical Write
ARMW	Auto Increment Physical Read Multiple Write
BRD	Broadcast Read
BWR	Broadcast Write
CiA	CAN in Automation
CoE	CANopen over EtherCAT
CRC	cyclic redundancy check
CSMA-CD	Carrier Sense Multiple Access with Collision Detection
DA	Destination MAC Address
DL	Data Link layer (as a prefix)
DLL	DL-Layer
EoE	Ethernet over EtherCAT
ESC	EtherCAT Slave Controller
ESM	EtherCAT State Machine
FCS	frame check sequence

FMMU	Fieldbus Memory Management Unit
FoE	File Access over EtherCAT
FW	Firmware
HDR	Header
ID	Identifier
IP	Internet Protocol
LAN	Local Area Network
LRD	Logical Read
LRW	Logical ReadWrite
LVDS	Low Voltage Differential Signals
LWR	Logical Write
MAC	Media Access Control
NPRD	Node-Addressed Physical Read
NPWR	Node-Addressed Physical Write
PDI	Process Data Interface
PDO	Process Data Object
PDU	Protocol Data Unit
PTP	Precision Time Protocol [IEC 61588:2004]
Rx	Receive
SDO	Service Data Object
SII	Slave Information Interface
Sync Mngr	Sync Manager
TCP	Transmission Control Protocol
Tx	Transmit
UDP	User Datagram Protocol
WKC	Working Counter

5 Technology Overview

EtherCAT is a Real Time Ethernet technology that aims to maximize the utilization of the full duplex Ethernet bandwidth. Medium access control employs the Master/Slave principle, where the master node (typically the control system) sends the Ethernet frames to the slave nodes, which extract data from and insert data into these frames.

5.1 Operating principle

From an Ethernet point of view, an EtherCAT segment is a single Ethernet device, which receives and sends standard ISO/IEC 8802-3 Ethernet frames. However, this Ethernet device is not limited to a single Ethernet controller with downstream microprocessor, but may consist of a large number of EtherCAT slave devices. These process the incoming frames directly and extract the relevant user data, or insert data and transfer the frame to the next EtherCAT slave device. The last EtherCAT slave device within the segment sends the fully processed frame back, so that it is returned by the first slave device to the master as response frame.

This procedure utilizes the full duplex mode of Ethernet: both communication directions are operated independently. Direct communication without switch between a master device and an EtherCAT segment consisting of one or several slave devices may be established.

5.2 Topology

The topology of a communication system is one of the crucial factors for the successful application in automation. The topology has significant influence on the cabling effort, diagnostic features, redundancy options and hot-plug-and-play features.

The star topology commonly used for Ethernet leads to enhanced cabling effort and infrastructure costs. Especially for automation applications a line or tree topology is preferable.

The EtherCAT slave node arrangement represents an open ring bus. At the open end, the master device sends frames, either directly or via Ethernet switches, and receives them at the other end after they have been processed. All frames are relayed from the first node to the next ones. The last node returns the telegram back to the master. Utilizing the full duplex capabilities of Ethernet, the resulting topology is a physical line.

Branches, which in principle are possible anywhere, can be used to enhance the line structure into a tree structure from. A tree structure supports very simple wiring; individual branches, for example, can branch into control cabinets or machine modules, while the main line runs from one module to the next.

5.2.1 Telegram processing principles

In order to achieve maximum performance, the Ethernet frames should be processed directly “on the fly”. If it is implemented this way, the slave node recognizes relevant commands and executes them accordingly while the frames are already passed on.

NOTE EtherCAT can be implemented using standard Ethernet controllers without direct processing. The influence of the forwarding mechanism implementation on communication performance will be detailed in conjunction with IEC 61784-2.

The nodes have an addressable memory that can be accessed with read or write services, either each node consecutively or several nodes simultaneously. Several EtherCAT telegrams can be embedded within an Ethernet frame, each telegram addressing a cohesive data section. As noted in Figure 1, the EtherCAT telegrams are either transported:

a) directly in the data area of the Ethernet frame,

b) or within the data section of an UDP datagram transported via IP.

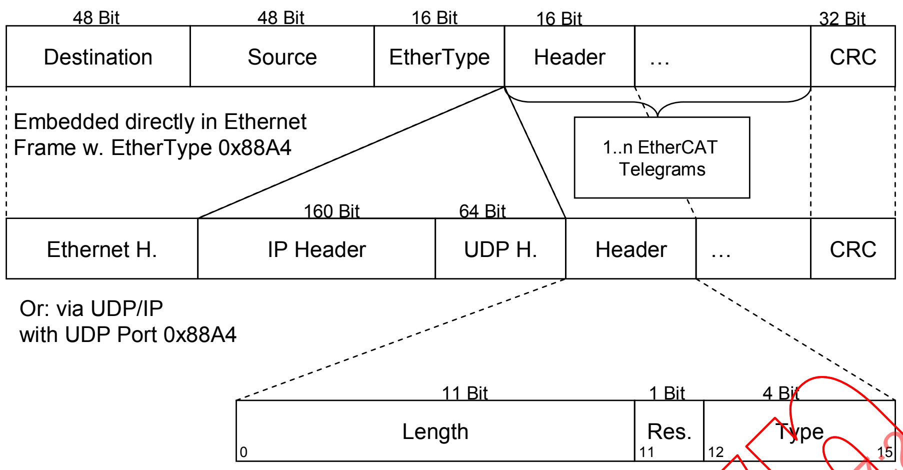


Figure 1 – Frame Structure

Variant a) is limited to one Ethernet subnet, since associated frames are not relayed by routers. For machine control applications this usually does not represent a constraint. Multiple EtherCAT segments can be connected to one or several switches. The Ethernet MAC address of the first node within the segment is used for addressing the EtherCAT segment.

NOTE: further addressing details are given in the Data Link Layer (clause 7)

Variant b) via UDP/IP generates a slightly larger overhead (IP and UDP header), but for less time-critical applications such as building automation it allows using IP routing. On the master side any standard UDP/IP implementation can be used.

5.3 Data Link Layer overview

Several EtherCAT nodes can be addressed individually via a single Ethernet frame carrying several EtherCAT telegrams. Compared with one frame per node this leads to a better utilization of the Ethernet bandwidth. However, for e.g. a 2 channel digital input node with just 2 bit of user data, the overhead of a single EtherCAT telegram is still excessive.

Therefore the EtherCAT slave nodes may also support logical address mapping. The process data can be inserted anywhere within a logical address space. If an EtherCAT telegram is sent that contains read or write services for a certain process image area located at the corresponding logical address, instead of addressing a particular EtherCAT node, the nodes insert the data at or extract the data from the right place within the process data, as noted in Figure **2**.

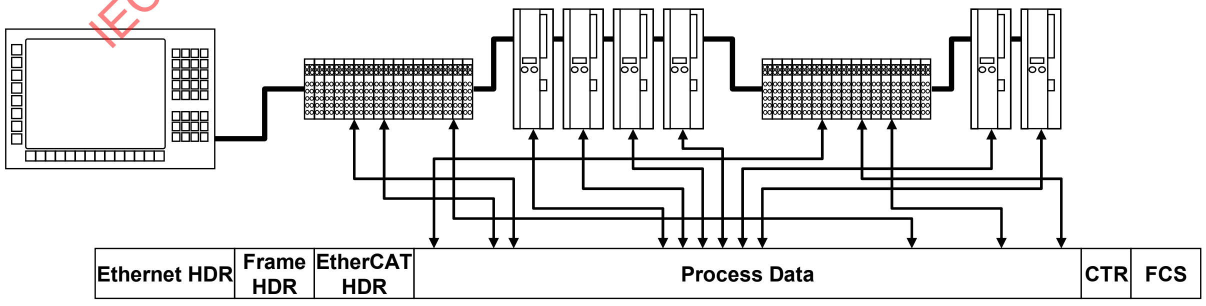


Figure 2 – Nodes map data directly in frame

All other nodes that also detect an address match with the process image also insert their data, so that many nodes can be addressed simultaneously with a single EtherCAT telegram. The master can assemble completely sorted logical process images via a single EtherCAT telegram. Additional mapping is no longer required in the master, so that the process data can be assigned directly to the different control tasks. Each task can create its own process image and exchange it within its own timeframe. The physical order of the EtherCAT nodes is completely arbitrary and is only relevant during the first initialization phase.

The logical address space is 2^{32} Bytes = 4 GB. EtherCAT can be considered to be a serial backplane for automation systems that enables connection to distributed process data for both large and very small automation devices. Using a standard Ethernet controller and a standard Ethernet cable, a very large number of I/O channels without practical restrictions on the distribution can be connected to automation devices, which can be accessed with high bandwidth, minimum delay and near-optimum usable data rate. At the same time, devices such as fieldbus scanners can be connected as well, thus preserving existing technologies and standards.

5.4 Error Detection Overview

EtherCAT checks by the Ethernet Frame Check Sequence (FCS) whether a frame was transmitted correctly. Since one or several slaves modify the frame during the transfer, the FCS is recalculated by each slave. If a slave detects a checksum error, the slave does not repair the FCS but flags the master by setting the corresponding status bit, so that a fault can be located precisely.

When reading data from or writing data to an EtherCAT telegram, the addressed slave increments a working counter (WKR) positioned at the end of each EtherCAT telegram. Analyzing the working counter allows the master to check if the expected number of nodes has processed the corresponding EtherCAT telegram.

5.5 Parameter and Process Data Handling Introduction

Industrial communication systems have to meet different requirements in terms of the data transmission characteristics. Parameter data is transferred acyclically and in large quantities, whereby the timing requirements are relatively non-critical, and the transmission is usually triggered by the control system. Diagnostic data is also transferred acyclically and event-driven, but the timing requirements are more demanding, and the transmission is usually triggered by a peripheral device.

Process data, on the other hand, is typically transferred cyclically with different cycle times. The timing requirements are most stringent for process data communication. EtherCAT supports a variety of services and protocols to meet these differing requirements. The application layer services and protocols are specified in clauses 9 and 10 of this specification.

5.6 Node Reference Model

5.6.1 Mapping onto OSI basic reference model

EtherCAT is described using the principles, methodology and model of ISO/IEC 7498-1. The OSI model provides a layered approach to communications standards, whereby the layers can be developed and modified independently. The EtherCAT specification defines functionality from top to bottom of a full OSI stack, and some functions for the users of the stack. Functions of the intermediate OSI layers, layers 3 – 6, are consolidated into either the EtherCAT Data Link layer or the EtherCAT Application layer. Likewise, features common to users of the Fieldbus Application Layer may be provided by the EtherCAT Application layer to simplify user operation, as noted in Figure 3.

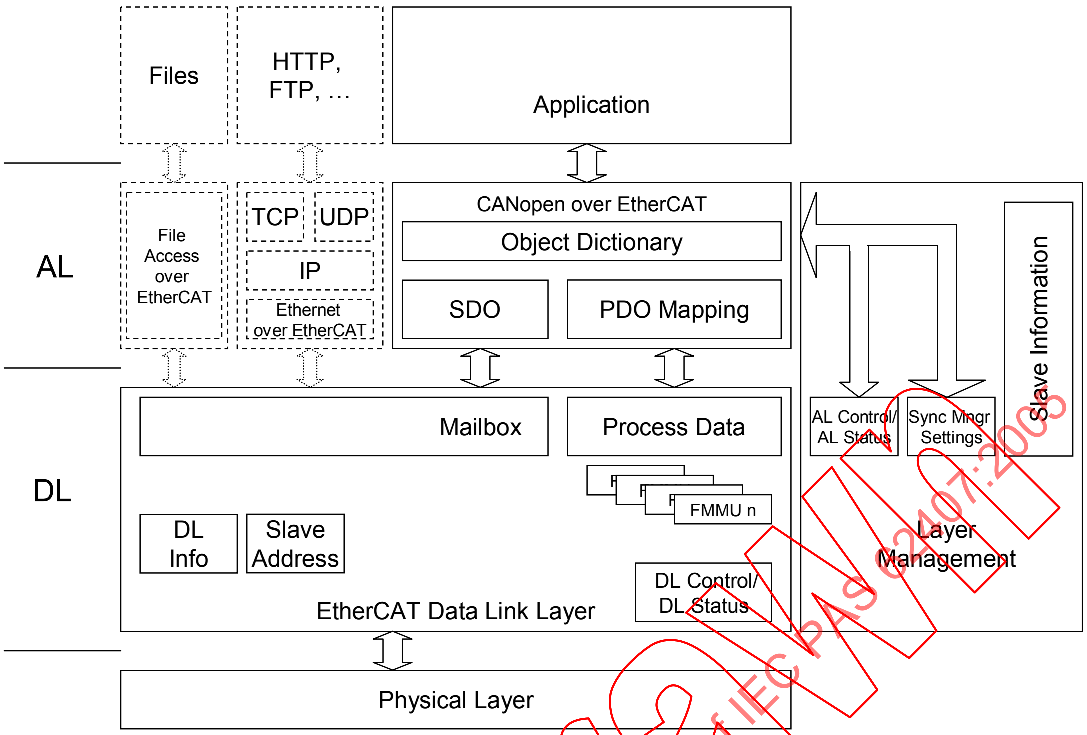


Figure 3 – Slave Node Reference Model

5.6.2 Physical Layer

The physical layer receives data units from the Data Link Layer, encodes the bits into signals, and transmits the resulting physical signals to the transmission medium connected to the node. Signals are then received by the subsequent node and decoded, and then the data units are passed to the data link layer of the receiving device.

5.6.3 Data Link Layer

The data link layer provides basic time critical support for data communications among devices connected via EtherCAT. The term “time-critical” is used to describe applications having a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

The data link layer has the task to compute, compare and generate the frame check sequence and provide communications by extracting data from and/or including data into the Ethernet frame. This is done depending on the data link layer parameters which are stored at pre-defined memory locations. The application data is made available to the application layer in physical memory, either in a mailbox configuration or within the process data section.

5.6.4 Application Layer

The Application Layer is designed to support the conveyance of time-critical application requests and responses among devices in an automation environment. EtherCAT allows for several optional service and protocol families to co-exist within the same device. In this way, generic Ethernet communication such as TCP/UDP/IP-based protocols may be implemented side by side with real time communication, file access protocols, and other generic protocols and services. One protocol technology applied to EtherCAT is CANopen[2], which defines SDOs, PDOs and the Object Dictionary Structure to manage the parameters.

[2] CANopen is a CAN based technology specified in EN 50325-4.

5.7 Data types and encoding rules

5.7.1 General description of data types and encoding rules

To be able to exchange meaningful data, the format of this data and it's meaning have to be known by the producer and consumer(s). This specification models this by the concept of data types.

The encoding rules define the representation of values of data types and the transfer syntax for the representations. Values are represented as bit sequences. Bit sequences are transferred in sequences of octets (bytes). For numerical data types the encoding is little endian style as shown in Table 3.

The data types and encoding rules shall be valid for the DL services and protocols as well as for the AL services and protocols specified. The encoding rules for the Ethernet frame are specified in ISO/IEC 8802-3.

5.7.2 Data type definitions

A data type determines a relation between values and encoding for data of that type. Names are assigned to data types in their type definitions. The syntax of data and data type definitions is specified in Table 2.

Table 2 –Data Type Syntax

Descriptor	Definition
data_definition	type_name data_name
type_definition	constructor type_name
array_constructor	'ARRAY' '[' length ']' 'OF' type_name
structure_constructor	'STRUCT' 'OF' component_list
component_list	component { ',' component }
component	type_name component_name
basic_constructor	'BOOLEAN' | 'VOID' bit_size | 'INTEGER' bit_size | 'UNSIGNED' bit_size | 'REAL32' | 'REAL64' | 'NIL'
bit_size	'1' | '2' | <...> | '64'
length	positive_integer
data_name	symbolic_name
type_name	symbolic_name
component_name	symbolic_name
symbolic_name	letter { ['_'] (letter | digit) }
positive_integer	('1' | '2' | <...> | '9') { digit }
letter	'A' | 'B' | <...> | 'Z' | 'a' | 'b' | <...> | 'z'
digit	'0' | '1' | <...> | '9'

Recursive definitions are not allowed.

5.7.3 Bit sequences

5.7.3.1 Definition of bit sequences

A bit can take the values 0 or 1. A bit sequence *b* is an ordered set of 0 or more bits. If a bit sequence *b* contains more than 0 bits, they are denoted as b_j, $j \geq 0$. Let b_0, ..., b_{n-1} be bits, *n* a positive integer. Then

$$b = b_0\ b_1 \ldots b_{n-1}$$

is called a bit sequence of length |b| = n. The empty bit sequence of length 0 is denoted ε.

NOTE: Examples: 10110100_b, 1_b, 101_b, etc. are bit sequences.

The inversion operator (¬) on bit sequences assigns to a bit sequence

$$b = b_0\ b_1 \ldots b_{n-1}$$

the bit sequence

$$\neg b = \neg b_0\ \neg b_1 \ldots \neg b_{n-1}$$

Here ¬0 = 1 and ¬1 = 0 on bits.

The basic operation on bit sequences is concatenation.

Let $a = a_0 \ldots a_{m-1}$ and $b = b_0 \ldots b_{n-1}$ be bit sequences. Then the concatenation of *a* and *b*, denoted *ab*, is

$$ab = a_0 \ldots a_{m-1}\ b_0 \ldots b_{n-1}$$

NOTE: Example: (10)(111) = 10111 is the concatenation of 10 and 111.

The following holds for arbitrary bit sequences *a* and *b*:

$$|ab| = |a| + |b|$$

and

$$\varepsilon a = a\varepsilon = a$$

5.7.3.2 Transfer syntax for bit sequences

For transmission across EtherCAT a bit sequence is reordered into a sequence of octets. Here and in the following hexadecimal notation is used for octets. Let $b = b_0 \ldots b_{n-1}$ be a bit sequence with $n \leq 64$. Denote *k* a non-negative integer such that $8(k - 1) < n \leq 8k$. Then *b* is transferred in *k* octets assembled as shown in Table 3. The bits b_i, $i \geq n$ of the highest numbered octet are do not care bits.

Octet 1 is transmitted first and octet *k* is transmitted last. Hence the bit sequence is transferred as follows across the network:

$$b_7, b_6, \ldots, b_0, b_{15}, \ldots, b_8, \ldots$$

Table 3 – Transfer Syntax for bit sequences

octet number	**1.**	**2.**	**k.**
	$b_7 .. b_0$	$b_{15} .. b_8$	$b_{8k-1} .. b_{8k-8}$

NOTE: Example:

Bit 9	...	Bit 0
10_b	0001_b	1100_b
0x2	0x1	0xC
		= 0x21C

The bit sequence b = b_0 .. b_9 = 0011 1000 01_b represents an UNSIGNED10 with the value 0x21C and is transferred in two octets: First 0x1C and then 0x02.

5.7.4 Basic data types

5.7.4.1 Type Definition

For basic data types “type_name” equals the literal string of the associated constructor (aka *symbolic_name*), e.g.,

BOOLEANTable 4Table 4BOOLEAN

is the type definition for the BOOLEAN data type.

5.7.4.2 NIL

Data of basic data type NIL is represented by ε.

5.7.4.3 Boolean

Data of basic data type BOOLEAN attains the values TRUE or FALSE. The values are represented as bit sequences of length 1. The value TRUE (res. FALSE) is represented by the bit sequence 1 (res. 0).

5.7.4.4 Void

Data of basic data type VOIDn is represented as bit sequences of length *n* bit. The value of data of type VOIDn is undefined. The bits in the sequence of data of type VOIDn must either be specified explicitly or else marked "do not care".

NOTE: Data of type VOIDn is useful for reserved fields and for aligning components of compound values on octet boundaries.

5.7.4.5 Unsigned Integer

Data of basic data type UNSIGNEDn has values in the non-negative integers. The value range is 0, ..., 2^n-1. The data is represented as bit sequences of length *n*. The bit sequence

$$b = b_0 \ldots b_{n-1}$$

is assigned the value

$$\text{UNSIGNEDn(b)} = b_{n-1}\, 2^{n-1} + \ldots + b_1\, 2^1 + b_0\, 2^0$$

The bit sequence starts on the left with the least significant byte.

NOTE: Example: The value 266 = $10A_h$ with data type UNSIGNED16 is transferred in two octets across the bus, first $0A_h$ and then 01_h.

The following UNSIGNEDn data types are transferred as specified in Table 4:

Table 4 – Transfer syntax for data type UNSIGNEDn

octet number	1.	2.	3.	4.	5.	6.	7.	8.
UNSIGNED8	$b_7..b_0$							
UNSIGNED16	$b_7..b_0$	$b_{15}..b_8$						
UNSIGNED24	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$					
UNSIGNED32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
UNSIGNED40	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$			
UNSIGNED48	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$		
UNSIGNED56	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	
UNSIGNED64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.7.4.6 Signed Integer

Data of basic data type INTEGERn has values in the integers. The value range is from -2^{n-1} to $2^{n-1}-1$. The data is represented as bit sequences of length n. The bit sequence

$$b = b_0 .. b_{n-1}$$

is assigned the value

$$\text{INTEGERn}(b) = b_{n-2}\, 2^{n-2} + ... + b_1\, 2^1 + b_0\, 2^0 \qquad \text{if } b_{n-1} = 0$$

and, performing two's complement arithmetic,

$$\text{INTEGERn}(b) = - \text{INTEGERn}(\hat{}b) - 1 \quad \text{if } b_{n-1} = 1$$

Note that the bit sequence starts on the left with the least significant bit.

NOTE: Example: The value –266 = $FEF6_h$ with data type INTEGER16 is transferred in two octets across the network, first $F6_h$ and then FE_h.

The following INTEGERn data types are transferred as specified in :

Table 5 –Transfer syntax for data type INTEGERn

octet number	1.	2.	3.	4.	5.	6.	7.	8.
INTEGER8	$b_7..b_0$							
INTEGER16	$b_7..b_0$	$b_{15}..b_8$						
INTEGER24	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$					
INTEGER32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
INTEGER40	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$			
INTEGER48	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$		
INTEGER56	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	
INTEGER64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.7.4.7 Floating-Point Numbers

Data of basic data types *REAL32* and *REAL64* have values in the real numbers.

The data type *REAL32* is represented as bit sequence of length 32.

The data type *REAL64* is represented as bit sequence of length 64.

A bit sequence of length 32 either has a value (finite non-zero real number, ±0, ± _) or is NaN (not-a-number). The bit sequence

$$b = b_0 \ldots b_{31}$$

is assigned the value (finite non-zero number)

$$\mathrm{REAL32}(b) = (-1)^S\, 2^{E\,-\,127}\,(1 + F)$$

Here

$S = b_{31}$ is the sign.

$E = b_{30}\, 2^7 + \ldots + b_{23}\, 2^0$, $0 < E < 255$, is the un-biased exponent.

$F = 2^{-23}\ (b_{22}\, 2^{22} + \ldots + b_1\, 2^1 + b_0\, 2^0)$ is the fractional part of the number.

$E = 0$ is used to represent ± 0. $E = 255$ is used to represent infinities and NaN's.

The bit sequence starts on the left with the least significant bit.

NOTE: Example:

$6.25 = 2^{E\ -127}\ (1 + F)$ with

$E = 129 = 2^7 + 2^0$ and

$F = 2^{-1} + 2^{-4} = 2^{-23}(2^{22} + 2^{19})$ hence the number is represented as:

S	E	F
b_{31}	$b_{30} .. b_{23}$	$b_{22} .. b_0$
0	100 0000 1$_b$	100 1000 0000 0000 0000 0000$_b$

$6.25 = b_0 .. b_{31}$ = 0000 0000 0000 0000 0001 0011 0000 0010$_b$

It is transferred in the following order:

Table 6 – Transfer syntax for data type INTEGERn

octet number	1.	2.	3.	4.
REAL32	0x00	0x00	0xC8	0x40
	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$

5.7.5 Extended data types

5.7.5.1 Octet String

The data type OCTET_STRING*length* is defined below; *length* is the length of the octet string.

ARRAY [length] OF UNSIGNED8 OCTET_STRING*length*

5.7.5.2 Visible String

The data type VISIBLE_STRING*length* is defined below. The admissible values of data of type *VISIBLE_CHAR* are 0_h and the range from 0x20 to 0x7E. The data are interpreted as 7-bit coded characters. *length* is the length of the visible string.

UNSIGNED8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VISIBLE_STRING*length*

There is no 0x0 necessary to terminate the string.

6 Contents of Part 2: Physical Layer

6.1 Overview

Like with most ISO/IEC 8802-3 based technologies there is a choice of physical layers. For EtherCAT systems, the following full duplex physical layer technologies shall be used:

- 100BASE-TX
- 100BASE-FX
- LVDS

Inside an EtherCAT segment, any combination of these physical layer technologies may be used. Changeovers from one physical layer to another one are supported.

NOTE: In Project 61918 IEC/SC 65C JWG 10 (Industrial Cabling) currently specifies installation practise including connectors, topologies, wiring infrastructure components and methodologies. The results of this work will be honoured by EtherCAT in an appropriate way. Therefore the current edition of this specification does not cover this kind of definitions.

6.2 100BASE-TX

100BASE-TX is an electrical physical layer system specified in ISO/IEC 8802-3. It uses two pairs of Category 5 balanced cabling as specified by ISO/IEC 11801.

This Physical Layer Variant may be used both internally (within an EtherCAT segment) and externally.

6.3 100BASE-FX

100BASE-FX is a fibre optical physical layer system specified in ISO/IEC 8802-3. It uses two multi mode fibres.

This Physical Layer Variant may be used both internally (within an EtherCAT segment) and externally.

6.4 LVDS

The LVDS (Low Voltage Differential Signals) is a physical layer system specified in IEEE 803-3ae-2002 and TIA/EIA-644.

This Physical Layer Variant shall be used within an EtherCAT segment only.

NOTE: In EtherCAT systems, the LVDS Physical Layer is also referred to as E-Bus.

7 Contents of Part 3: Data Link Layer Service definition

7.1 Overview

7.1.1 Relation to ISO/IEC8802-3

This part specifies data link layer services in addition to those specified in ISO/IEC 8802-3.

7.1.2 Frame Structure

An EtherCAT Ethernet frame contains one or several EtherCAT telegrams (As noted in Figure 4), each addressing individual devices and/or memory areas. The EtherCAT frame is recognized by the combination of the EtherType 0x88A4[3] and the corresponding EtherCAT frame header or, when transported via UDP/IP (As noted in Figure 5) by the UDP port 34980=0x88A4[4] and the EtherCAT frame header.

Each EtherCAT telegram consists of an EtherCAT header, the data area and a subsequent counter area (working counter), which is incremented by all EtherCAT nodes that were addressed by the EtherCAT telegram and have exchanged associated data.

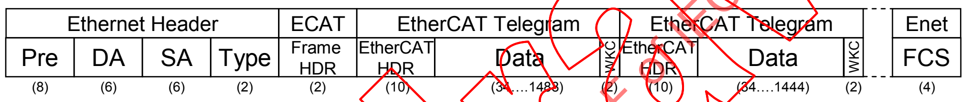


Figure 4: EtherCAT Telegrams embedded in Ethernet Frame

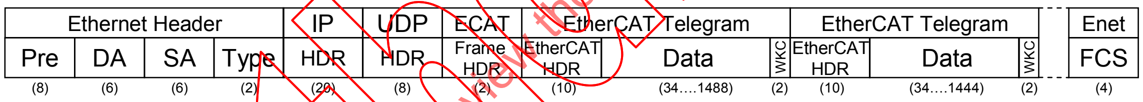


Figure 5: EtherCAT Telegrams embedded in UDP/IP

7.1.3 EtherCAT Modes

7.1.3.1 Open Mode

In Open Mode, one or several EtherCAT Segments may be connected to a standard switching device. The first slave device within an EtherCAT segment then has an ISO/IEC 8802-3 MAC address representing the entire segment. This Segment Address Slave Device exchanges destination address field and source address field within the Ethernet frame. If EtherCAT is transported via UDP, this device also exchanges Source and Destination IP Address and the UDP source and destination port numbers in order to ensure that the response frame fully satisfies UDP/IP protocol standards.

[3] The EtherType 0x88A4 was assigned for EtherCAT by the IEEE Registration Authority.

[4] The UDP Port 34980 was assigned for EtherCAT by the Internet Assigned Numbers Authority (IANA).

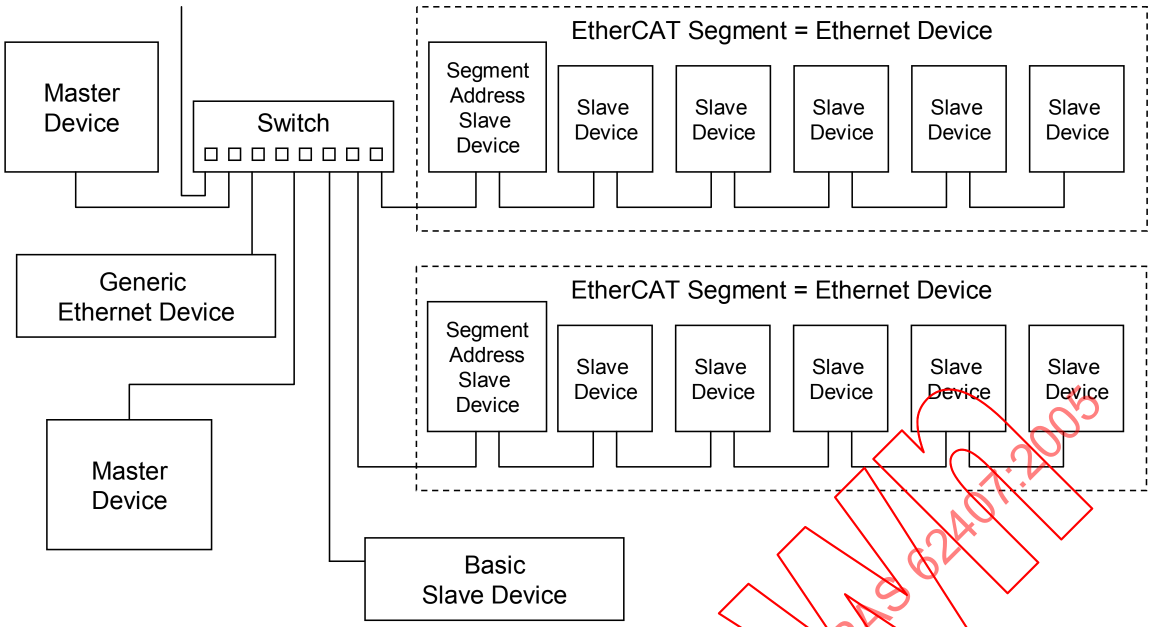


Figure 6: EtherCAT Segments in Open Mode

7.1.3.2 Direct Mode

In Direct Mode, one EtherCAT segment is connected directly to the standard Ethernet port of the controlling or master device.

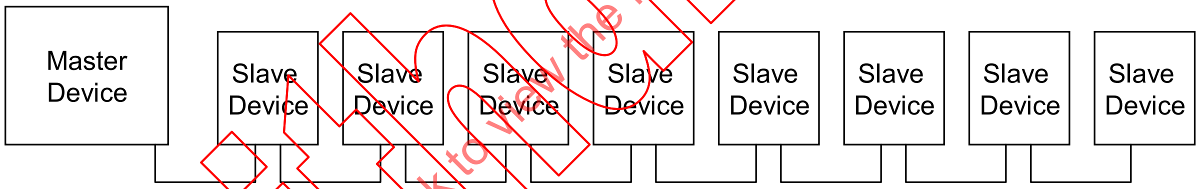


Figure 7: EtherCAT Segment in Direct Mode

7.1.4 Segment represents logical ring

In logic terms the slave arrangement within an EtherCAT segment represents an open ring bus. At the open end, the master device inserts Ethernet frames, either directly or via standard Ethernet switches, and receives them at the other end after they have been processed. All frames are relayed from the first slave device to the next ones. The last slave device returns the frame back to the master. Since Ethernet cabling is full duplex and thus bi-directional (separate Tx and Rx lines), and since all EtherCAT slave devices can also transfer in the reverse direction, the result is a physical line.

Frames should be processed directly "on the fly" by the slave devices according to their physical sequence within the ring structure. In this case, the slave device recognizes relevant commands and executes them accordingly while the frames (delayed by only a few bits) are already passed on. Data extraction and insertion should be done within the data link layer hardware. Then it is independent of the response times of any microprocessors that may be connected.

Branches are possible in the EtherCAT Segment at any location, since a branch does not break the logical ring. Branches can be used to build a flexible tree structure and thus allows for very simple wiring.

7.1.5 Addressing

Different addressing modes are supported for EtherCAT slaves, as noted in Figure 8. The EtherCAT header within the EtherCAT telegram contains a 32 bit address, which is used for node or logical addressing.

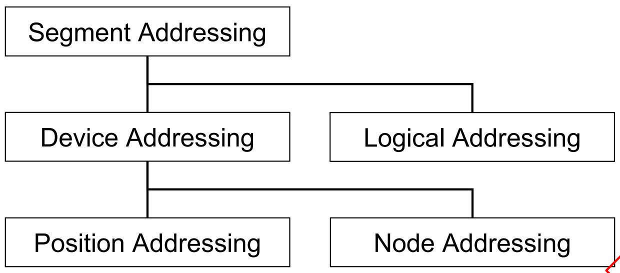


Figure 8: Addressing Mode Overview

7.1.5.1 Segment addressing

For segment addressing the MAC Addresses according to ISO/IEC 8802.3 shall be used.

7.1.5.2 Device addressing

With this address mode, the 32 bit address within the EtherCAT telegram is split into 16 bit slave device address and 16 bit physical address within the slave device, thus leading to 64535 slave device addresses with 64 kB local address space each. With node addressing each EtherCAT telegram invariably addresses one single slave device. This mode is most suitable for transferring parameter data. There are two different device addressing mechanisms:

- position addressing, and
- node addressing.

7.1.5.2.1 Position addressing

Position addressing shall be used to address each slave device via its physical position within the segment. Each slave device increments the 16 bit address field while the telegram passes through; the slave device receiving an address field with value 0 is the one being addressed. Due to the mechanism employed the slave device address in position addressing is referred to as auto increment address.

NOTE: In previous editions of this specification position addressing was known as Auto Increment Addressing.

For example, if the tenth slave device in the segment is to be addressed, the master device sends a telegram with position addressing with the start address value of -9, which is incremented by one by each device. Position addressing is used during the start-up phase, in which the master assigns configured node addresses to the slaves. Subsequently, they can be addressed irrespective of their physical position in the segment.

This mechanism has the advantage that no slave node addresses have to be set manually at the slaves.

7.1.5.2.2 Node addressing

With Node addressing, the slaves are addressed via a configured node address assigned by the master during the start-up phase. This ensures that, even if the segment topology is changed or devices are added or remove after initial setup or during operation, the slave devices can still be addressed via the same configured address.

The slave device address in configured addressing is referred to as node address.

7.1.5.3 Logical Addressing

For logical addressing within a segment the entire 32 bit address field is used for one address value. With logical addressing, slaves are not addressed individually, but instead a section of the segment wide 4 GB logical address space is addressed. This section may be used by any number of slaves.

The logical addressing mode is particularly suitable for transferring cyclic process data.

7.1.5.3.1 FMMU introduction

Fieldbus Memory Management Units (FMMU) handle the local assignment of physical slave memory addresses to logical segment wide addresses, as noted in Figure 9.

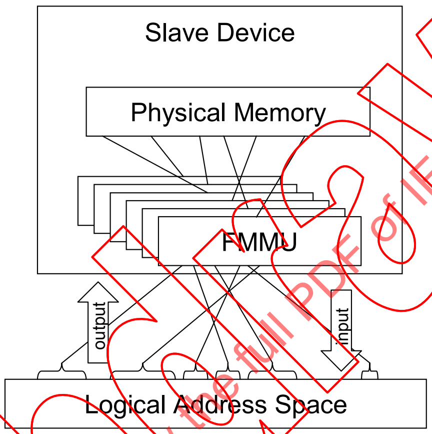


Figure 9: Fieldbus Memory Management Unit Overview

The configuration of the FMMU entities is made available by the master device and transferred to the slave devices during the start-up phase. For each FMMU entity, the following items are configured: a logical, bit-oriented start address, a physical memory start address, a bit length, and a type that specifies the direction of the mapping (input or output). Any data within the memory of a slave device can thus be mapped bit-wise to any logical address.

When a telegram with logical addressing is received, the slave device checks whether one of its FMMU entities shows an address match. If appropriate, it inserts data at the associated position of the data field into the telegram (input type) or extracts data from the associated position of the telegram (output type). Telegrams can therefore be assembled flexibly and optimized to the requirements of the control application.

7.1.6 Slave Classification

There is a differentiation between Full Slaves, supporting all addressing modes and Basic Slaves, supporting only a subset of the addressing modes. Master devices may support the Basic Slave functionality to allow for direct communication with another master device. Slave devices should support the Full Slave functionality.

7.1.6.1 Full Slave

The Full Slave shall support

- logical addressing,
- position addressing and

- node addressing.

Thus Full Slave devices need FMMU and address increment functionality. Full Slaves may support segment addressing. Full Slaves that support segment addressing are called Segment Address Slave Device.

Only Full Slaves can be connected within an EtherCAT Segment.

7.1.6.2 Basic Slave

Basic Slave devices shall support node addressing and segment Addressing.

7.2 EtherCAT Services

The Data Link Layer specifies EtherCAT services for reading, writing and read-writing data from physical memory within the slaves.

NOTE: For simplification the expression "reads memory" is used instead of "reads data from physical memory". Equivalently the expression "writes memory" is used instead of "writes data to physical memory".

7.2.1 Read

With the Read service a master reads memory from one or many slaves.

7.2.1.1 Auto Increment Physical Read (APRD)

With the APRD service a master reads memory from one slave selected by the segment position of the slave.

Table 7 – Auto Increment Physical Read (APRD)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Auto Increment Address	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Result		Mandatory
Index		Mandatory
Data		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Auto Increment Address

The slave will be addressed by its position in the segment. Each slave shall increment this parameter, the slave who receives the value zero of this parameter shall respond on this service.

Physical Memory Address

This parameter shall contain the start address in the physical memory of the slave where the data to be read is stored.

Length

This parameter shall contain the size in bytes of the data to be read.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Data

This parameter shall contain the read data.

Working Counter

This parameter shall be incremented by one if the data was successfully read.

7.2.1.2 Node-Addressed Physical Read (NPRD)

With the NPRD service a master reads memory from one slave selected by the slave's configured station address.

Table 8 – Node-Addressed Physical Read (NPRD)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Configured Station Address	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Result		Mandatory
Index		Mandatory
Data		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Station Address

The slave which configured station address matches to this parameter shall respond on this service

Physical Memory Address

This parameter shall contain the start address in the physical memory of the slave where the data to be read is stored.

Length

This parameter shall contain the size in bytes of the data to be read.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Data

This parameter shall contain the read data.

Working Counter

This parameter shall be incremented by one if the data was successfully read.

7.2.1.3 Logical Read (LRD)

With the LRD service a master reads memory from one or many slaves selected by a logical address.

Table 9 – Logical Read (LRD)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Logical Memory Address	Mandatory	
Length	Mandatory	
Result		Mandatory
Index		Mandatory
Data		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Logical Memory Address

This parameter shall contain the start address in the logical memory where the data to be read is located. All slaves which have one or more address matches of the requested logical memory area (logical memory address and length) in their FMMUs shall respond on this service.

Length

This parameter shall contain the size in bytes of the data to be read.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Data

This parameter shall contain the read data. Each slave which detects an address match of the requested logical memory area will put the data of the corresponding physical memory area in the correct part of this parameter.

Working Counter

This parameter shall be incremented by one by all slaves which detect an address match of the requested logical memory area.

7.2.1.4 Broadcast Read (BRD)

With the BRD service a master reads a physical memory area, which will be wired or by all slaves.

Table 10 – Broadcast Read (BRD)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Result		Mandatory
Index		Mandatory
Data		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Physical Memory Address

This parameter shall contain the start address in the physical memory where the data to be read is stored. Each slave who supports the requested physical memory area (physical memory address and length) shall respond on this service.

Length

This parameter shall contain the size in bytes of the data to be read.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Data

This parameter shall contain the data read by wired or. Each slave shall make a logical or with the received data and its read data.

Working Counter

This parameter shall be incremented by one by all slaves which made the wired or of the requested data.

7.2.2 Write

With the Write services a master writes data to memory of one or many slaves.

7.2.2.1 Auto Increment Physical Write (APWR)

With the APWR service a master writes memory to one slave selected by the slave's position in the segment.

Table 11 – Auto Increment Physical Write (APWR)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Auto Increment Address	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Data	Mandatory	
Result		Mandatory
Index		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Auto Increment Address

The slave will be addressed by its position in the segment. Each slave shall increment this parameter, the slave who receives the value zero of this parameter shall respond to this service.

Physical Memory Address

This parameter shall contain the start address in the physical memory of the slave where data to be written is stored.

Length

This parameter shall contain the size in bytes of the data to be written.

Data

This parameter shall contain the data to be written.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Working Counter

This parameter shall be incremented by one if the data was successfully written.

7.2.2.2 Node-Addressed Physical Write (NPWR)

With the NPWR service a master writes memory to one slave selected by the slave's configured station address.

Table 12 – Node-Addressed Physical Write (NPWR)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Configured Station Address	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Data	Mandatory	
Result		Mandatory
Index		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Station Address

The slave which configured station address matches to this parameter shall respond to this service

Physical Memory Address

This parameter shall contain the start address in the physical memory of the slave where the data to be written is stored.

Length

This parameter shall contain the size in bytes of the data to be written.

Data

This parameter shall contain the data to be written.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Working Counter

This parameter shall be incremented by one if the data was successfully written.

7.2.2.3 Logical Write (LWR)

With the LWR service a master writes memory to one or many slaves selected by a logical address.

Table 13 – Logical Write (LWR)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Logical Memory Address	Mandatory	
Length	Mandatory	
Data	Mandatory	
Result		Mandatory
Index		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Logical Memory Address

This parameter hall contain the start address in the logical memory where the data to be written is located. All slaves which have one or more address matches of the requested logical memory area (logical memory address and length) in their FMMUs shall respond to this service.

Length

This parameter shall contain the size in bytes of the data to be written.

Data

This parameter shall contain the data to be written. Each slave which detects an address match of the requested logical memory area will put the data of the correct part of this parameter in the corresponding physical memory area.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Working Counter

This parameter shall be incremented by one by all slaves who detect an address match of the requested logical memory area.

7.2.2.4 Broadcast Write (BWR)

With the BRD service a master writes a physical memory area to all slaves.

Table 14 – Broadcast Write (BWR)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Data	Mandatory	
Result		Mandatory
Index		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Physical Memory Address

This parameter shall contain the start address in the physical memory where the data to be written is stored. Each slave who supports the requested physical memory area (physical memory address and length) shall respond to this service.

Length

This parameter shall contain the size in bytes of the data to be written.

Data

This parameter shall contain the data to be written.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Working Counter

This parameter shall be incremented by one by all slaves which write data in their physical memory.

7.2.3 ReadWrite

With the ReadWrite service a master writes and reads memory of one or many slaves.

7.2.3.1 Logical ReadWrite (LRW)

With the LRW service a master writes and reads memory to one or many slaves selected by a logical address. A slave device can retrieve data from the EtherCAT frame (write operation) and put data in the same EtherCAT frame (read operation).

Table 15 – Logical ReadWrite (LRW)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Logical Memory Address	Mandatory	
Length	Mandatory	
Data	Mandatory	
Result		Mandatory
Index		Mandatory
Data		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Logical Memory Address

This parameter shall contain the start address in the logical memory where the data to be read or written is located. All slaves which have one or more address matches of the requested logical memory area (logical memory address and length) in their FMMU shall respond to this service.

Length

This parameter shall contain the size in bytes of the data to be read and written.

Data

This parameter shall contain the data to be written. Each slave which detects an address match of the requested logical memory area will put the data of the correct part of this parameter in the corresponding physical memory area.

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Data

This parameter shall contain the read data. Each slave which detects an address match of the requested logical memory area will put the data of the corresponding physical memory area in the correct part of this parameter.

Working Counter

This parameter shall be incremented by one by all slaves which detect an address match of the requested logical memory area (for read or write operation).

7.2.3.2 Auto Increment Physical Read Multiple Write (ARMW)

With the ARMW service a master reads data memory of one slave selected by the slave's position in the segment and writes data to the same memory of all other slaves following after the addressed slave.

Table 16 – Auto Increment Physical Read Multiple Write (ARMW)

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Index	Mandatory	
Auto Increment Address	Mandatory	
Physical Memory Address	Mandatory	
Length	Mandatory	
Data	Mandatory	
Working Counter	Mandatory	
Result		Mandatory
Index		Mandatory
Data		Mandatory
Working Counter		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Index

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

Auto Increment Address

The slave which executes the read operation will be addressed by its position in the segment. Each slave shall increment this parameter the slave who receives the value zero of this parameter shall perform the read operation. All other slaves perform a write operation at the requested physical address, if this address is present.

Physical Memory Address

This parameter shall contain the start address in the physical memory of the slave where the data to be read or written is stored.

Length

This parameter shall contain the size in bytes of the data to be read or written.

Data

This parameter shall contain the data to be written.

Working Counter

This parameter shall be incremented by one by all slaves which detect an address match of the requested physical memory area (for read or write operation).

Result

The result shall convey the service specific parameters of the service response.

Index

The parameter Index shall not change in the response.

Data

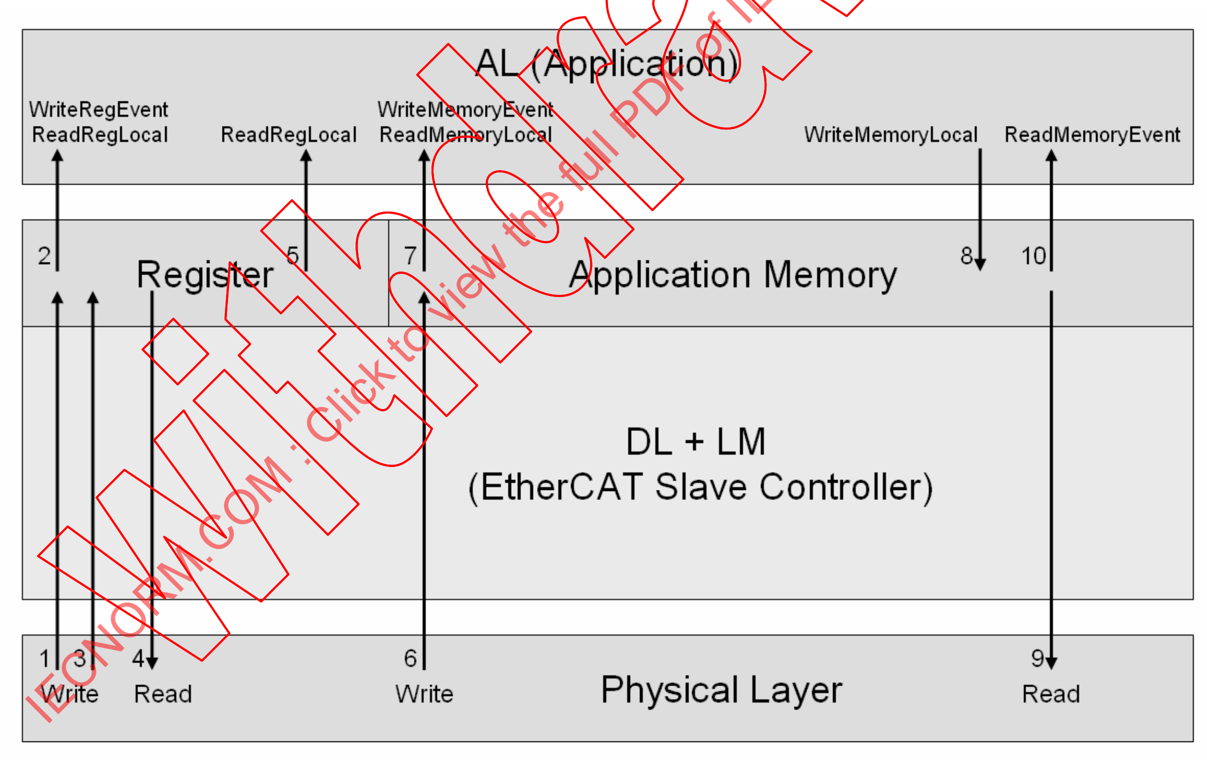
This parameter shall contain the data to be read.

Working Counter

This parameter shall be incremented by one by every slave performing the read or the write operation.

7.3 EtherCAT Attributes

The EtherCAT attributes are related to the physical memory of an EtherCAT slave, which can be read or written from the EtherCAT master. The physical memory consists of registers and application memory. The register area contains information for configuration, management and device identification in the DL. The use of the application memory is defined by the application layer (AL).



An EtherCAT Write service to the register area (1) may (depending on the written register) result in a Write Register Event primitive in the AL, followed by a Read Register Local primitive from the AL to get the written value (2). Otherwise the EtherCAT Write service will only access the register area without informing the AL (3).

An EtherCAT Read service to the register area will only access the register area without informing the AL (4).

The AL can read the register area with Read Register Local primitive at any time (5).

An EtherCAT Write service to the application memory area (6) will result in a Write Memory Event primitive in the AL, followed by a Read Memory Local primitive from the AL to get the written value (7).

The AL will write the application memory area with a Write Memory Local primitive (8). The application memory area will be read by the master with an EtherCAT Read service (9) which will result in a Read Memory Event primitive in the AL (10) to indicate that the memory area has been read and can be written by the AL again.

7.3.1 Register

7.3.1.1 Management

7.3.1.1.1 DL Information

The DL Information Registers contain type, version and supported resources of the EtherCAT Slave Controller (ESC).

Parameter

Type

This parameter shall contain the type of the EtherCAT Slave Controller.

Revision

This parameter shall contain the revision of the EtherCAT Slave Controller.

Build

This parameter shall contain the build number of the EtherCAT Slave Controller.

Number of supported FMMU channels

This parameter shall contain the number of supported FMMU channels (or entities) of the EtherCAT Slave Controller.

Number of supported Sync Manager channels

This parameter shall contain the number of supported Sync Manager channels (or entities) of the EtherCAT Slave Controller.

RAM size

This parameter shall contain the RAM size supported by the EtherCAT Slave Controller.

FMMU Bit Operation Not Supported

This parameter shall contain the information, if the FMMU in the EtherCAT Slave Controller supports bit operations or not.

7.3.1.1.2 Configured Station Address

The Configured Station Address Register contains the station address of the slave which will be set by the master during start up to activate the NPRD and NPWR service in the EtherCAT Slave Controller.

Parameter

Configured Station Address

This parameter shall contain the configured station address of the EtherCAT Slave Controller.

7.3.1.1.3 DL Control

The DL Control Register is used to control the operation of the EtherCAT Slave Controller by the master.

Parameter

Channel A Loop Control

This parameter shall contain the information if there is an automatic activation of the loop between the receive and transmission port of channel A in case of no signal detection.

Channel B Loop Control

This parameter shall contain the information if there is an automatic activation of the loop between the receive and transmission port of channel B in case of no signal detection.

7.3.1.1.4 DL Status

The DL Status Register is used to confirm the operation requests of the master in the DL Control Register and to indicate the status of the EtherCAT Slave Controller.

Parameter

PDI operational

This parameter shall contain the information if an application hardware is connected to the process data interface of the EtherCAT Slave Controller.

PDI Watchdog Status

This parameter shall contain the status of the process data interface watchdog.

Channel A Loop Status

This parameter shall contain the information if there is a loop between the receive and the transmission port on channel A.

Channel A Signal Detection

This parameter shall contain the information if there is signal detected at the receive port on channel A.

Channel B Loop Status

This parameter shall contain the information if there is a loop between the receive and the transmission port on channel B.

Channel B Signal Detection

This parameter shall contain the information if there is signal detected at the receive port on channel B.

7.3.1.1.5 AL Management Interface

The following diagram shows the primitives between master, DL and AL in case of a successful write sequence to the AL Control register.

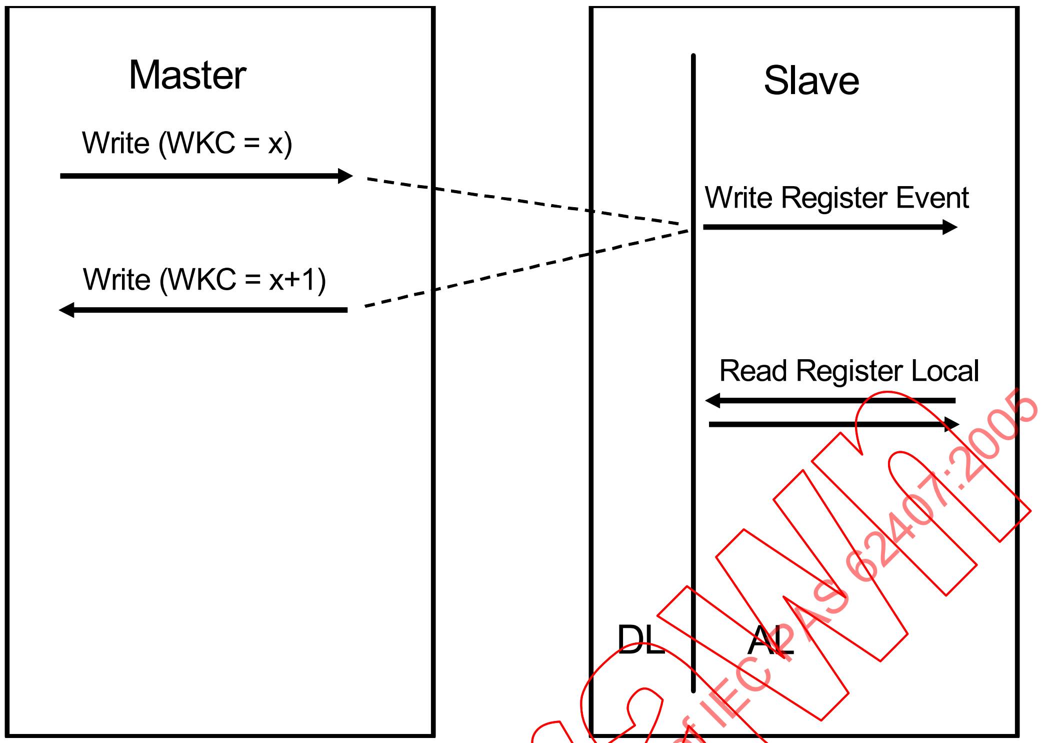


Figure 11: Successful Write sequence to AL Control Register

The master sends an EtherCAT write service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave write the received data in the register area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller and the AL controller reads the AL Control register.

The following diagram shows the primitives between master, DL and AL in case of a successful read sequence to the AL Status register.

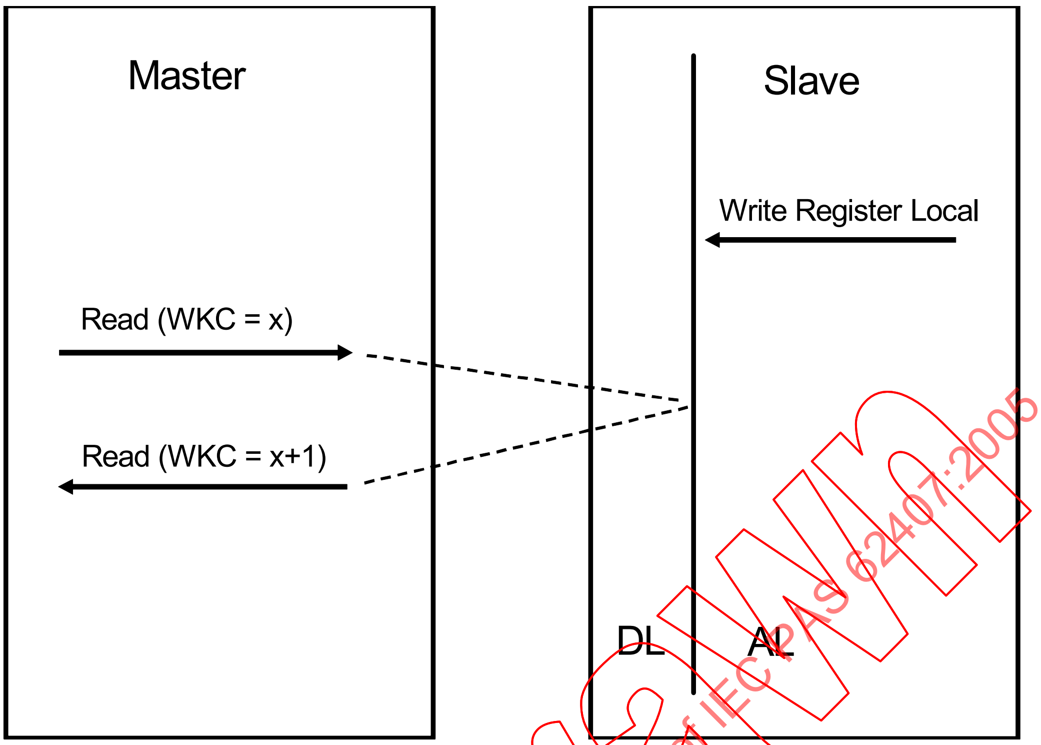


Figure 12: Successful Read sequence to the AL Status Register

The AL of the slave writes the AL Status register locally. The master sends an EtherCAT read service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave sends the data from the register area and increments the Working Counter (WKC = x + 1).

7.3.1.1.5.1 AL Control

The AL Control Register is used to request the AL state from the master to the slave or to acknowledge a change in the AL state of the slave.

Parameter

AL State

This parameter shall contain the requested or acknowledged AL state.

Acknowledge

This parameter shall contain the information if the AL state shall be requested or acknowledged by the master.

Application Specific

This parameter shall contain application specific information requested by the master.

7.3.1.1.5.2 AL Status

The AL Status Register is used to confirm or to indicate a change in the AL state from the slave to the master.

Parameter

AL State

This parameter shall contain the changed or confirmed AL state.

Change

This parameter shall contain the information if the AL state is changed or confirmed by the slave.

Application Specific

This parameter shall contain application specific information confirmed by the slave.

7.3.1.1.6 PDI Control

The PDI Control Register is used to control the hardware interface behaviour of the EtherCAT Slave Controller to the application layer. The PDI Control Register shall be loaded during Power-On or due to a Reload operation from the Slave Information Interface (E²PROM).

Parameter

PDI Type

This parameter shall contain the process data interface type describing the hardware interface behaviour of the EtherCAT Slave Controller.

AL Management emulation

This parameter shall contain the information if the AL Management operation is done by the application layer or will be emulated by the EtherCAT Slave Controller by copying the AL Control Register into the AL Status Register.

7.3.1.1.7 PDI Configuration

The PDI Configuration Register depends on the PDI type and contains special configuration settings of the selected hardware interface of the EtherCAT Slave Controller. The PDI Configuration Register shall be loaded during Power-On or due to a Reload operation from the Slave Information Interface (E²PROM).

Parameter

8 Bit Micro Controller Interface

This parameter shall contain the configuration settings for an 8 Bit Micro Controller Interface.

16 Bit Micro Controller Interface

This parameter shall contain the configuration settings for a 16 Bit Micro Controller Interface.

7.3.1.1.8 AL Event

The AL Event Register is used to indicate an AL event to the AL controller. The AL event shall be acknowledged if the corresponding event source is read by the AL controller.

Parameter

AL Control Register Event

This parameter is set if an EtherCAT Write service to the AL Control Register is received (WriteRegEvent) and is reset when the AL Control Register is read by the AL controller (ReadRegLocal).

Sync Manager Watchdog Event

This parameter is set if a watchdog of a Sync Manager times out. The corresponding Sync Manager can be found by reading the Sync Manager parameters.

Sync Manager Channel Access Events

This parameter is set if an EtherCAT Write service to an application memory area configured as write by the master (WriteMemoryEvent) or an EtherCAT Read service to an application memory area configured as read by the master (ReadMemoryEvent) is received and will be reset when the application memory area will be read (ReadMemoryLocal) or written (WriteMemoryLocal) again.

7.3.1.2 Statistics

7.3.1.2.1 Channel CRC Fault Counter

The Channel CRC Fault Counter Registers contain information about the CRC faults detected on the receive channels A and B.

Parameter

Channel A CRC fault counter

This parameter shall contain the number of CRC faults detected on channel A.

Channel B CRC fault counter

This parameter shall contain the number of CRC faults detected on channel B.

7.3.1.3 Watchdogs

7.3.1.3.1 Watchdog Divider

The system clock of the EtherCAT Slave Controller is divided by the Watchdog Divider.

Parameter

Watchdog Divider

This parameter shall contain the value by how much the system clock will be divided.

7.3.1.3.2 PDI Watchdog

The application controller is monitored with the value of the PDI Watchdog. Each access from the application controller to the EtherCAT Slave Controller shall reset this watchdog.

Parameter

PDI Watchdog

This parameter shall contain the watchdog to monitor the PDI.

7.3.1.3.3 Sync Manager Channel Watchdog

Each Sync Manager channel is monitored with the value of its Sync Manager Channel Watchdog. Each write access to the application memory area configured in the Sync Manager channel shall reset this watchdog.

Parameter

Sync Manager Channel Watchdog

This parameter shall contain the watchdog to monitor the corresponding Sync Manager Channel.

7.3.1.3.4 Sync Manager Watchdog Status

The status of each Sync Manager channel watchdog is included in the Sync Manager Watchdog Status.

Parameter

Sync Manager Watchdog Status

This parameter shall contain the watchdog status of all Sync Manager channel watchdogs.

7.3.1.4 Slave Information Interface

The slave information interface[5] includes the boot configuration data and the application information data. The boot configuration data contain the settings to initialize the hardware interface of the EtherCAT Slave Controller during power on (PDI Control, PDI Configuration). The application information data contain the Product Code that can be used by the master to find the corresponding configuration file for the device. The slave information interface registers define the access to the slave information interface which shall be supported by the EtherCAT Slave Controller (ESC).

7.3.1.4.1 Slave Information Interface Content

The following parameter shall be stored in the slave information interface.

Parameter

PDI Control Init

This parameter shall contain the initialization value for the parameter PDI Control.

PDI Configuration Init

This parameter shall contain the initialization value for the parameter PDI Configuration.

PDI Size Init

This parameter shall contain the initialization value for the parameter PDI Size.

Vendor ID Init

This parameter shall contain the initialization value for the parameter Vendor ID of the application layer.

Product Code Init

This parameter shall contain the initialization value for the parameter Product Code of the application layer.

Revision Number Init

This parameter shall contain the initialization value for the parameter Revision Number of the application layer.

Serial Number Init

This parameter shall contain the initialization value for the parameter Serial Number of the application layer.

Execution Delay Init

This parameter is reserved for future extensions.

Channel A Delay Init

This parameter is reserved for future extensions.

Channel B Delay Init

This parameter is reserved for future extensions.

Bootstrap Receive Mailbox Offset

This parameter shall contain the receive mailbox offset in the physical memory for the Bootstrap state.

Bootstrap Receive Mailbox Size

This parameter shall contain the receive mailbox size in the physical memory for the Bootstrap state.

5 The Slave Information Interface can be implemented in a variety of ways, e.g. using a serial E²PROM.

Bootstrap Send Mailbox Offset

This parameter shall contain the send mailbox offset in the physical memory for the Bootstrap state.

Bootstrap Send Mailbox Size

This parameter shall contain the send mailbox size in the physical memory for the Bootstrap state.

7.3.1.4.2 Slave Information Interface Size

The Slave information interface Size Register contains the size of the slave information interface. The Slave Information Interface Size Register shall be loaded during Power-On or due to a Reload operation from the Slave Information Interface (E²PROM).

Parameter

Size

This parameter shall contain the size of the slave information interface.

7.3.1.4.3 Slave Information Interface Control/Status

With the Slave Information Interface Control/Status Register the read or write operation to the slave information interface is controlled.

Parameter

Slave Information Interface Write Access

This parameter shall contain the information, if a write access to the slave information interface is allowed.

Slave Information Interface Address Algorithm

This parameter shall contain the information, if the protocol to the Slave Information Interface contains one or two address bytes.

Read Operation

This parameter will be written from the master to start the read operation of 32 bits in the slave information interface. This parameter will be read from the master to check if the read operation is finished.

Write Operation

This parameter will be written from the master to start the write operation of 16 bits in the slave information interface. This parameter will be read from the master to check if the write operation is finished.

Reload Operation

This parameter will be written from the master to start the reload operation of 32 bits in the slave information interface. This parameter will be read from the master to check if the reload operation is finished.

Write Error

This parameter shall contain the information if the last write access to the slave information interface was successful.

Busy

This parameter contains the information if an access operation is ongoing.

7.3.1.4.4 Actual Slave Information Interface Address

The Actual Slave Information Interface Address Register contains the actual address in the slave information interface which is accessed by the next read or write operation (by writing the Slave Information Interface Control/Status Register).

Parameter

Address

This parameter shall contain the address which is accessed by the next read or write operation.

7.3.1.4.5 Actual Slave Information Interface Data

The Actual Slave Information Interface Data Register contains the data (16 bit) to be written in the slave information interface with the next write operation or the read data (32 bit) with the last read operation.

Parameter

Data

The master will write this parameter with the data (16 bit) to be written in the slave information interface with the next write operation. The master will receive the last read data (32 bit) from the slave information interface when reading this parameter.

7.3.1.5 Fieldbus Memory Management Unit (FMMU)

The Fieldbus Memory Management Unit (FMMU) converts logical addresses into physical addresses by the means of internal address. Thus, FMMUs allow one to use logical addressing for data segments that span several slave devices: one telegram addresses data within several arbitrarily distributed devices. The FMMUs support bit wise mapping. A DLE may contain several FMMU entities (channels). Each FMMU channel maps one cohesive logical address space to one cohesive physical address space of the slave.

The FMMU consists of up to 16 channels. Each channel describes one memory translation between the logical memory of the EtherCAT bus and the physical memory of the slave.

Parameter

Logical Start Address

This parameter shall contain the start address in bytes in the logical memory area of the memory translation.

Logical Start Bit

This parameter shall contain the bit offset of the logical start address.

Logical End Bit

This parameter shall contain the bit offset of the logical end address.

Physical Start Address

This parameter shall contain the start address in bytes in the physical memory area of the memory translation.

Physical Start Bit

This parameter shall contain the bit offset of the physical start address.

Length

This parameter shall contain the size in bytes of the memory translation.

Read Enable

This parameter shall contain the information if a read operation (physical memory is source, logical memory is destination) is enabled.

Write Enable

This parameter shall contain the information if a write operation (logical memory is source, physical memory is destination) is enabled.

Channel Enable

This parameter shall contain the information if the memory translation is active or not.

7.3.1.6 Sync Manager

The Sync Manager controls the access to the application memory. Each channel defines a consistent area of the application memory.

Parameter

Physical Start Address

This parameter shall contain the start address in bytes in the physical memory of the consistent application memory area.

Length

This parameter shall contain the size in bytes of the consistent application memory area.

Buffer Type

This parameter shall contain the information if the consistent application memory area is of queued access Type or an Buffered Access Type.

Direction

This parameter shall contain the information if the consistent application memory area is read or written by the master.

AL Event Enable

This parameter shall contain the information if an AL event is generated if there is new data available in the consistent application memory area which was written by the master (direction write) or if the new data from the AL controller was read by the master (direction read).

Watchdog Enable

This parameter shall contain the information if the monitoring of an access to the consistent application memory area is enabled.

Write Event

This parameter shall contain the information if the consistent application memory (direction write) has been written by the master and the AL Event Enable parameter is set.

Read Event

This parameter shall contain the information if the consistent application memory (direction read) has been read by the master and the AL Event Enable parameter is set.

Watchdog Trigger

This parameter shall contain the information if an access to the consistent application memory has been done and the watchdog was triggered.

Queued Access Type State

This parameter shall contain the state (buffer read, buffer written) of the consistent application memory if it is of queued access Type.

Buffered Access Type State

This parameter shall contain the state (buffer no., locked) of the consistent application memory if it is of Buffered Access Type.

Channel Enable

This parameter shall contain the information if the Sync Manager channel is active.

7.3.1.7 Distributed Clock

Distributed clocks enable all slave devices to have the same time. The first slave device within the segment that contains a clock is the clock master. Its clock is used to synchronize the slave clocks of the other slave devices and of the master device. The master device sends a synchronisation telegram at certain intervals (as required in order to avoid the slave clocks diverging beyond application specific limits), in which the slave device containing the master clock enters its current time. The slave devices with slave clocks then read the time from the same telegram. Due to the logical ring structure this is possible since the master clock is located before the slave clocks in the segment.

Since each slave introduces a small delay in the outgoing and return direction (within the device and also in the physical link), the propagation delay time between master clock and the respective slave clock has to be considered during the synchronisation of the slave clocks. For measuring the propagation delay, the master device sends a broadcast read to a special address, which causes each slave device to save the time when the telegram was received (or its local clock time) in the outgoing direction and on the way back. The master can read these saved times and balance them accordingly.

External Synchronisation is accomplished by mechanisms specified in IEEE1588:2002. Any device with external communication interfaces may contain a boundary clock. The EtherCAT slave with the master clock is synchronized to the Boundary Clock. An EtherCAT segment shall have only one active boundary clock at any time in order to meet the IEEE 1588 topology requirements.

Parameter

Receive Time Channel A

This parameter shall contain the receiving time of a special telegram's beginning on channel A. The special telegram shall be a write access to this parameter. The receiving time of this telegram will be latched in this parameter at the end of this telegram if the receiving was correctly. Additionally the latch for the Receive Time Channel B Register will enabled for the same telegram.

Receive Time Channel B

This parameter shall contain the receiving time of a special telegram's beginning on channel B. The special telegram shall be a write access to the Receive Time Channel A Register. The receiving time of this telegram will be latched in this parameter at the end of this telegram if the receiving was correctly.

Local System Time

This parameter shall contain the local system time latched when a telegram is received. A write access to this parameter shall start a compare of the latched local system time with the written reference system time. The result of this compare shall be an input of the PLL for the local system time.

System Time Offset

This parameter shall contain the offset between the local time and the local system time.

System Time Transmission Delay

This parameter shall contain the transmission delay from the EtherCAT Slave Controller with the reference system time to the local EtherCAT Slave Controller.

Interrupt Signal Configuration

This parameter shall contain the behaviour of the Distributed Clock interrupt signals and their acknowledgement.

Interrupt 1 Status

This parameter shall contain the status of the first Distributed Clock interrupt.

Interrupt 2 Status

This parameter shall contain the status of the second Distributed Clock interrupt.

Interrupt Enable

This parameter enables the Distributed Clock interrupts.

Cyclic Operation Enable

This parameter enables the cyclic operation of Distributed Clock.

Cyclic Operation Start Time

This parameter shall contain the start time of the Distributed Clock's interrupt generation.

Interrupt 1 Cyclic Time

This parameter shall contain the cycle time of the Distributed Clock's first interrupt generation.

Interrupt 2 Delay Time

This parameter shall contain the delay time between the first and second interrupt of the Distributed Clock if two interrupts will be used.

7.3.2 Application Memory

The behaviour of an application memory area depends on the access type configured in the corresponding Sync Manager channel.

7.3.2.1 Queued Access Type

Application memory areas of Queued Access Type shall always be read from the communication partner when it was written. Write accesses to these application memory areas before they have been read will result in an error.

7.3.2.1.1 Write Access from Master

The following diagram shows the primitives between master, DL and AL in case of a successful write sequence.

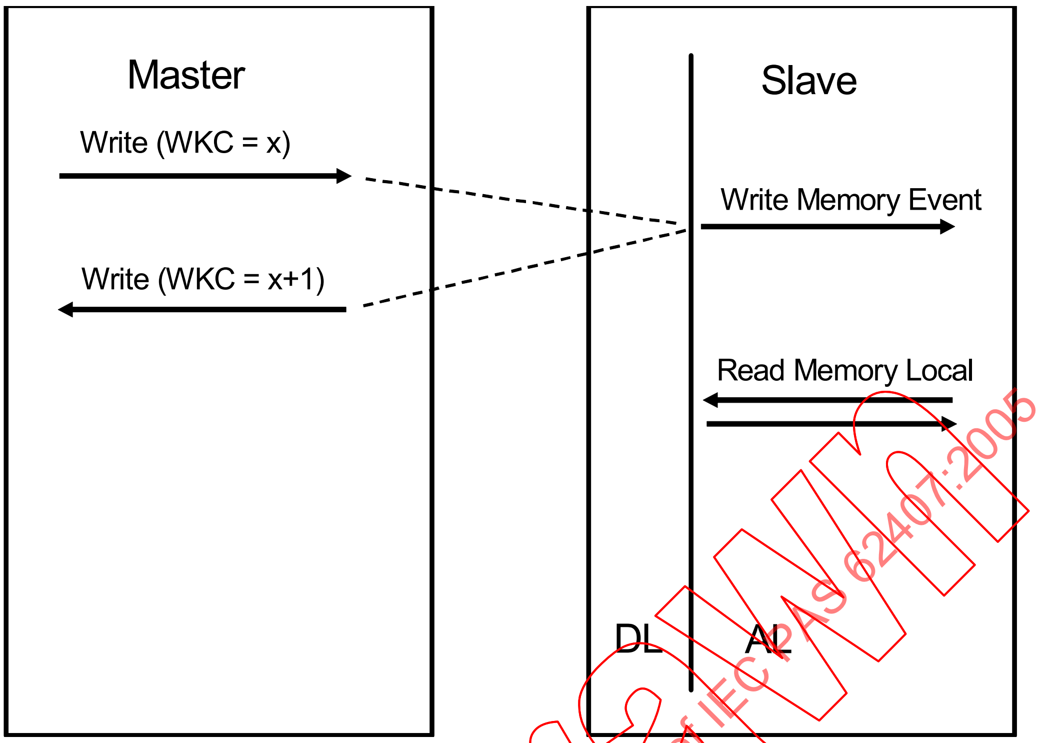


Figure 13: Successful Write sequence to application memory area of Queued Access Type

The master sends an EtherCAT write service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave write the received data in the application memory area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller. The corresponding Sync Manager channel locks the application memory area until it will be read from the AL controller. The master receives a successful write response because the WKC was incremented. The AL controller reads the application memory area and the corresponding sync Manager channel unlocks the application memory area that it can be written by the master again.

The following diagram shows the primitives between master, DL and AL in case of a bad write sequence.

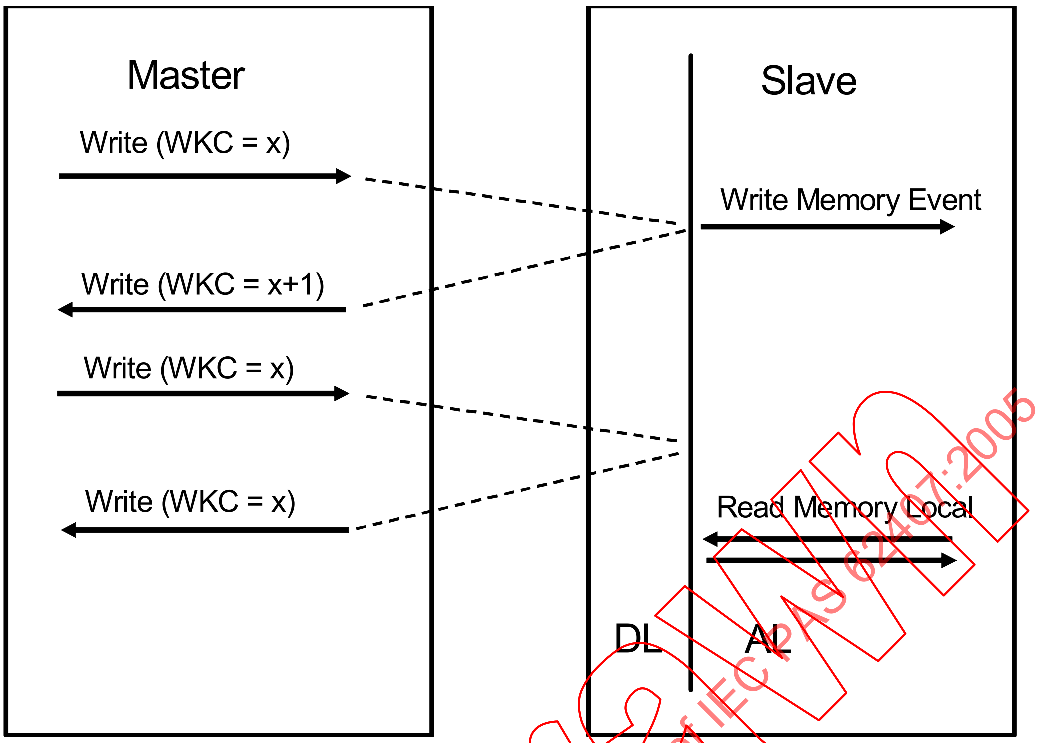


Figure 14: Bad Write sequence to application memory area of Queued Access Type

The master sends an EtherCAT write service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave write the received data in the application memory area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller. The corresponding Sync Manager channel locks the application memory area until it will be read from the AL controller. The master receives a successful write response because the WKC was incremented. Before the AL controller reads the application memory area the master writes the same area again with the Working Counter (WKC = x). Because the application memory area is still locked, the DL of the slave will ignore the received data and will not increment the Working Counter. The master receives a bad write response because the WKC was not incremented. Later the AL controller reads the application memory area and the corresponding sync Manager channel unlocks the application memory area that it can be written by the master again.

7.3.2.1.2 Read Access from Master

The following diagram shows the primitives between master, DL and AL in case of a successful Read sequence.

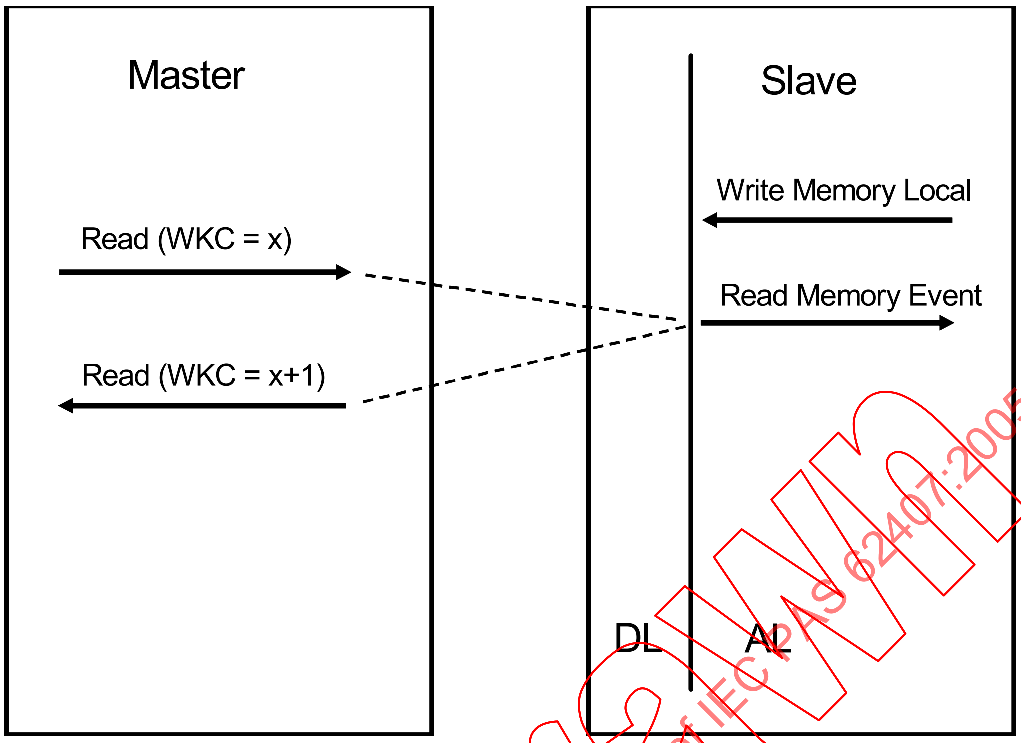


Figure 15: Successful Read sequence to application memory area of Queued Access Type

The AL controller updates the application memory area. The corresponding Sync Manager channel locks the application memory area until it will be read from the master. The master sends an EtherCAT read service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave sends the data of the application memory area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller. The master receives a successful read response because the WKC was incremented. The corresponding Sync Manager channel unlocks the application memory area that it can be written by the AL controller again.

The following diagram shows the primitives between master, DL and AL in case of a bad read sequence.

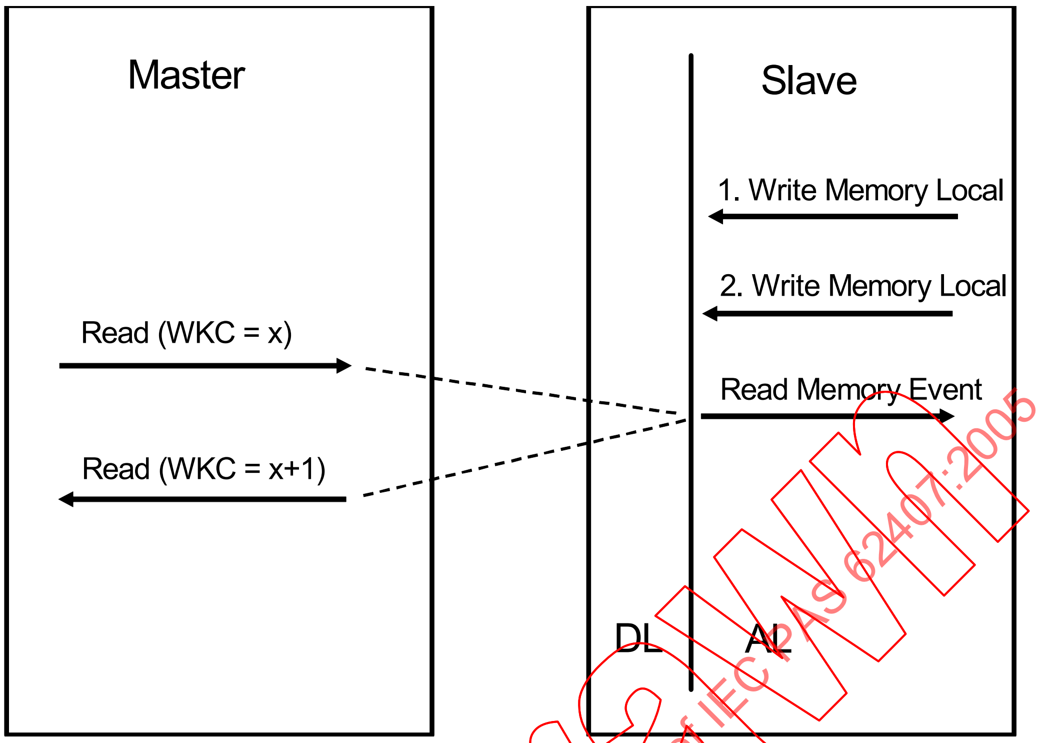


Figure 16: Bad read sequence to application memory area of Queued Access Type

The AL controller updates the application memory area (1. WriteMemoryLocal). The corresponding Sync Manager channel locks the application memory area until it will be read from the master. The AL controller updates the application memory area again (2. WriteMemoryLocal), but this update will be ignored by the corresponding Sync Manager channel because the old data was not read by the master. Then the master sends an EtherCAT read service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave sends the data of the application memory area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller. The master receives a successful read response because the WKC was incremented. The corresponding Sync Manager channel now unlocks the application memory area that it can be written by the AL controller again.

7.3.2.2 Buffered Access Type

Application memory areas of Buffered Access Type can always be read and written from the communication partner.

7.3.2.2.1 Write Access from Master

The following diagram shows the primitives between master, DL and AL in case of a write sequence.

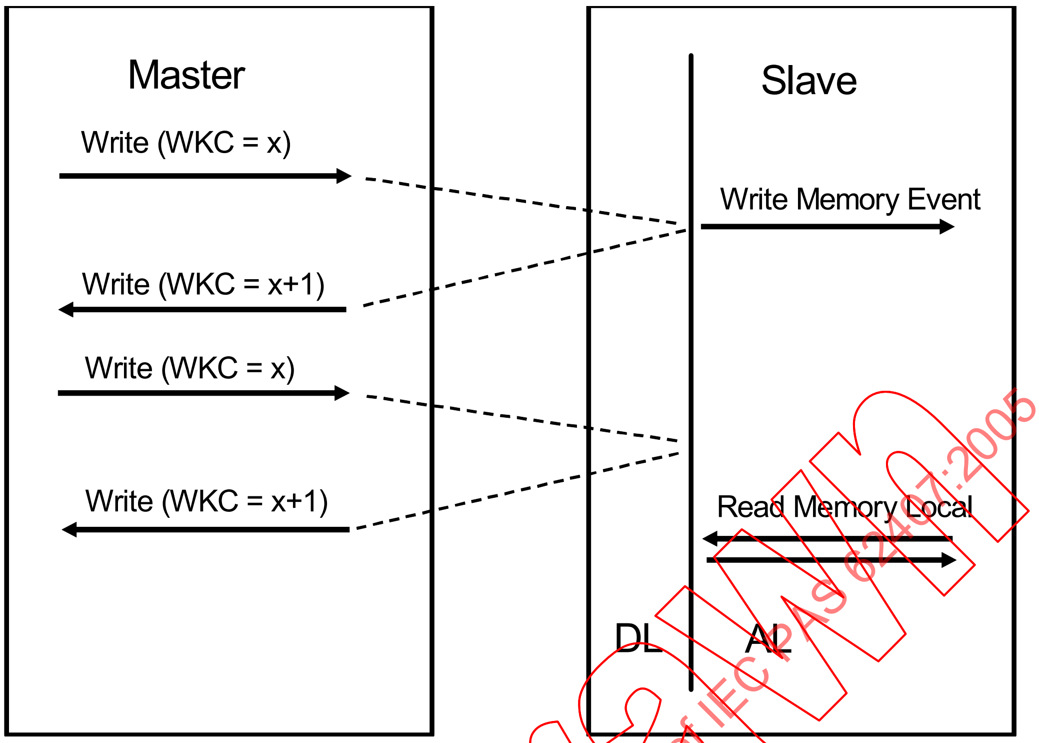


Figure 17: Successful Write sequence to application memory area of Buffered Access Type

The master sends an EtherCAT write service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave write the received data in the application memory area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller. The master receives a successful write response because the WKC was incremented. Before the AL controller reads the application memory area the master writes the same area again with the Working Counter (WKC = x). Because the Buffered Access Type application memory area is never locked, the DL of the slave overwrites the received data in the application memory area, increments the Working Counter (WKC = x + 1) and generates again an AL event to the AL controller. The master receives a successful write response because the WKC was incremented. Later the AL controller reads the application memory area.

7.3.2.2.2 Read Access from Master

The following diagram shows the primitives between master, DL and AL in case of a successful Read sequence.

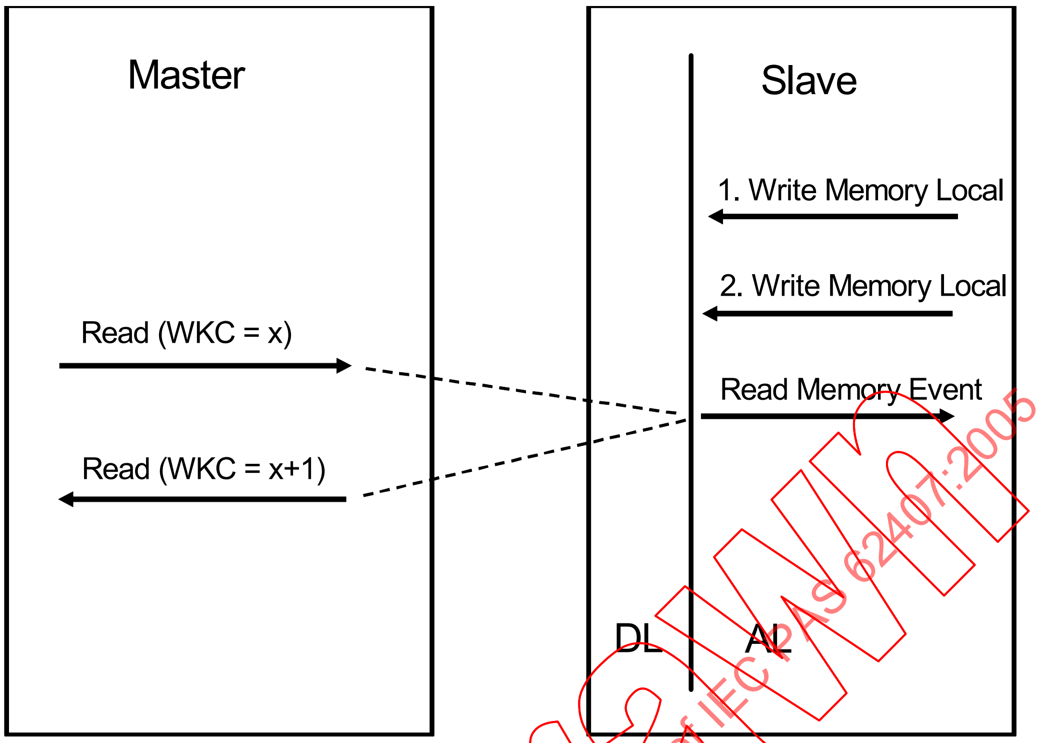


Figure 18: Successful read sequence to application memory area of Buffered Access Type

The AL controller updates the application memory area (1. WriteMemoryLocal). The AL controller updates the application memory area again with new values (2. WriteMemoryLocal), because buffered Type application memory areas will never be locked, the corresponding Sync Manager channel overwrites the old data. Then the master sends an EtherCAT read service with the Working Counter (WKC = x), the DL (EtherCAT Slave Controller) of the slave sends the data of the application memory area, increments the Working Counter (WKC = x + 1) and generates an AL event to the AL controller. The master receives a successful read response because the WKC was incremented.

8 Contents of Part 4: Data Link Layer Protocol definition

8.1 Frame Structure

EtherCAT uses a standard IEEE 802.3 Ethernet frame structure for transporting EtherCAT telegrams. The telegrams may alternatively be sent via UDP/IP. The EtherCAT specific protocol parts are identical in both cases and consist of an EtherCAT frame header and one or more EtherCAT telegrams.

8.1.1 EtherCAT telegram inside an Ethernet frame

The frame structure in consists of the following data entries as specified in Table 17.

Table 17 – EtherCAT telegram inside an Ethernet frame

Frame part	Data Field	Data Type	Value/Description
Ethernet	Dest MAC	BYTE[6]	Destination MAC Address
	Src MAC	BYTE[6]	Source MAC Address
	Ether Type	WORD	0x88A4 (EtherCAT)
EtherCAT Frame	Length	unsigned : 11	Following byte length (excl. CRC)
	Reserved	unsigned : 1	0
	Type	unsigned : 4	Protocol Type = EtherCAT command (0x01)
EtherCAT Telegram 1			Will be described below
…			Will be described below
EtherCAT Telegram n			Will be described below
Ethernet CRC	CRC	DWORD	Standard Ethernet Checksum

8.1.2 EtherCAT frame inside an UDP datagram

The frame structure in UDP mode consists of the following data entries:

Table 18 – EtherCAT frame inside an UDP datagram

Frame part	Data Field	Data Type	Value/Description
Ethernet	Dest MAC	BYTE[6]	Destination MAC Address
	Src MAC	BYTE[6]	Source MAC Address
	Ether Type	WORD	0x0800 (IP)
IP		BYTE[20]	Standard IP Header
UDP	Src Port	WORD	Source Port
	Dest Port	WORD	0x88A4 (EtherCAT)
	Length	WORD	Following byte length (excl. CRCs)
	CRC	WORD	CRC of the UDP-Header
EtherCAT Frame	Length	unsigned : 11	Following byte length (excl. CRC)
	Reserved	unsigned : 1	0
	Type	unsigned : 4	Protocol Type = EtherCAT command (0x01)
EtherCAT Telegram 1			Will be described below
…			Will be described below
EtherCAT Telegram n			Will be described below
Ethernet CRC	CRC	DWORD	Standard Ethernet Checksum

8.1.3 EtherCAT telegram structure

In this chapter the EtherCAT telegram formats are described. One or more of these telegrams are transported by an Ethernet frame or UDP datagram as described above.

8.1.3.1 Read

8.1.3.1.1 Auto Increment Physical Read (APRD)

The Auto Increment Physical Read (APRD) protocol is specified in Table 19.

Table 19 – Auto Increment Physical Read (APRD)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x01 (EtherCAT command APRD)
	IDX	BYTE	Index
	ADP	WORD	Auto Increment Address
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.1.2 Node-Addressed Physical Read (NPRD)

The Node-Addressed Physical Read (NPRD) protocol is specified in Table 20.

Table 20 – Node-Addressed Physical Read (NPRD)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x04 (EtherCAT command NPRD)
	IDX	BYTE	Index
	ADP	WORD	Configured Station Address
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.1.3 Logical Read (LRD)

The Logical Read (LRD) protocol is specified in Table 21.

Table 21 – Logical Read (LRD)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x0A (EtherCAT command LRD)
	IDX	BYTE	Index
	ADR	DWORD	Logical Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.1.4 Broadcast Read (BRD)

The Broadcast Read (BRD) protocol is specified in Table 22.

Table 22 – Broadcast Read (BRD)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x07 (EtherCAT command BRD)
	IDX	BYTE	Index
	Reserved	WORD	0x0000
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.2 Write

8.1.3.2.1 Auto Increment

The Auto Increment Physical Write (APWR) protocol is specified in Table 23.

Table 23 – Auto Increment Physical Write (APWR)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x02 (EtherCAT command APWR)
	IDX	BYTE	Index
	ADP	WORD	Auto Increment Address
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.2.2 Node-Addressed Physical Write (NPWR)

The Node-Addressed Physical Write (NPWR) protocol is specified in Table 24.

Table 24 – Node-Addressed Physical Write (NPWR)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x05 (EtherCAT command NPWR)
	IDX	BYTE	Index
	ADP	WORD	Configured Station Address
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.2.3 Logical Write (LWR)

The Logical Write (LWR) protocol is specified in Table 25.

Table 25 – Logical Write (LWR)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x0B (EtherCAT command LWR)
	IDX	BYTE	Index
	ADR	DWORD	Logical Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.2.4 Broadcast Write (BWR)

The Broadcast Write (BWR) protocol is specified in Table 26.

Table 26 – Broadcast Write (BWR)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x08 (EtherCAT command BWR)
	IDX	BYTE	Index
	Reserved	WORD	0x0000
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.3 ReadWrite

8.1.3.3.1 Logical ReadWrite (LRW)

The Logical ReadWrite (LRW) protocol is specified in Table 27.

Table 27 – Logical ReadWrite (LRW)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x0C (EtherCAT command LRW)
	IDX	BYTE	Index
	ADR	DWORD	Logical Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.1.3.3.2 Auto Increment Physical Read Multiple Write (ARMW)

The Auto Increment Physical Read Multiple Write (ARMW) protocol is specified in Table 28.

Table 28 – Auto Increment Physical Read Multiple Write (ARMW)

Frame part	Data Field	Data Type	Value/Description
EtherCAT Telegram	CMD	BYTE	0x0D (EtherCAT command ARMW)
	IDX	BYTE	Index
	ADP	WORD	Auto Increment Address
	ADO	WORD	Physical Memory Address
	LEN	unsigned : 11	Length
	Reserved	unsigned : 4	0x00
	NEXT	unsigned : 1	0x00: last EtherCAT telegram in Ethernet frame or UDP datagram 0x01: EtherCAT telegram in Ethernet frame or UDP datagram follows
	IRQ	WORD	Reserved for future use
	DATA	BYTE[n]	Data
	WKC	WORD	Working Counter

8.2 EtherCAT Attributes

8.2.1 Register

8.2.1.1 Management

8.2.1.1.1 DL Information

8.2.1.1.1.1 Coding

```
typedef struct
{
  BYTE          Type;
  BYTE          Revision;
  WORD          Build;
  BYTE          NoOfSuppFmmuChannels;
  BYTE          NoOfSuppSyncManChannels;
  BYTE          RamSize;
  BYTE          Reserved1;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                7;
  unsigned      Reserved3:                8;
} TDLINFORMATION;
```

8.2.1.1.1.2 Description

The DL Information protocol is specified in Table 29.

Table 29 – DL Information

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Type	0x0000	BYTE	R	r	
Revision	0x0001	BYTE	R	r	
Build	0x0002	WORD	R	r	
Number of supported FMMU channels	0x0004	BYTE	R	r	0x0001-0x0010
Number of supported Sync Manager channels	0x0005	BYTE	R	r	0x0001-0x0010
RAM Size	0x0006	BYTE	R	r	RAM Size in kByte (5-64)
Reserved	0x0007	BYTE	R	r	
FMMU Bit Operation Not Supported	0x0008	unsigned: 1	R	r	0: Bit Operation supported 1: Bit Operation not supported
Reserved	0x0008	unsigned: 15	R	r	

8.2.1.1.2 Configured Station Address

8.2.1.1.2.1 Coding

```
typedef struct
{
  WORD          FixedStationAddress;
} TFIXEDSTATIONADDRESS;
```

8.2.1.1.2.2 Description

The Configured Station Address protocol is specified in Table 30.

Table 30 – Configured Station Address

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Configured Station Address	0x0010	WORD	Rwl	R	

8.2.1.1.3 DL Control

8.2.1.1.3.1 Coding

```
typedef struct
{
  unsigned      Reserved1:                8;
  unsigned      ChannelALoopControl:      2;
  unsigned      ChannelBLoopControl:      2;
  unsigned      Reserved2:                4;
} TDLCONTROL;
```

8.2.1.1.3.2 Description

The DL Control protocol is specified in Table 31.

Table 31 – DL Control

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Reserved	0x0100	unsigned:8	R	r	0x00
Channel A Loop Control	0x0101	unsigned:2	Rwl	r	0x00: manual loop not active 0x01: manual loop active 0x02,0x03: automatic loop
Channel B Loop Control	0x0101	unsigned:2	Rwl	r	0x00: manual loop not active 0x01: manual loop active 0x02,0x03: automatic loop
Reserved	0x0101	unsigned:4	R	r	0x00

8.2.1.1.4 DL Status

8.2.1.1.4.1 Coding

```
typedef struct
{
  unsigned      PdiOperational:           1;
  unsigned      Reserved1:                1;
  unsigned      PdiWatchdogStatus:        1;
  unsigned      Reserved2:                5;
  unsigned      ChannelALoopStatus:       1;
  unsigned      ChannelASignalDetection:  1;
  unsigned      ChannelBLoopStatus:       1;
  unsigned      ChannelBSignalDetection:  1;
  unsigned      Reserved3:                4;
} TDLSTATUS;
```

8.2.1.1.4.2 Description

The DL Status protocol is specified in Table 32.

Table 32 – DL Status

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
PDI Operational	0x0110	unsigned:1	R	r	0x00: PDI not operational 0x01: PDI operational
Reserved	0x0110	unsigned:1	R	r	0x00
PDI watchdog status	0x0110	unsigned:1	R	r	0x00: PDI watchdog expired 0x01: PDI watchdog not expired
Reserved	0x0110	unsigned:5	R	r	0x00
Channel A Loop Status	0x0111	unsigned:1	R	r	0x00: loop not active 0x01: loop active
Channel A Signal Detection	0x0111	unsigned:1	R	r	0x00: signal not detected on RX-port 0x01: signal detected on RX-port
Channel B Loop Status	0x0111	unsigned:1	R	r	0x00: loop not active 0x01: loop active
Channel B Signal Detection	0x0111	unsigned:1	R	r	0x00: signal not detected on RX-port 0x01: signal detected on RX-port
Reserved	0x0111	unsigned:4	R	r	0x00

8.2.1.1.5 AL Management Interface

8.2.1.1.5.1 AL Control

8.2.1.1.5.1.1 Coding

```
typedef struct
{
  usnigned       State:                    4;
  unsigned       Acknowledge:              1;
  unsigned       Reserved:                 3;
  unsigned       ApplicationSpecific:      8;
} TALCONTROL;
```

8.2.1.1.5.1.2 Description

The AL Control protocol is specified in Table 33.

Table 33 – AL Control

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
State	0x0120	unsigned:4	Rwl	r	0x01: Init Request 0x02: Pre-Operational Request 0x03: Bootstrap Mode Request 0x04: Safe Operational Request 0x08: Operational Request
Acknowledge	0x0120	unsigned:1	Rw	r	0x00: no acknowledge 0x01 acknowledge (must be a positive edge)
Reserved	0x0121	unsigned:3	Rwl	r	0x00
Application Specific	0x0121	unsigned:8	Rwl	r	

8.2.1.1.5.2 AL Status

8.2.1.1.5.2.1 Coding

```
typedef struct
{
   usnigned        State:                 4;
   unsigned        Change:                1;
   unsigned        Reserved:              3;
   unsigned        ApplicationSpecific:   8;
} TALSTATUS;
```

8.2.1.1.5.2.2 Description

The AL Status protocol is specified in Table 34.

Table 34 – AL Status

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
State	0x0130	unsigned:4	R	rw	0x01: Init 0x02: Pre-Operational 0x03: Bootstrap Mode 0x04: Safe Operational 0x08: Operational
Change	0x0130	unsigned:1	R	rw	0x00: confirmation 0x01: change/error
Reserved	0x0131	unsigned:3	R	rw	0x00
Application Specific	0x0131	unsigned:8	R	rw	

8.2.1.1.6 PDI Control

8.2.1.1.6.1 Coding

```
typedef struct
{
  usnigned      Type:                        8;
  unsigned      AlManagementEmulation:       1;
  unsigned      Reserved:                    7;
} TPDICONTROL;
```

8.2.1.1.6.2 Description

The PDI Control protocol is specified in Table 35.

Table 35 – PDI Control

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Type	0x0140	unsigned:8	R	r	0x00: deactivated 0x01: 4 DI 0x02: 4 DO 0x03: 2 DI / 2 DO 0x05: SPI 0x08: 16 Bit µC 0x09: 8 Bit µC 0x10: 32 DI 0x11: 24 DI / 8 DO 0x12: 16 DI / 16 DO 0x13: 8 DI / 24 DO 0x14: 32 DO
AlManagementEmulation	0x0140	unsigned:1	R	r	0x00: AL Management will be done by an application Controller 0x01: AL Management will be emulated
Reserved	0x0131	unsigned:7	R	rw	0x00

8.2.1.1.7 PDI Configuration

8.2.1.1.7.1 8 Bit Micro Controller Interface

8.2.1.1.7.1.1 Coding

```
typedef struct
{
  usnigned      BusyOutputDriver:            1;
  unsigned      BusyPolarity:                1;
  usnigned      IntOutputDriver:             1;
  unsigned      IntPolarity:                 1;
  unsigned      Reserved:                    12;
} TPDICONFIGURATIONMCI8;
```

8.2.1.1.7.1.2 Description

The 8 Bit Micro Controller Interface protocol is specified in Table 36.

Table 36 – 8 Bit Micro Controller Interface

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
BusyOutputDriver	0x0150	unsigned:1	R	r	BUSY-Signal Output Driver 0: PushPull 1: OpenDrain
BusyPolarity	0x0150	unsigned:1	R	r	BUSY-Signal Polarity 0: low active 1: high active
IntOutputDriver	0x0150	unsigned:1	R	r	INT-Signal Output Driver 0: PushPull 1: OpenDrain
IntPolarity	0x0150	unsigned:1	R	r	INT-Signal Polarity 0: low active 1: high active
Reserved	0x0150	unsigned:12	R	r	0x00

8.2.1.1.7.2 16 Bit Micro Controller Interface

8.2.1.1.7.2.1 Coding

```
typedef struct
{
  usnigned      BusyOutputDriver:          1;
  unsigned      BusyPolarity:              1;
  usnigned      IntOutputDriver:           1;
  unsigned      IntPolarity:               1;
  unsigned      Reserved:                  12;
} TPDICONFIGURATIONMCI16;
```

8.2.1.1.7.2.2 Description

The 16 Bit Micro Controller Interface protocol is specified in Table 37.

Table 37 – 16 Bit Micro Controller Interface

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
BusyOutputDriver	0x0150	unsigned:1	R	r	BUSY-Signal Output Driver 0: PushPull 1: OpenDrain
BusyPolarity	0x0150	unsigned:1	R	r	BUSY-Signal Polarity 0: low active 1: high active
IntOutputDriver	0x0150	unsigned:1	R	r	INT-Signal Output Driver 0: PushPull 1: OpenDrain
IntPolarity	0x0150	unsigned:1	R	r	INT-Signal Polarity 0: low active 1: high active
Reserved	0x0150	unsigned:12	R	r	0x00

8.2.1.1.8 AL Event

8.2.1.1.8.1 Coding

```
typedef struct
{
  unsigned      AlControlEvent:              1;
  unsigned      SyncManWatchdogEvent:        1;
  unsigned      Reserved1:                   6;
  unsigned      SyncManChannel0Event:        1;
  unsigned      SyncManChannel1Event:        1;
  unsigned      SyncManChannel2Event:        1;
  unsigned      SyncManChannel3Event:        1;
  unsigned      SyncManChannel4Event:        1;
  unsigned      SyncManChannel5Event:        1;
  unsigned      SyncManChannel6Event:        1;
  unsigned      SyncManChannel7Event:        1;
  unsigned      SyncManChannel8Event:        1;
  unsigned      SyncManChannel9Event:        1;
  unsigned      SyncManChannel10Event:       1;
  unsigned      SyncManChannel11Event:       1;
  unsigned      SyncManChannel12Event:       1;
  unsigned      SyncManChannel13Event:       1;
  unsigned      SyncManChannel14Event:       1;
  unsigned      SyncManChannel15Event:       1;
  unsigned      Reserved2:                   1;
} TALEVENT;
```

8.2.1.1.8.2 Description

The AL Event protocol is specified in Table 38.

Table 38 – AL Event

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
AL Control Event	0x0220	unsigned:1	R	r	0x00: no event active 0x01: event active (AL Control register was written)
Sync Manager Watchdog Event	0x0220	unsigned:1	R	r	0x00: no event active 0x01 event active (at least one Sync Manager watchdog was expired)
Reserved	0x0220	unsigned:6	R	r	0x00
Sync Manager Channel 0 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 1 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 2 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 3 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Sync Manager Channel 4 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 5 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 6 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 7 Event	0x0221	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 8 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 9 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 10 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 11 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 12 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 13 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 14 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Sync Manager Channel 15 Event	0x0222	unsigned:1	R	r	0x00: no event active 0x01: event active (Sync Manager channel was accessed by EtherCAT service)
Reserved	0x0223	unsigned:8	R	r	0x00

8.2.1.2 Statistics

8.2.1.2.1 Channel CRC Fault Counter

8.2.1.2.1.1 Coding

```
typedef struct
{
  WORD              ChannelACrcFaultCounter;
  WORD              ChannelBCrcFaultCounter;
} TCHANNELCRCFAULTCOUNTER;
```

8.2.1.2.1.2 Description

The Channel CRC Fault Counter protocol is specified in Table 39.

Table 39 – Channel CRC Fault Counter

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Channel A CRC fault counter	0x0300	WORD	RwI	R	
Channel B CRC fault counter	0x0302	WORD	RwI	R	

8.2.1.3 Watchdogs

8.2.1.3.1 Watchdog Divider

8.2.1.3.1.1 Coding

```
typedef struct
{
  WORD              WatchdogDivider;
} TWATCHDOGDIVIDER;
```

8.2.1.3.1.2 Description

The Watchdog Divider protocol is specified in Table 40.

Table 40 – Watchdog Divider

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Watchdog divider	0x0400	WORD	RwI	r	

8.2.1.3.2 PDI Watchdog

8.2.1.3.2.1 Coding

```
typedef struct
{
  WORD              PdiWatchdog;
} TPDIWATCHDOG;
```

8.2.1.3.2.2 Description

The PDI Watchdog protocol is specified in Table 41.

Table 41 – PDI Watchdog

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
PDI Watchdog	0x0410	WORD	Rwl	r	

8.2.1.3.3 Sync Manager Channel Watchdog

8.2.1.3.3.1 Coding

```
typedef struct
{
  WORD          SyncManChannelWatchdog[16];
} TSYNCMANCHANNELWATCHDOG;
```

8.2.1.3.3.2 Description

The Sync Manager Channel Watchdog protocol is specified in Table 42.

Table 42 – Sync Manager Channel Watchdog

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Sync Manager Channel 0 Watchdog	0x0420	WORD	Rwl	r	
Sync Manager Channel 1 Watchdog	0x0422	WORD	Rwl	r	
Sync Manager Channel 2 Watchdog	0x0424	WORD	Rwl	r	
Sync Manager Channel 3 Watchdog	0x0426	WORD	Rwl	r	
Sync Manager Channel 4 Watchdog	0x0428	WORD	Rwl	r	
Sync Manager Channel 5 Watchdog	0x042A	WORD	Rwl	r	
Sync Manager Channel 6 Watchdog	0x042C	WORD	Rwl	r	
Sync Manager Channel 7 Watchdog	0x042E	WORD	Rwl	r	
Sync Manager Channel 8 Watchdog	0x0430	WORD	Rwl	r	
Sync Manager Channel 9 Watchdog	0x0432	WORD	Rwl	r	
Sync Manager Channel 10 Watchdog	0x0434	WORD	Rwl	r	
Sync Manager Channel 11 Watchdog	0x0436	WORD	Rwl	r	
Sync Manager Channel 12 Watchdog	0x0438	WORD	Rwl	r	
Sync Manager Channel 13 Watchdog	0x043A	WORD	Rwl	r	
Sync Manager Channel 14 Watchdog	0x043C	WORD	Rwl	r	
Sync Manager Channel 15 Watchdog	0x043E	WORD	Rwl	r	

8.2.1.3.4 Sync Manager Watchdog Status

8.2.1.3.4.1 Coding

```
typedef struct
{
  unsigned      SyncManChannel0WdStatus:     1;
  unsigned      SyncManChannel1WdStatus:     1;
  unsigned      SyncManChannel2WdStatus:     1;
  unsigned      SyncManChannel3WdStatus:     1;
  unsigned      SyncManChannel4WdStatus:     1;
  unsigned      SyncManChannel5WdStatus:     1;
  unsigned      SyncManChannel6WdStatus:     1;
  unsigned      SyncManChannel7WdStatus:     1;
  unsigned      SyncManChannel8WdStatus:     1;
  unsigned      SyncManChannel9WdStatus:     1;
  unsigned      SyncManChannel10WdStatus:    1;
  unsigned      SyncManChannel11WdStatus:    1;
  unsigned      SyncManChannel12WdStatus:    1;
  unsigned      SyncManChannel13WdStatus:    1;
  unsigned      SyncManChannel14WdStatus:    1;
  unsigned      SyncManChannel15WdStatus:    1;
} TSYNCMANCHANNELWDSTATUS;
```

8.2.1.3.4.2 Description

The Sync Manager Watchdog Status protocol is specified in Table 43.

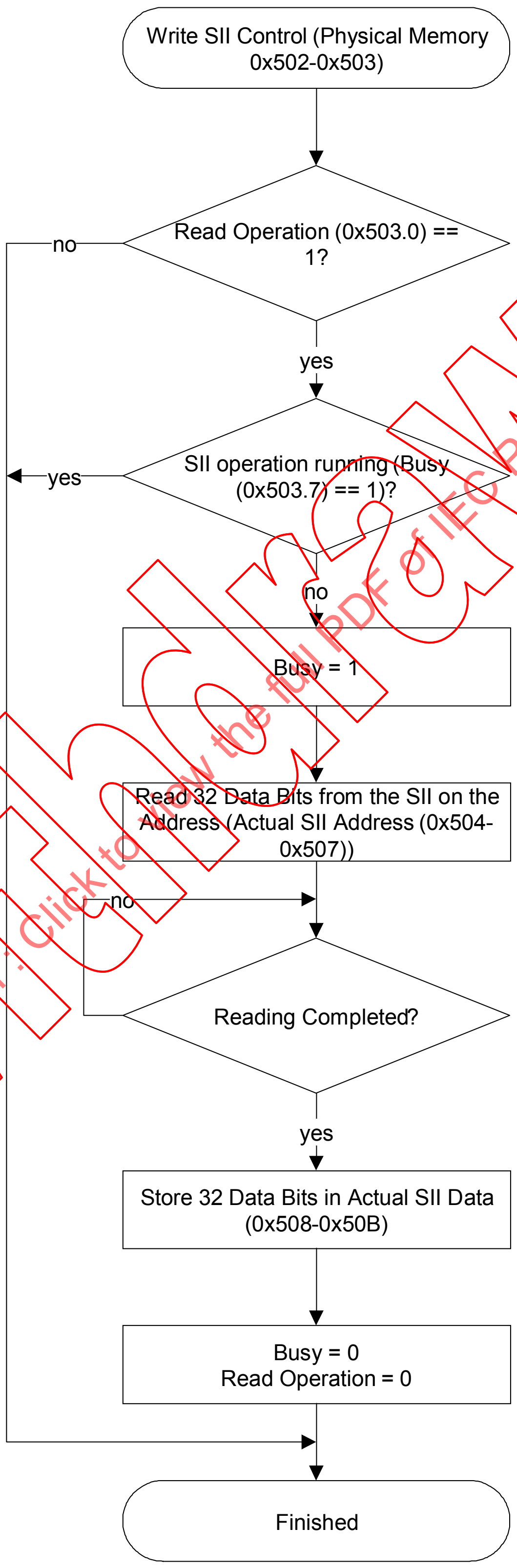
Table 43 – Sync Manager Watchdog Status

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Sync Manager Channel 0 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 1 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 2 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 3 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 4 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 5 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 6 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 7 Watchdog Status	0x0450	Unsigned:1	R	r	
Sync Manager Channel 8 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 9 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 10 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 11 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 12 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 13 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 14 Watchdog Status	0x0451	Unsigned:1	R	r	
Sync Manager Channel 15 Watchdog Status	0x0451	Unsigned:1	R	r	

8.2.1.4 Slave Information Interface

8.2.1.4.1 Slave Information Interface Access Flow Charts

8.2.1.4.1.1 Read Operation



Write SII Control (Physical Memory 0x502-0x503)
Read Operation (0x503.0) == 1?
no
yes
SII operation running (Busy (0x503.7) == 1)?
yes
no
Busy = 1
Read 32 Data Bits from the SII on the Address (Actual SII Address (0x504-0x507))
no
Reading Completed?
yes
Store 32 Data Bits in Actual SII Data (0x508-0x50B)
Busy = 0
Read Operation = 0
Finished

Figure 19: Slave Information Interface Read Operation

8.2.1.4.1.2 Write Operation

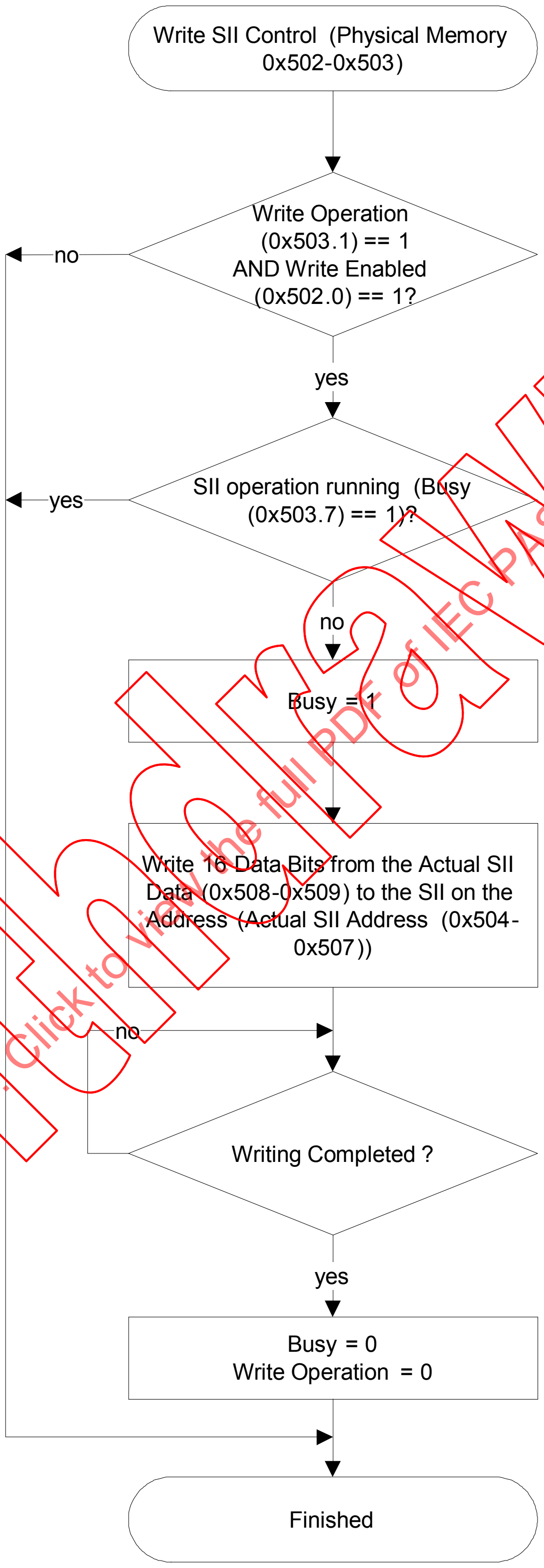

Write SII Control (Physical Memory 0x502-0x503)
Write Operation (0x503.1) == 1 AND Write Enabled (0x502.0) == 1?
no
yes
SII operation running (Busy (0x503.7) == 1)?
yes
no
Busy = 1
Write 16 Data Bits from the Actual SII Data (0x508-0x509) to the SII on the Address (Actual SII Address (0x504-0x507))
no
Writing Completed ?
yes
Busy = 0
Write Operation = 0
Finished

Figure 20: Slave Information Interface Write Operation

8.2.1.4.1.3 Reload Operation

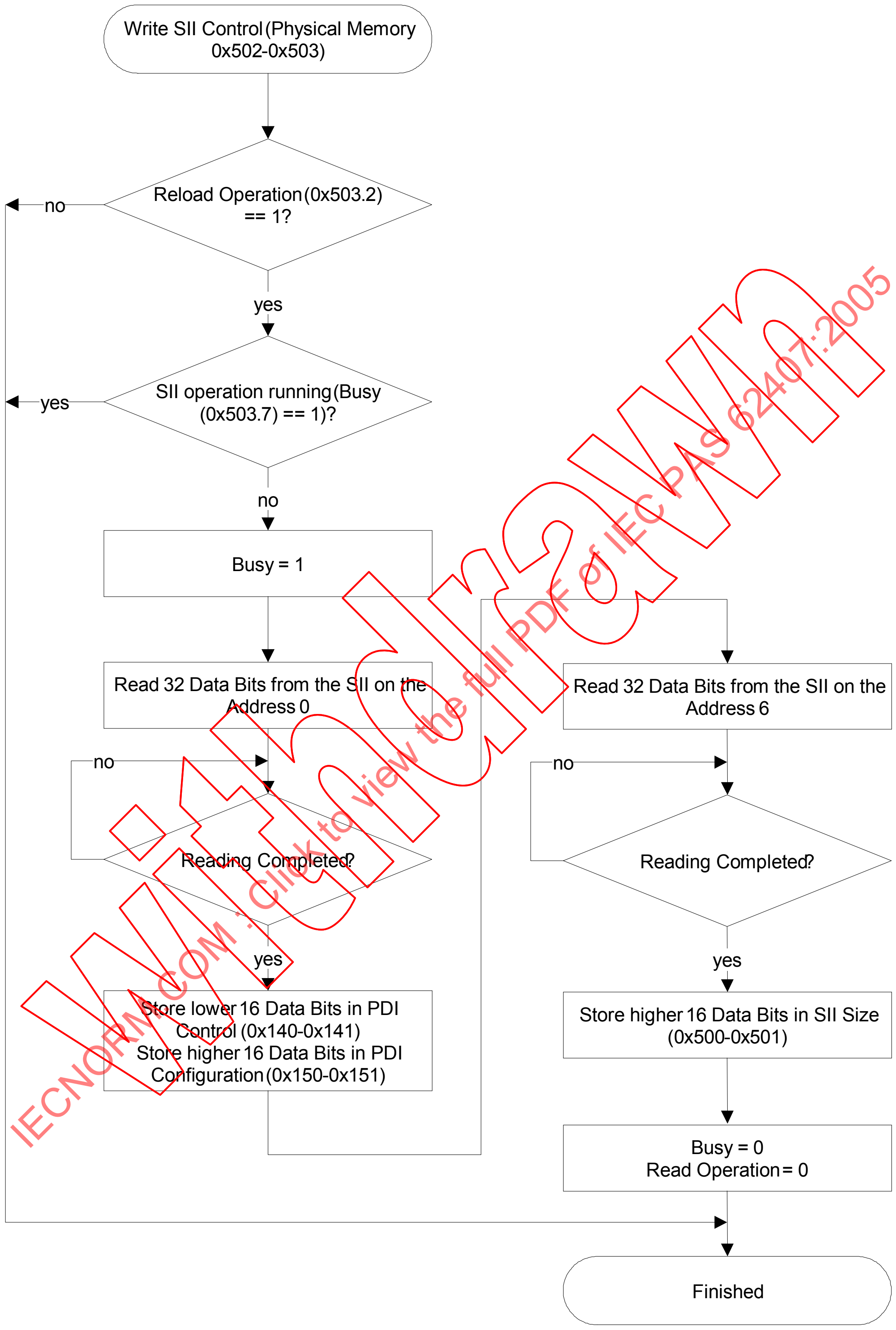

Write SII Control (Physical Memory 0x502-0x503)
Reload Operation (0x503.2) == 1?
no
yes
SII operation running (Busy (0x503.7) == 1)?
yes
no
Busy = 1
Read 32 Data Bits from the SII on the Address 0
no
Reading Completed?
yes
Store lower 16 Data Bits in PDI Control (0x140-0x141)
Store higher 16 Data Bits in PDI Configuration (0x150-0x151)
Read 32 Data Bits from the SII on the Address 6
no
Reading Completed?
yes
Store higher 16 Data Bits in SII Size (0x500-0x501)
Busy = 0
Read Operation = 0
Finished

Figure 21: Slave Information Interface Reload Operation

8.2.1.4.2 Slave Information Interface Area

The Slave Information Interface Area protocol is specified in Table 44.

Table 44 – Slave Information Interface Area

Parameter	Address	Data Type	Value/Description
PDI Control	0x0000	WORD	Initialization value for PDI Control register (0x140-0x141)
PDI Configuration	0x0002	WORD	Initialization value for PDI Configuration register (0x150-0x151)
Reserved	0x0004	BYTE[10]	Shall be zero
E²PROM size	0x000E	WORD	Initialization value for Slave Information Interface Size register (0x500-0x501)
Vendor ID	0x0010	DWORD	Object 0x1018, Subindex 1
Product Code	0x0014	DWORD	Object 0x1018, Subindex 2
Revision Number	0x0018	DWORD	Object 0x1018, Subindex 3
Serial Number	0x001C	DWORD	Object 0x1018, Subindex 4
Execution Delay	0x0020	WORD	To be done
Channel A Delay	0x0022	WORD	To be done
Channel B Delay	0x0024	WORD	To be done
Reserved	0x0026	WORD	Shall be zero
Bootstrap Receive Mailbox Offset	0x0028	WORD	Receive Mailbox Offset for Bootstrap state (Master to Slave)
Bootstrap Receive Mailbox Size	0x002A	WORD	Receive Mailbox Size for Bootstrap state (Master to Slave)
Bootstrap Send Mailbox Offset	0x002C	WORD	Send Mailbox Offset for Bootstrap state (Slave to Master)
Bootstrap Send Mailbox Size	0x002E	WORD	Send Mailbox Size for Bootstrap state (Slave to Master)
Reserved	0x0030	BYTE[80]	Shall be zero
First Category Header	0x0080	Unsigned: 15	Category Type
	0x0080	Unsigned: 1	Vendor Specific
	0x0082	WORD	Following Category Word Size
First Category Data	0x0084	Category dependent	Category Data
Second Category Header	0x0084 + x	Unsigned: 15	Category Type
	0x0084 + x	Unsigned: 1	Vendor Specific
	0x0086 + x	WORD	Following Category Word Size
Second Category Data	0x0088 + x	Category dependent	Category Data
…			

8.2.1.4.2.1 Slave Information Interface Size

8.2.1.4.2.1.1 Coding

```
typedef struct
{
  WORD            SerialEeepromSize;
} TSERIALEEPROMSIZE;
```

8.2.1.4.2.1.2 Description

The Slave Information Interface Size protocol is specified in Table 45.

Table 45 – Slave Information Interface Size

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Size	0x0500	WORD	R	r	E²PROM bit size

8.2.1.4.2.2 Slave Information Interface Control/Status

8.2.1.4.2.2.1 Coding

```
typedef struct
{
  unsigned      EepromWriteAccess:         1;
  unsigned      Reserved1:                 6;
  unsigned      EepromAddressAlgorithm:    1;
  unsigned      ReadOperation:             1;
  unsigned      WriteOperation:            1;
  unsigned      ReloadOperation:           1;
  unsigned      Reserved2:                 3;
  unsigned      WriteError:                1;
  unsigned      Busy:                      1;
} TSERIALEEPROMCONTROL;
```

8.2.1.4.2.2.2 Description

The Slave Information Interface Control/Status protocol is specified in Table 46.

Table 46 – Slave Information Interface Control/Status

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
E²PROM Write Access	0x0502	Unsigned:1	RwI	r	0x00: only read access to E²PROM 0x01: read and write access to E²PROM
Reserved	0x0502	Unsigned:6	R	r	0x00
E²PROM Address Algorithm	0x0502	Unsigned:1	R	r	0x00: I²C protocol to E²PROM has one address byte 0x01: I²C protocol to E²PROM has two address bytes
Read Operation	0x0503	Unsigned:1	Rw	r	0x00: no read operation requested (parameter write) or read operation not busy (parameter read) 0x01: read operation requested (parameter write) or read operation busy (parameter read) To start a new read operation there must be a positive edge on this parameter
Write Operation	0x0503	Unsigned:1	Rw	r	0x00: no write operation requested (parameter write) or write operation not busy (parameter read) 0x01: write operation requested (parameter write) or write operation busy (parameter read) To start a new write operation there must be a positive edge on this parameter
Reload Operation	0x0503	Unsigned:1	Rw	r	0x00: no reload operation requested (parameter write) or reload operation not busy (parameter read) 0x01: reload operation

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
					requested (parameter write) or reload operation busy (parameter read) To start a new reload operation there must be a positive edge on this parameter
Reserved	0x0503	Unsigned:3	R	r	0x00
Write Error	0x0503	Unsigned:1	R	r	0x00: no error on write operation 0x01: error on write operation
Busy	0x0503	Unsigned:1	R	r	0x00: operation is finished 0x01: operation is busy

8.2.1.4.2.3 Actual Slave Information Interface Address

8.2.1.4.2.3.1 Coding

```
typedef struct
{
  WORD          SerialEeepromAddress;
} TSERIALEEPROMADDRESS;
```

8.2.1.4.2.3.2 Description

The Actual Slave Information Interface Address protocol is specified in Table 47.

Table 47 – Actual Slave Information Interface Address

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Address	0x0504	WORD	Rw	r	16-Bit address

8.2.1.4.2.4 Actual Slave Information Interface Data

8.2.1.4.2.4.1 Coding

```
typedef struct
{
  DWORD          SerialEepromData;
} TSERIALEEPROMDATA;
```

8.2.1.4.2.4.2 Description

The Actual Slave Information Interface Data protocol is specified in Table 48.

Table 48 – Actual Slave Information Interface Data

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Data	0x0508	DWORD	Rw	r	For the write operation only the lower 16 Bit (0x508-0x509) will be used

8.2.1.5 Fieldbus Memory Management Unit (FMMU)

8.2.1.5.1 Coding

```
typedef struct
{
  DWORD         LogicalStartAddress;
  WORD          Length;
  unsigned      LogicalStartBit:        3;
  unsigned      Reserved1:              5;
  unsigned      LogicalEndBit:          3;
  unsigned      Reserved2:              5;
  WORD          PhysicalStartAddress;
  unsigned      PhysicalStartBit:       3;
  unsigned      Reserved3:              5;
  unsigned      ReadEnable:             1;
  unsigned      WriteEnable:            1;
  unsigned      Reserved4:              6;
  unsigned      ChannelEnable:          1;
  unsigned      Reserved5:              7;
  unsigned      Reserved6:              8;
  WORD          Reserved7;
} TFMMU;
```

8.2.1.5.2 Description

The FMMU structure as specified in Table 49 and Table 50 is the same for each FMMU channel.

Table 49 – Fieldbus Memory Management Unit (FMMU) Part 1

Parameter	Physical Address (Offset)	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Logical Start Address	0x0000	DWORD	Rwl	R	
Length	0x0004	WORD	Rwl	R	
Logical Start Bit	0x0006	Unsigned:3	Rwl	R	
Reserved	0x0006	Unsigned:5	R	R	0x00
Logical End Bit	0x0007	Unsigned:3	Rwl	R	
Reserved	0x0007	Unsigned:5	R	R	0x00
Physical Start Address	0x0008	WORD	Rwl	R	
Physical Start Bit	0x000A	Unsigned:3	Rwl	R	
Reserved	0x000A	Unsigned:5	R	R	0x00
Read Enable	0x000B	Unsigned:1	Rwl	R	0x00: channel will be ignored for read service 0x01: channel will be used for read service
Write Enable	0x000B	Unsigned:1	Rwl	R	0x00: channel will be ignored for write service 0x01: channel will be used for write service
Reserved	0x000B	Unsigned:6	R	R	0x00
Channel Enable	0x000C	Unsigned:1	Rwl	R	0x00: channel not active 0x01: channel active
Reserved	0x000C	Unsigned:15	R	R	0x0000
Reserved	0x000E	WORD	R	R	0x0000

Table 50 – Fieldbus Memory Management Unit (FMMU) Part 2

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
FMMU channel 0	0x0600	TFMMU	Rwl	R	
FMMU channel 1	0x0610	TFMMU	Rwl	R	
FMMU channel 2	0x0620	TFMMU	Rwl	R	
FMMU channel 3	0x0630	TFMMU	Rwl	R	
FMMU channel 4	0x0640	TFMMU	Rwl	R	
FMMU channel 5	0x0650	TFMMU	Rwl	R	
FMMU channel 6	0x0660	TFMMU	Rwl	R	
FMMU channel 7	0x0670	TFMMU	Rwl	R	
FMMU channel 8	0x0680	TFMMU	Rwl	R	
FMMU channel 9	0x0690	TFMMU	Rwl	R	
FMMU channel 10	0x06A0	TFMMU	Rwl	R	
FMMU channel 11	0x06B0	TFMMU	Rwl	R	
FMMU channel 12	0x06C0	TFMMU	Rwl	R	
FMMU channel 13	0x06D0	TFMMU	Rwl	R	
FMMU channel 14	0x06E0	TFMMU	Rwl	R	
FMMU channel 15	0x06F0	TFMMU	Rwl	R	

8.2.1.6 Sync Manager

8.2.1.6.1 Coding

```
typedef struct
{
  WORD          PhysicalStartAddress;
  WORD          Length;
  unsigned      BufferType:                 2;
  unsigned      Direction:                  2;
  unsigned      Reserved1:                  1;
  unsigned      AlEventEnable:              1;
  unsigned      WatchdogEnable:             1;
  unsigned      Reserved2:                  1;
  unsigned      WriteEvent:                 1;
  unsigned      ReadEvent:                  1;
  unsigned      WatchdogTrigger:            1;
  unsigned      OneBufferState:             1;
  unsigned      ThreeBufferState:           2;
  unsigned      Reserved3:                  2;
  unsigned      ChannelEnable:              1;
  unsigned      Reserved5:                  7;
  unsigned      Reserved6:                  8;
} TSYNCMAN;
```

8.2.1.6.2 Description

The structure specified in Table 51 and Table 52 is the same for each Sync Manager channel.

Table 51 – Sync Manager Structure Part 1

Parameter	Physical Address (Offset)	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Physical Start Address	0x0000	WORD	Rwl	r	
Length	0x0002	WORD	Rwl	r	
Buffer Type	0x0004	Unsigned:2	Rwl	r	0x00: buffered 0x02: queued
Direction	0x0004	Unsigned:2	Rwl	r	0x00: buffer shall be read from the master 0x01: buffer shall be written from the master
Reserved	0x0004	Unsigned:1	R	r	0x00
AL Event Enable	0x0004	Unsigned:1	Rwl	r	0x00: AL event is not active 0x01: AL event is active (when buffer was accessed and is no longer locked)
Watchdog Enable	0x0004	Unsigned:1	Rwl	r	0x00: Watchdog disabled 0x01: Watchdog enabled
Reserved	0x0004	Unsigned:1	R	r	0x00
Write Event	0x0005	Unsigned:1	R	r	0x00: no write event 0x01: write event
Read Event	0x0005	Unsigned:1	R	r	0x00: no read event 0x01: read event
Watchdog Trigger	0x0005	unsaigned:1	R	r	0x00: watchdog was not triggered 0x01: watchdog was triggered
queued State	0x0005	Unsigned:1	R	r	0x00: buffer read 0x01: buffer written
buffered State	0x0005	Unsigned:2	R	r	0x00: first buffer 0x01: second buffer 0x02: third buffer 0x03: buffer locked
Reserved	0x0005	Unsigned:2	R	r	0x00
Channel Enable	0x0006	Unsigned:1	Rwl	r	0x00: channel disabled 0x01: channel enabled
Reserved	0x0006	Unsigned:15	R	r	0x00

Table 52 – Sync Manager Structure Part 2

Parameter	Physical Address	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Sync Manager channel 0	0x0800	TSYNCMAN	Rwl	r	
Sync Manager channel 1	0x0808	TSYNCMAN	Rwl	r	
Sync Manager channel 2	0x0810	TSYNCMAN	Rwl	r	
Sync Manager channel 3	0x0818	TSYNCMAN	Rwl	r	
Sync Manager channel 4	0x0820	TSYNCMAN	Rwl	r	
Sync Manager channel 5	0x0828	TSYNCMAN	Rwl	r	
Sync Manager channel 6	0x0830	TSYNCMAN	Rwl	r	
Sync Manager channel 7	0x0838	TSYNCMAN	Rwl	r	
Sync Manager channel 8	0x0840	TSYNCMAN	Rwl	r	
Sync Manager channel 9	0x0848	TSYNCMAN	Rwl	r	
Sync Manager channel 10	0x0850	TSYNCMAN	Rwl	r	
Sync Manager channel 11	0x0858	TSYNCMAN	Rwl	r	
Sync Manager channel 12	0x0860	TSYNCMAN	Rwl	r	
Sync Manager channel 13	0x0868	TSYNCMAN	Rwl	r	
Sync Manager channel 14	0x0870	TSYNCMAN	Rwl	r	
Sync Manager channel 15	0x0878	TSYNCMAN	Rwl	r	

8.2.1.7 Distributed Clock

8.2.1.7.1 Coding

```
typedef struct
{
  DWORD          ReceiveTimeChannelA;
  DWORD          ReceiveTimeChannelB;
  BYTE           Reserved1[8];
  UINT64         LocalSystemTimeLatchAtReceiveBegin;
  BYTE           Reserved2[8];
  UINT64         SystemTimeOffset;
  DWORD          SystemTimeTransmissionDelay;
  BYTE           Reserved3[4];
} TDCTRANSMISSION;

typedef struct
{
  WORD           InterruptSignalConfiguration;
  BYTE           Reserved1[8];
  unsigned       Interrupt1Status:         1;
  unsigned       Reserved2:                7;
  unsigned       Interrupt2Status:         1;
  unsigned       Reserved3:                7;
  unsigned       CyclicOperationEnable:    1;
  unsigned       InterruptEnable:          1;
  unsigned       Reserved4:                14;
  DWORD          CyclicOperationStartTime;
  BYTE           Reserved5[4];
  DWORD          Interrupt1CycleTime;
  DWORD          Interrupt2DelayTime;
} TDCINTERRUPT;
```

8.2.1.7.2 Description

The Distributed Clock Structure is specified in Table 53 and Table 54.

Table 53 – Distributed Clock Structure Part 1

Parameter	Physical Address (Offset)	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Receive Time Channel A	0x0900	DWORD	Rf	r	A write access latches the local time (in ns) at receive begin on channel A of this telegram in this parameter (if the telegram was received correctly) and enables the latch of channel B
Receive Time Channel B	0x0904	DWORD	Rf	r	A write access latches the local time (in ns) at receive begin on channel B of this telegram in this parameter (if the latch of channel B is enabled and if the telegram was received correctly)
Reserved	0x0908	BYTE[8]	R	r	
System Time	0x0910	UINT64	Rf	r	A write access compares the latched local system time (in ns) at receive begin on channel A of this telegram with the written value (if the telegram was received correctly), the result will be the input of DC PLL
Reserved	0x0918	BYTE[8]	R	r	
System Time Offset	0x0920	UINT64	Rwl	r	Offset between the local time (in ns) and the local system time (in ns)
System Time Transmission Delay	0x0928	DWORD	Rwl	r	Offset between the reference system time (in ns) and the local system time (in ns)
Reserved	0x092C	BYTE[4]	R	r	

Table 54 – Distributed Clock Structure Part 2

Parameter	Physical Address (Offset)	Data Type	Access Type EtherCAT	Access Type PDI	Value/Description
Reserved	0x0970	BYTE[2]	R	r	
Interrupt Signal Configuration	0x0972	WORD	Rwl	r	0x0000: interrupt signal will be reset when the corresponding Interrupt Status is read 0x0001-0xFFFF: time in 40 ns how long the interrupt signal is set
Reserved	0x0974	BYTE[8]	R	r	
Interrupt 1 Status	0x097C	Unsigned:1	R	r	0: not active 1: active
Reserved	0x097C	Unsigned:7	R	r	
Interrupt 2 Status	0x097D	Unsigned:1	R	r	0: not active 1: active
Reserved	0x097D	Unsigned:7	R	r	
Cyclic Operation Enable	0x097E	Unsigned:1	Rwl	r	0: disabled 1: enabled
Interrupt Enable	0x097E	Unsigned:1	Rwl	r	0: disabled 1: enabled
Reserved	0x097E	Unsigned:14	R	r	
Cyclic Operation Start Time	0x0980	DWORD	Rw	r	The interrupt generation will start when the lower 32 bits of the system time will reach this value (in ns)
Reserved	0x0984	BYTE[4]	R	r	
Interrupt 1 Cycle Time	0x0988	DWORD	R	r	Cycle time of the first interrupt (in ns)
Interrupt 2 Delay Time	0x098C	DWORD	R	r	Delay time of the second interrupt after the first interrupt (in ns)

8.2.2 Application Memory

8.2.2.1 Queued Access Type Flow Charts

8.2.2.1.1 Write Access from Master (see Figures 22 and 23)

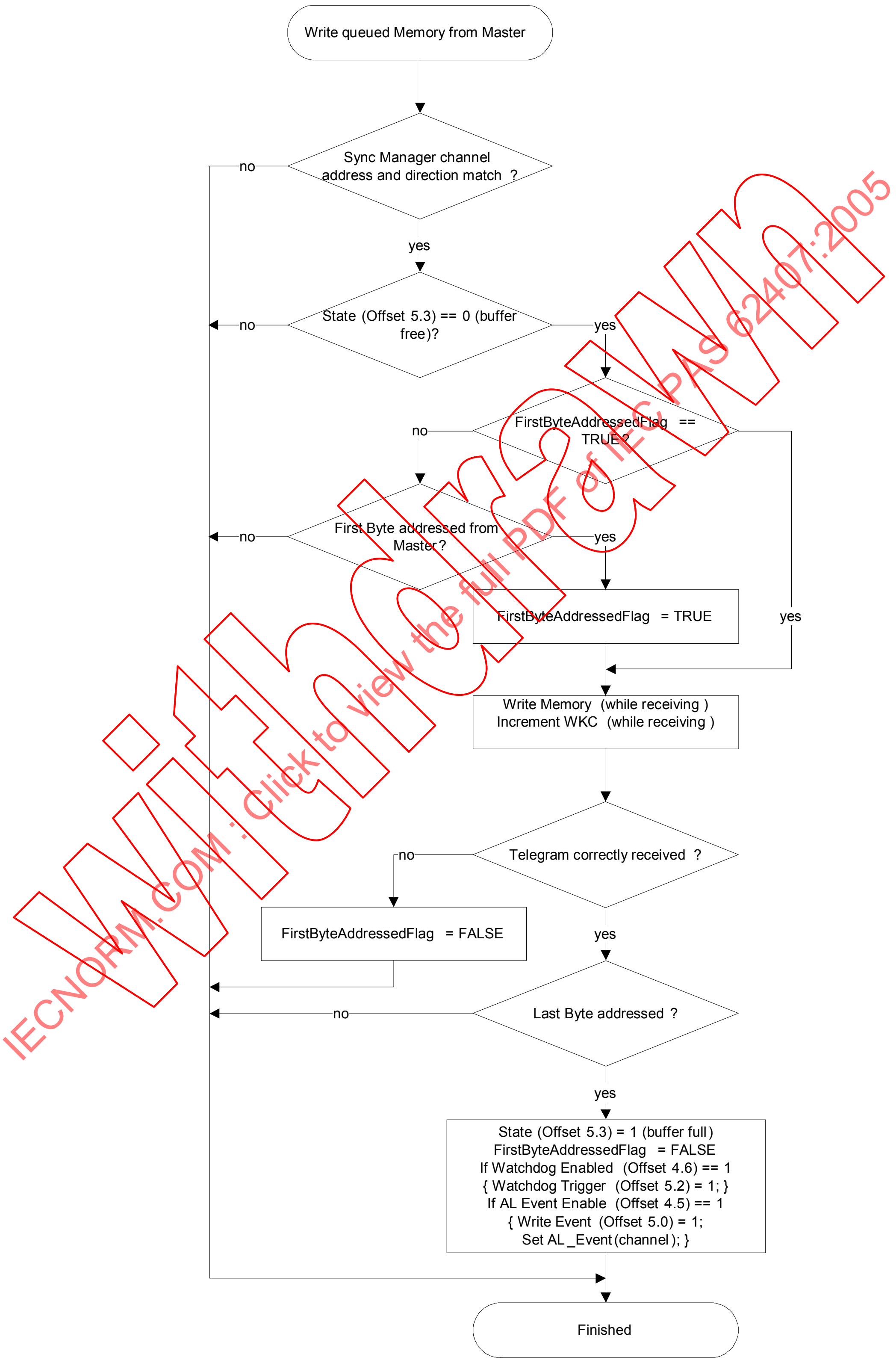

Write queued Memory from Master
Sync Manager channel address and direction match ?
no
yes
State (Offset 5.3) == 0 (buffer free)?
no
yes
FirstByteAddressedFlag == TRUE ?
no
yes
First Byte addressed from Master ?
no
yes
FirstByteAddressedFlag = TRUE
Write Memory (while receiving) Increment WKC (while receiving)
Telegram correctly received ?
no
yes
FirstByteAddressedFlag = FALSE
Last Byte addressed ?
no
yes
State (Offset 5.3) = 1 (buffer full) FirstByteAddressedFlag = FALSE If Watchdog Enabled (Offset 4.6) == 1 { Watchdog Trigger (Offset 5.2) = 1; } If AL Event Enable (Offset 4.5) == 1 { Write Event (Offset 5.0) = 1; Set AL_Event(channel); }
Finished

Figure 22: Write Queued Access Type Memory from Master

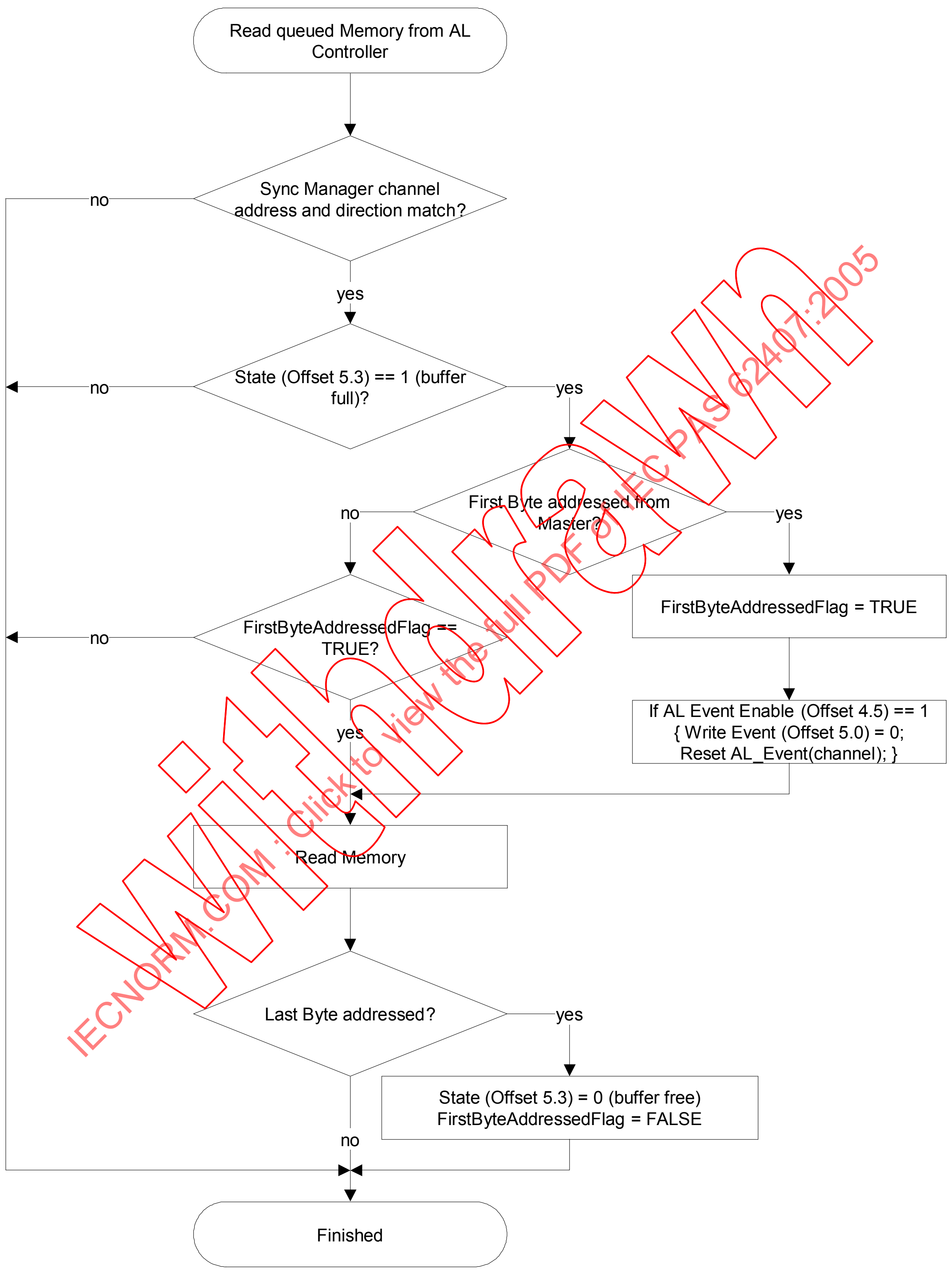

Read queued Memory from AL Controller
Sync Manager channel address and direction match?
no
yes
State (Offset 5.3) == 1 (buffer full)?
no
yes
First Byte addressed from Master?
no
yes
FirstByteAddressedFlag = TRUE
If AL Event Enable (Offset 4.5) == 1 { Write Event (Offset 5.0) = 0; Reset AL_Event(channel); }
FirstByteAddressedFlag == TRUE?
no
yes
Read Memory
Last Byte addressed?
yes
no
State (Offset 5.3) = 0 (buffer free) FirstByteAddressedFlag = FALSE
Finished

Figure 23: Read Queued Access Type Memory from AL Controller

8.2.2.1.2 Read Access from Master (see Figures 24 and 25)

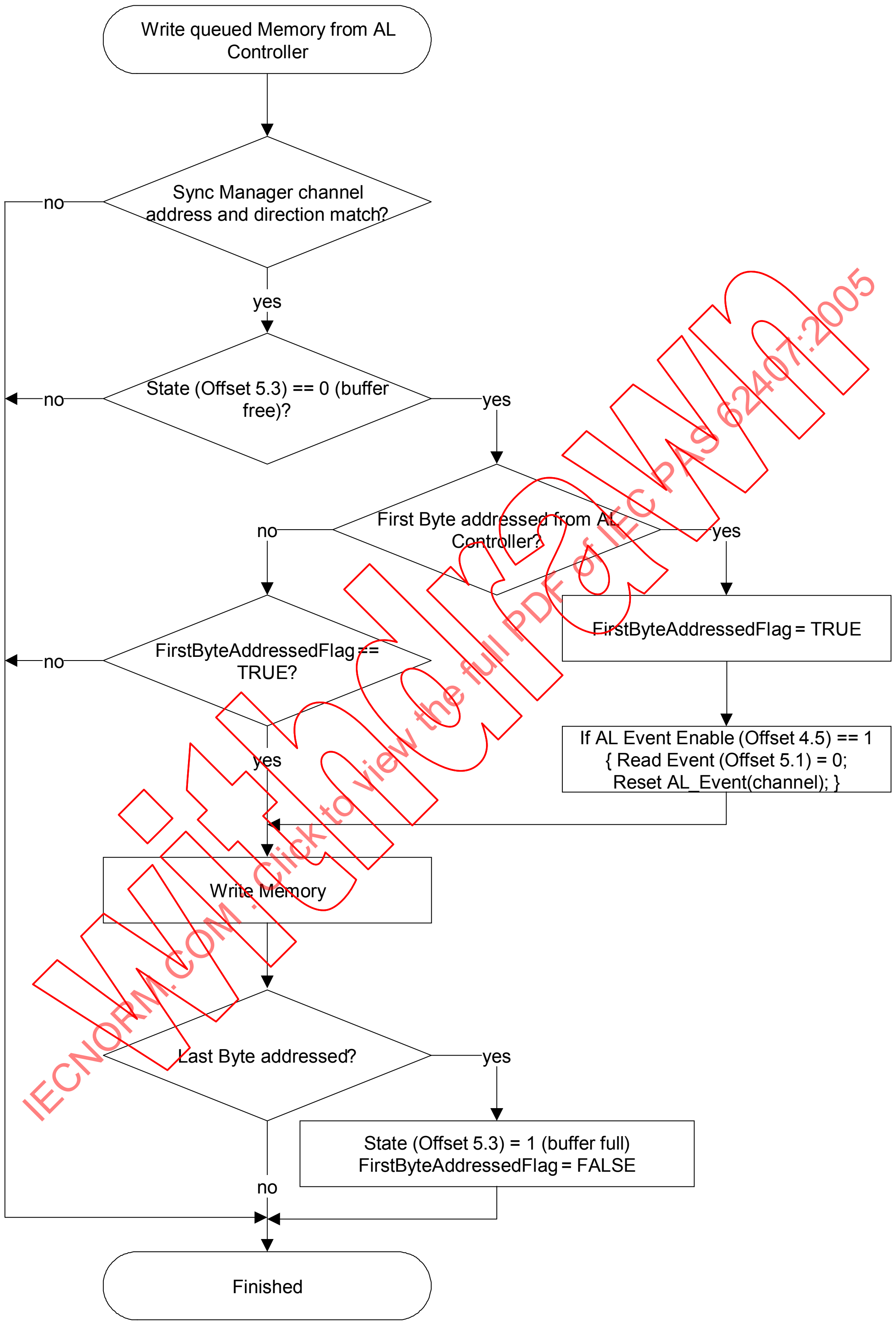

Write queued Memory from AL Controller
Sync Manager channel address and direction match?
no
yes
State (Offset 5.3) == 0 (buffer free)?
no
yes
First Byte addressed from AL Controller?
no
yes
FirstByteAddressedFlag = TRUE
If AL Event Enable (Offset 4.5) == 1 { Read Event (Offset 5.1) = 0; Reset AL_Event(channel); }
FirstByteAddressedFlag == TRUE?
no
yes
Write Memory
Last Byte addressed?
yes
no
State (Offset 5.3) = 1 (buffer full) FirstByteAddressedFlag = FALSE
Finished

Figure 24: Write Queued Access Type Memory from AL Controller

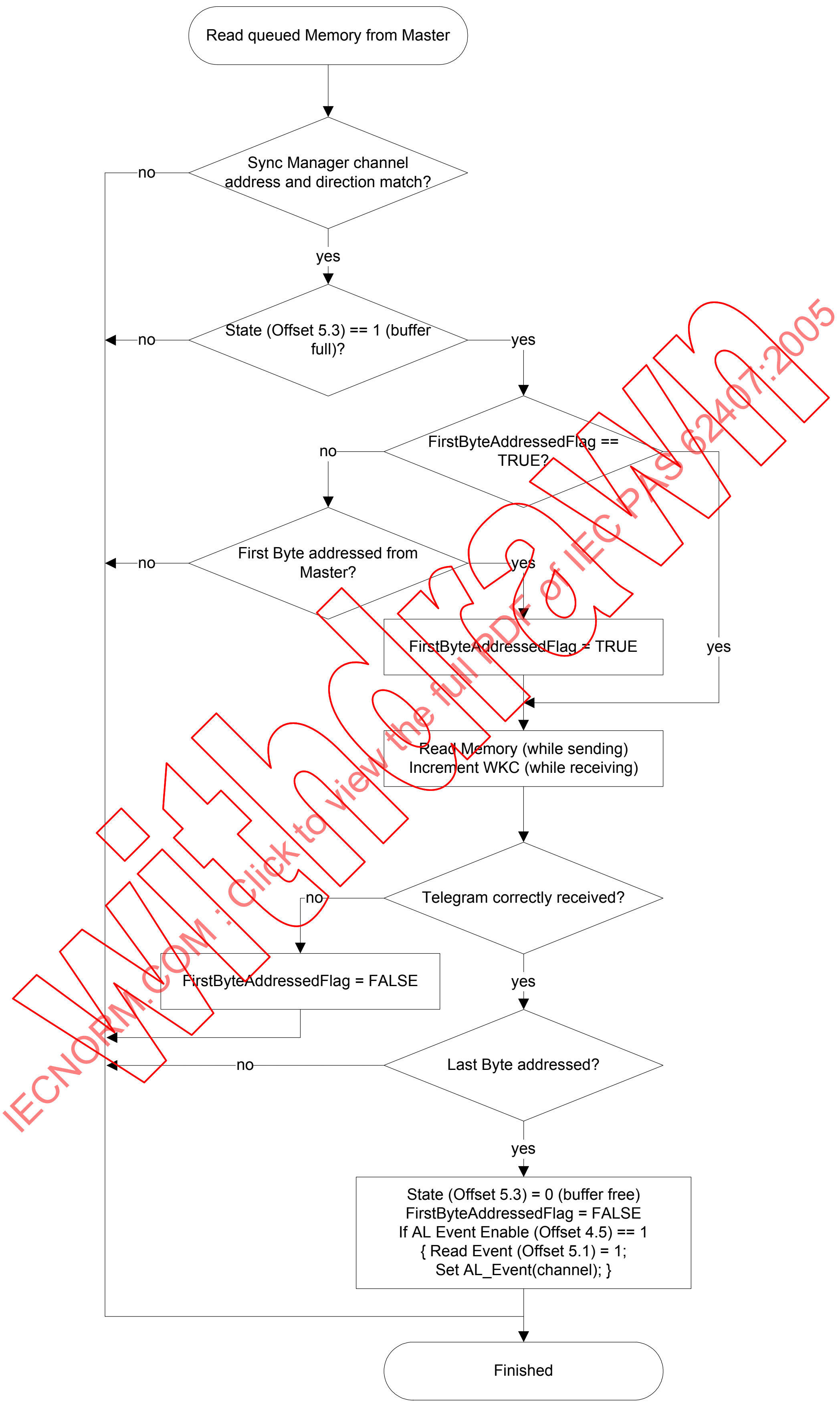

Read queued Memory from Master
Sync Manager channel address and direction match?
no
yes
State (Offset 5.3) == 1 (buffer full)?
no
yes
FirstByteAddressedFlag == TRUE?
no
yes
First Byte addressed from Master?
no
yes
FirstByteAddressedFlag = TRUE
Read Memory (while sending) Increment WKC (while receiving)
Telegram correctly received?
no
FirstByteAddressedFlag = FALSE
yes
Last Byte addressed?
no
yes
State (Offset 5.3) = 0 (buffer free)
FirstByteAddressedFlag = FALSE
If AL Event Enable (Offset 4.5) == 1
{ Read Event (Offset 5.1) = 1;
Set AL_Event(channel); }
Finished

Figure 25: Read Queued Access Type Memory from Master

8.2.2.2 Buffered Access Type Flow Charts

8.2.2.2.1 Write Access from Master (see Figures 26 and 27)

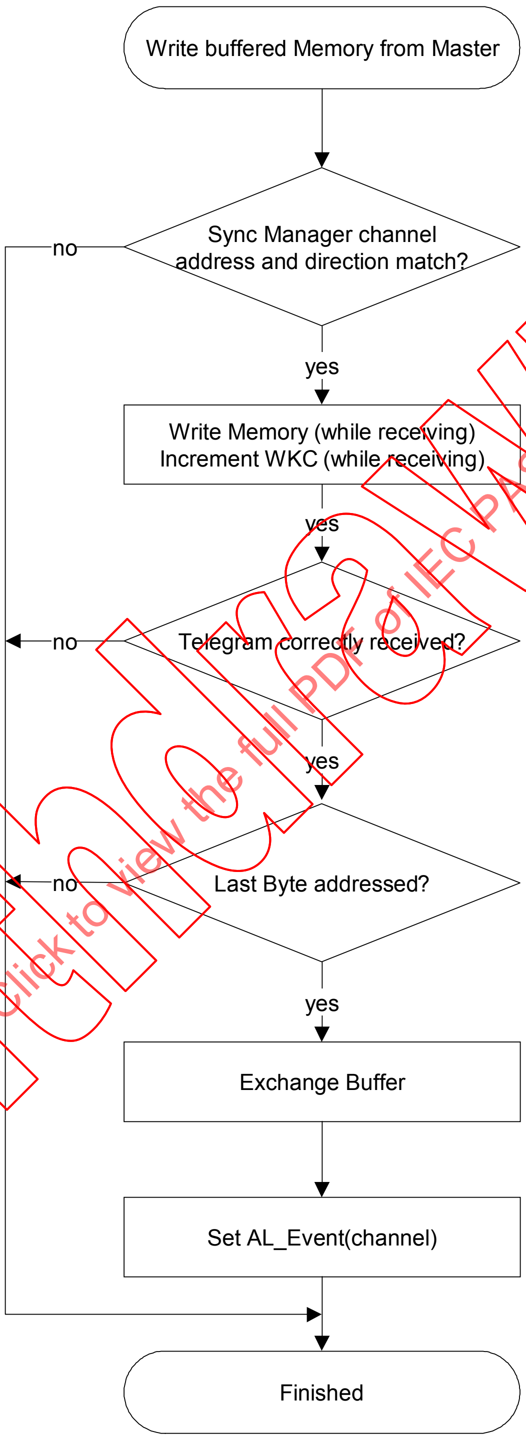

Write buffered Memory from Master
Sync Manager channel address and direction match?
no
yes
Write Memory (while receiving)
Increment WKC (while receiving)
yes
Telegram correctly received?
no
yes
Last Byte addressed?
no
yes
Exchange Buffer
Set AL_Event(channel)
Finished

Figure 26: Write Buffered Access Type Memory from Master

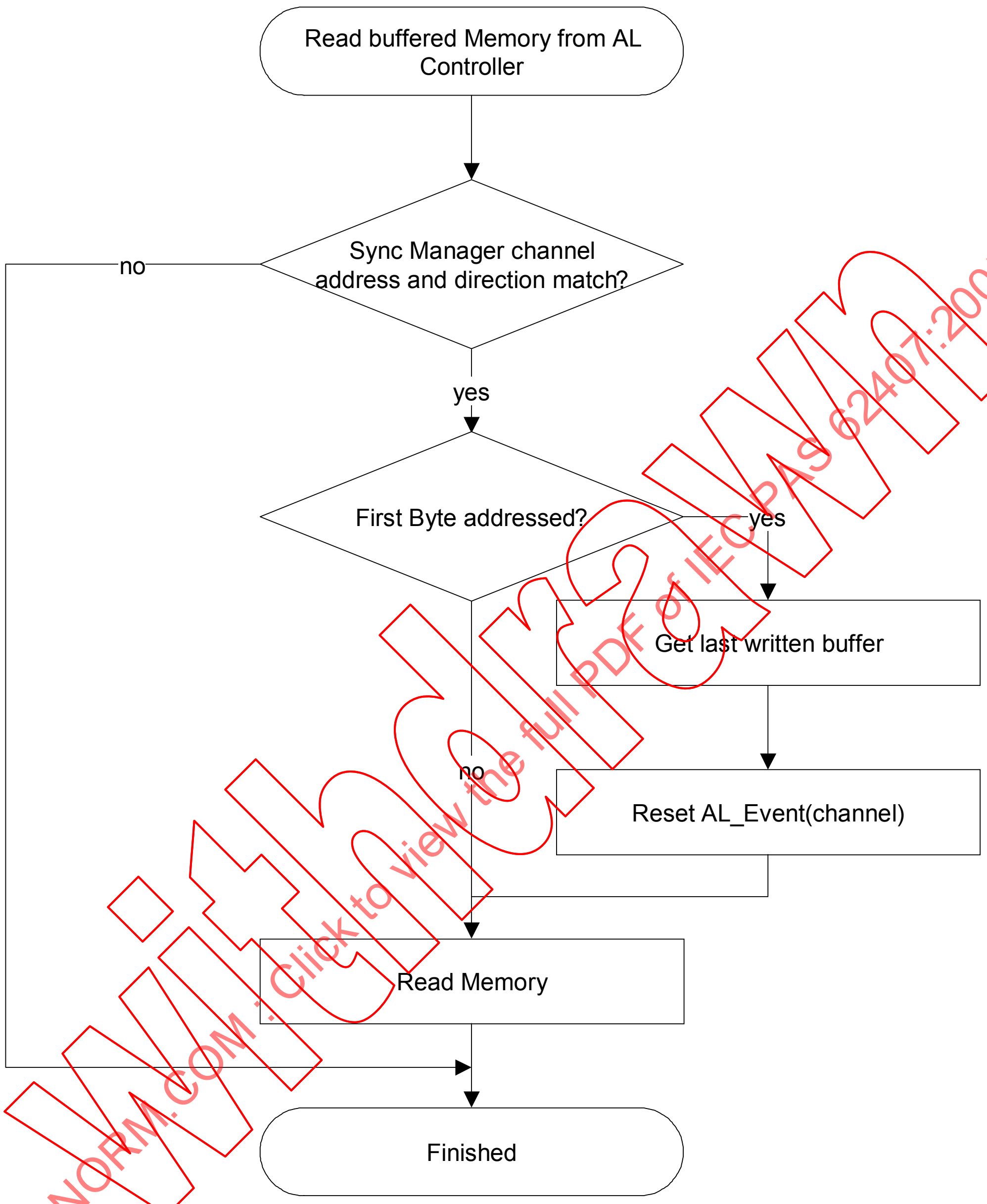


Figure 27: Read 3 Buffered Access Type Memory from AL Controller

8.2.2.2.2 Read Access from Master (see Figures 28 and 29)

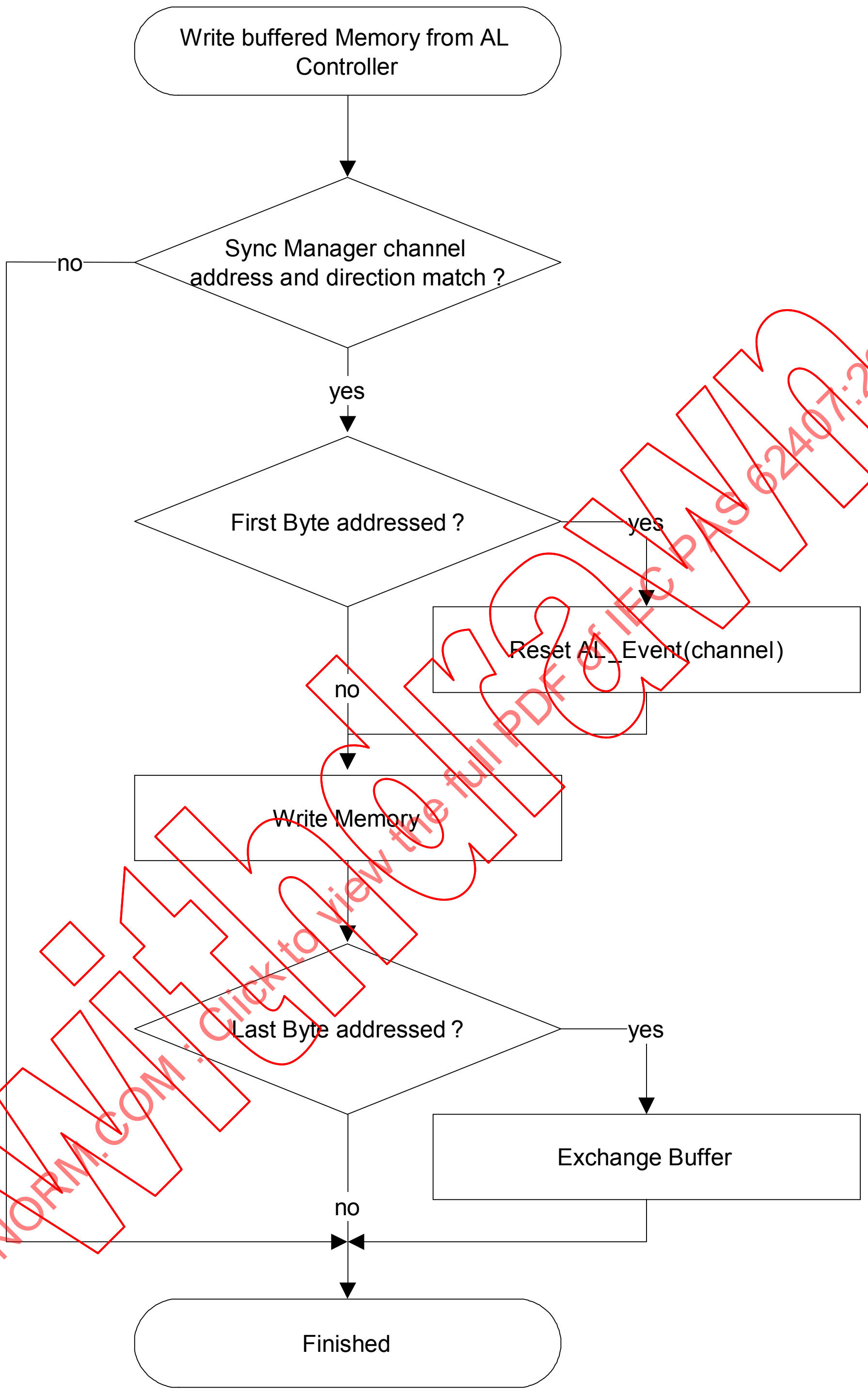


Figure 28: Write Buffered Access Type Memory from AL Controller

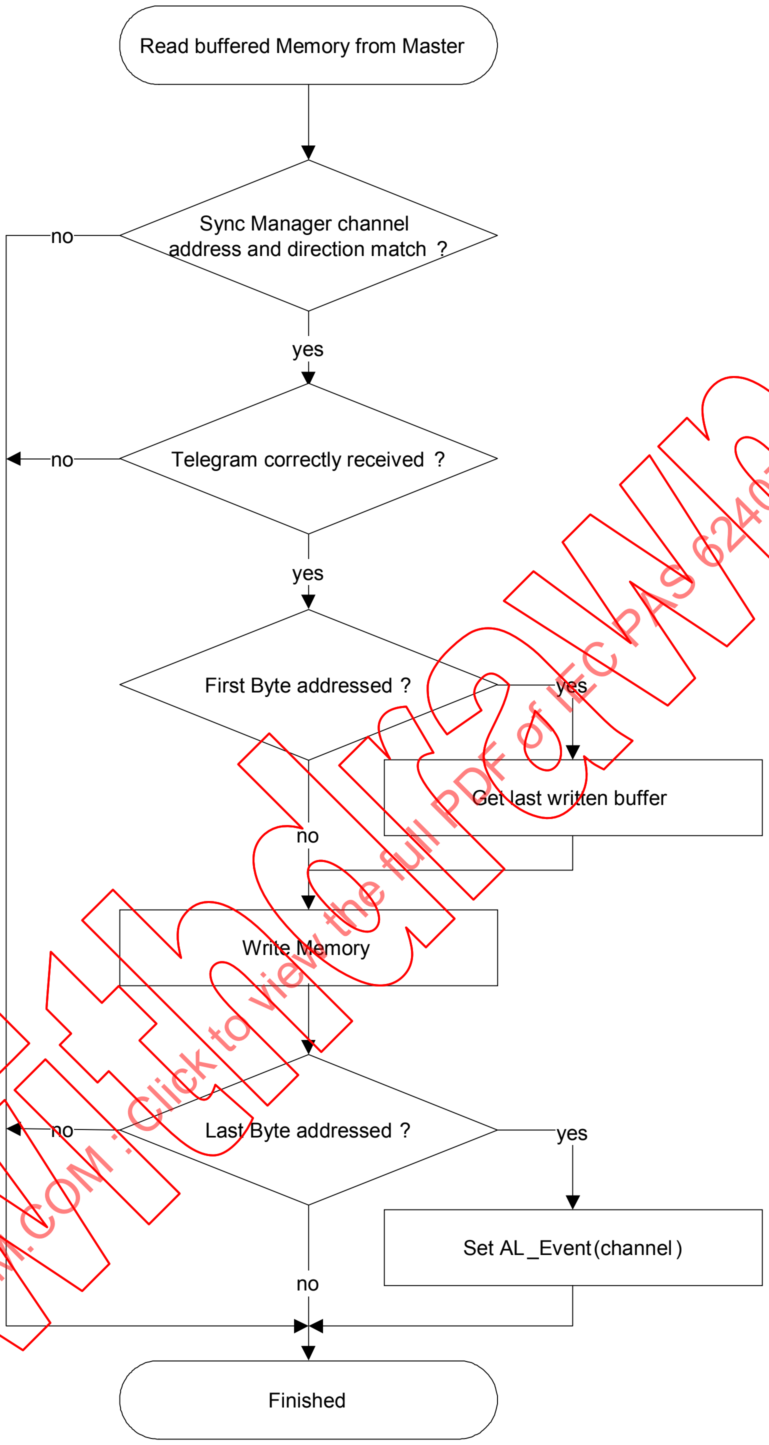


Figure 29: Read Buffered Access Type Memory from Master

9 Contents of Part 5: Application Layer Service definition

9.1 Communication Model Overview

The EtherCAT Application Layer distinguishes between master and slave. The communication relationship is always initiated by the master. The communication between two slaves is supported but in that case a master is involved for the routing. The communication between two masters is not supported but two devices with master functionality may communicate with each other if one of the devices supports slave functionality as well.

An EtherCAT communication network consists of at least one master device and one or many slave devices. All devices shall support the EtherCAT State Machine (ESM) and should support the transmission of EtherCAT Process Data.

9.2 Slave

9.2.1 Slave AL Classification

From the application layer point of view, slave devices are classified in simple devices without an application controller and more complex devices with an application controller.

NOTE: The DL slave classification in Basic Slaves and Full Slaves is independent of the AL view, since DL addressing mechanisms are invisible at the AL interface.

9.2.2 Simple Slave Device

Simple devices have a fixed process data layout, which shall be described in the device description file. Simple devices may confirm the AL Management services of the ESM without a reaction within the local application, as noted in Figure 30.

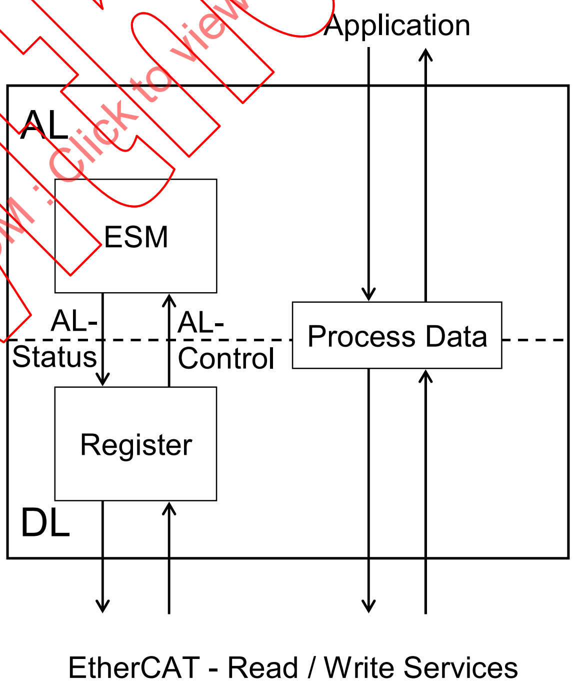


Figure 30: Simple Slave Device

9.2.3 Complex Slave Device

As noted in Figure 31, Complex devices shall support

- the Mailbox,
- the CoE object dictionary,
- the expedited SDO services to read and/or write the object dictionary data entries, and
- the SDO Information service to read the defined objects in the object dictionary and each entry description in compact format.

For the Process Data transmission the PDO mapping objects and the Sync Manager PDO Assign objects, which describe the process data layout, shall be supported for reading. If a complex device supports configurable process data, the configuration shall be done by writing the PDO mapping and/or the Sync Manager PDO Assign objects.

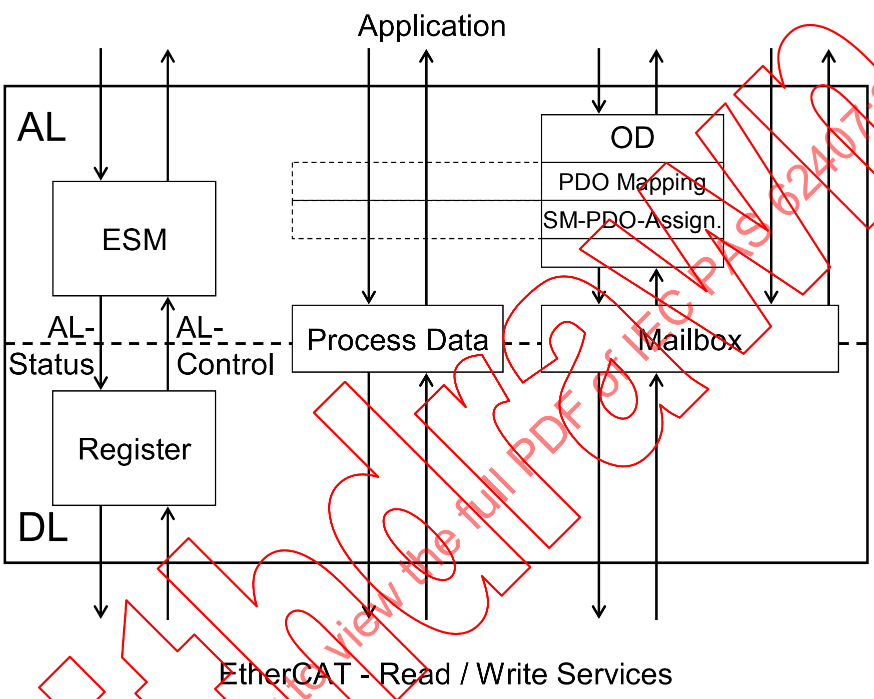


Figure 31: Complex Slave Device

9.2.4 Management

9.2.4.1 Overview

The AL Management includes the EtherCAT State Machine (ESM) which describes the states and state changes of the slave application. The actual state of a slave application shall be reflected in the AL Status Register by the application and requested state changes are indicated in the AL Control Register by the master.

The ESM logically is located between the EtherCAT Slave Controller (ESC) and the application.

The ESM defines four states, which shall be supported:

- Init,
- Pre-Operational,
- Safe-Operational, and
- Operational.

All state changes are possible except for the 'Init' state, where only the transition to the 'Pre-Operational" state is possible and for the 'Pre-Operational' state, where no direct state change to 'Operational' exists.

State changes are normally requested by the master. The master requests the AL Control service which results in an AL Control indication in the slave through an AL Control event. The slave shall respond to the AL Control service through a local AL Status write service after a successful or a failed state change. If the requested state change failed, the slave shall respond with the old state and the error flag set.

The Bootstrap state is optional and there is only a transition from or to the Init state. The only purpose of this state is to download the device's firmware. In Bootstrap state the mailbox is active but restricted to the FileAccess over EtherCAT (FoE) protocol.

The EtherCAT State Machine is specified in Figure 32.

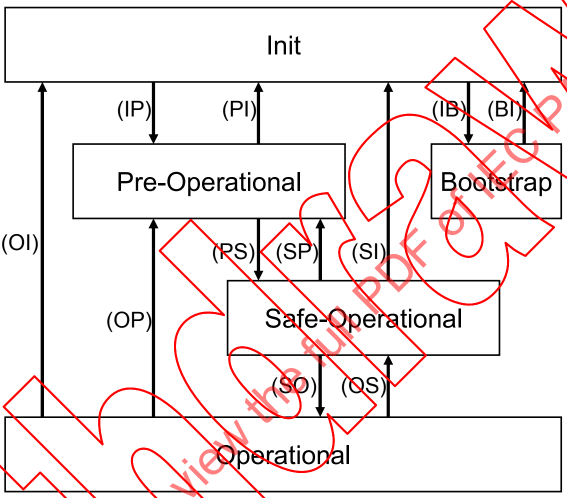


Figure 32: EtherCAT State Machine and services

The local management services are related to the transitions in the ESM, as specified in Table 55. If there is more than one service related to the transition, the slave's application will process all of the related services:

Table 55 – State transitions and local management services

state transition	local management service
IP	Start Mailbox Communication
PI	Stop Mailbox Communication
PS	Start Input Update
SP	Stop Input Update
SO	Start Output Update
OS	Stop Output Update
OP	Stop Output Update, Stop Input Update
SI	Stop Input Update, Stop Mailbox Communication
OI	Stop Output Update, Stop Input Update, Stop Mailbox Communication
IB	Start Bootstrap Mode
BI	Restart Device

9.2.4.2 EtherCAT States

9.2.4.2.1 Init

The ‘Init’ state defines the root of the communication relationship between the master and the slave in application layer. No direct communication between the master and the slave on application layer is possible. The master uses the ‘Init’ state to initialize a set of configuration register of the ECSC. If the slave supports an EtherCAT mailbox, the corresponding sync manager configurations are also done in the ‘Init’ state.

9.2.4.2.2 Pre-Operational

In the ‘Pre-Operational’ state the EtherCAT mailbox is active, if the slave supports the optional mailbox. Both the master and the slave can use the mailbox and the appropriate protocols to exchange application specific initializations and parameters. No process data communication is possible in this state.

9.2.4.2.3 Safe-Operational

In the ‘Safe-Operational’ state the application of the slave shall deliver actual input data without manipulating the output data. The outputs shall be set to their “safe state”.

9.2.4.2.4 Operational

In the ‘Operational’ state the application of the slave shall deliver actual input data and application of the master shall deliver actual output data.

9.2.4.2.5 Bootstrap

In the optional ‘Bootstrap’ state the application of the slave shall be able to accept a new firmware downloaded with the FoE protocol.

9.2.4.3 EtherCAT State Services

The primitives of the EtherCAT State services are mapped to the AL management primitives described in the DL as specified in Table 56.

Table 56 – AL management and ESM service primitives

AL management primitives	EtherCAT state service primitives
AL Control.ind:	WriteRegisterEvent(AL Control), ReadRegisterLocal(AL Control)
AL Control.rsp:	WriteRegisterLocal(AL Status)
AL State Acknowledge.ind:	WriteRegisterEvent(AL Control), ReadRegisterLocal(AL Control)
AL State Acknowledge.rsp:	WriteRegisterLocal(AL Status)
AL State Changed.req:	WriteRegisterLocal(AL Status)

9.2.4.3.1 AL Control Sequence

The primitives between master and slave in case of a successful AL Control sequence are specified in Figure 33.

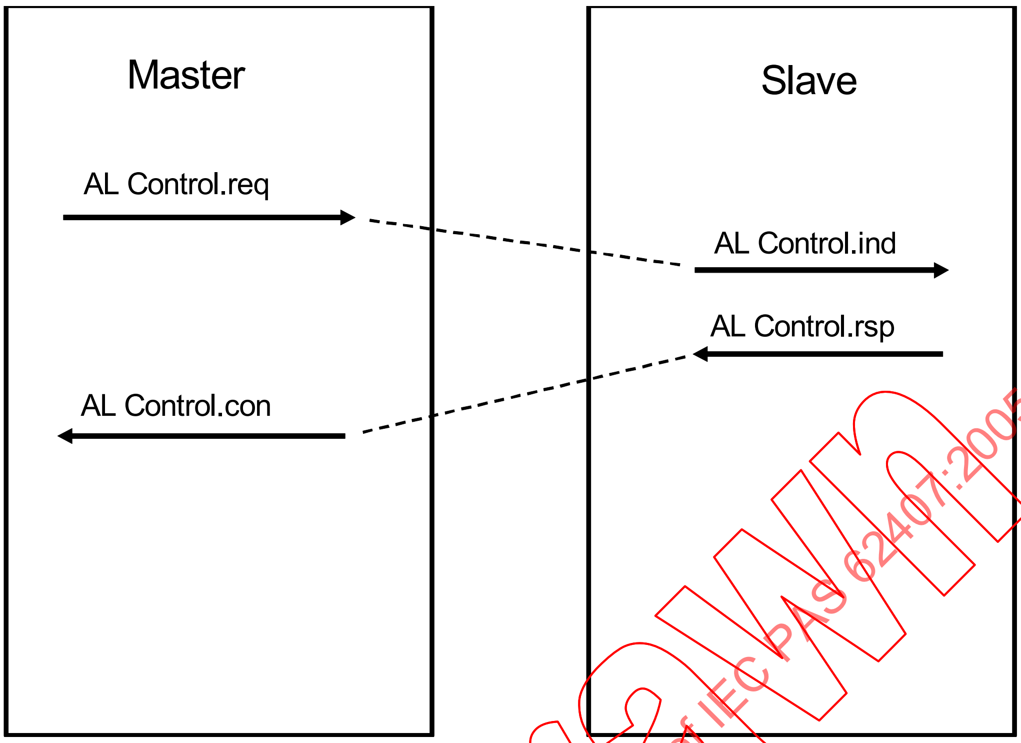


Figure 33: Successful AL Control sequence

The primitives between master and slave in case of a unsuccessful AL Control sequence are specified in Figure 34.

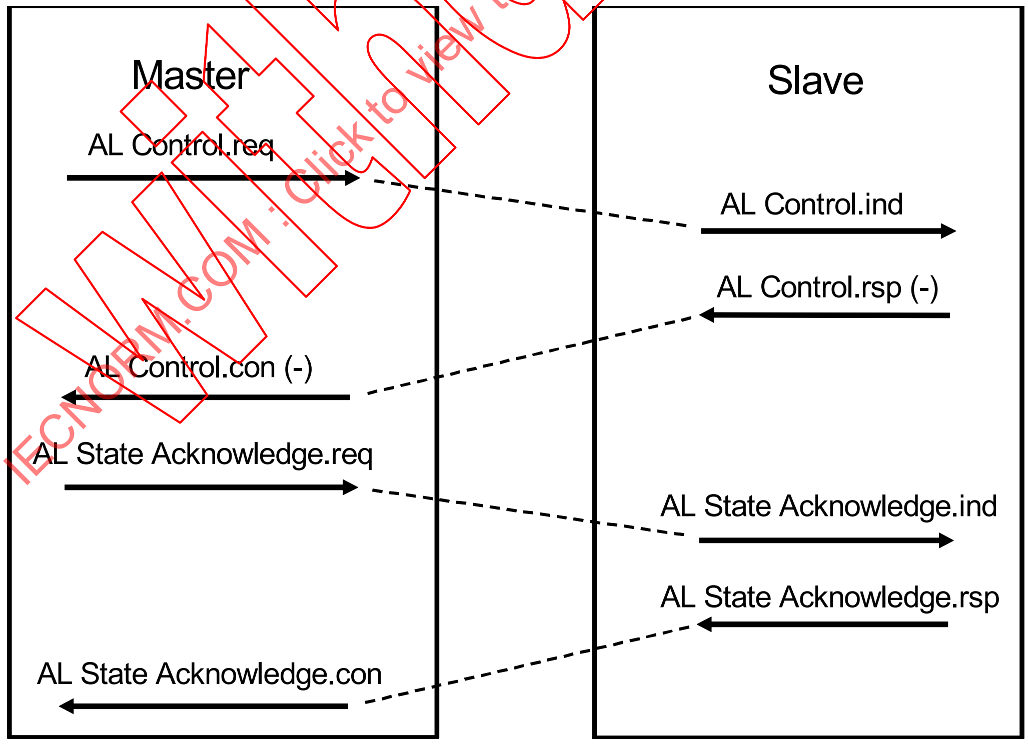


Figure 34: Unsuccessful AL Control sequence

9.2.4.3.2 AL State Changed Sequence

The primitives between master and slave in case of a AL State Changed sequence are specified in Figure 35.

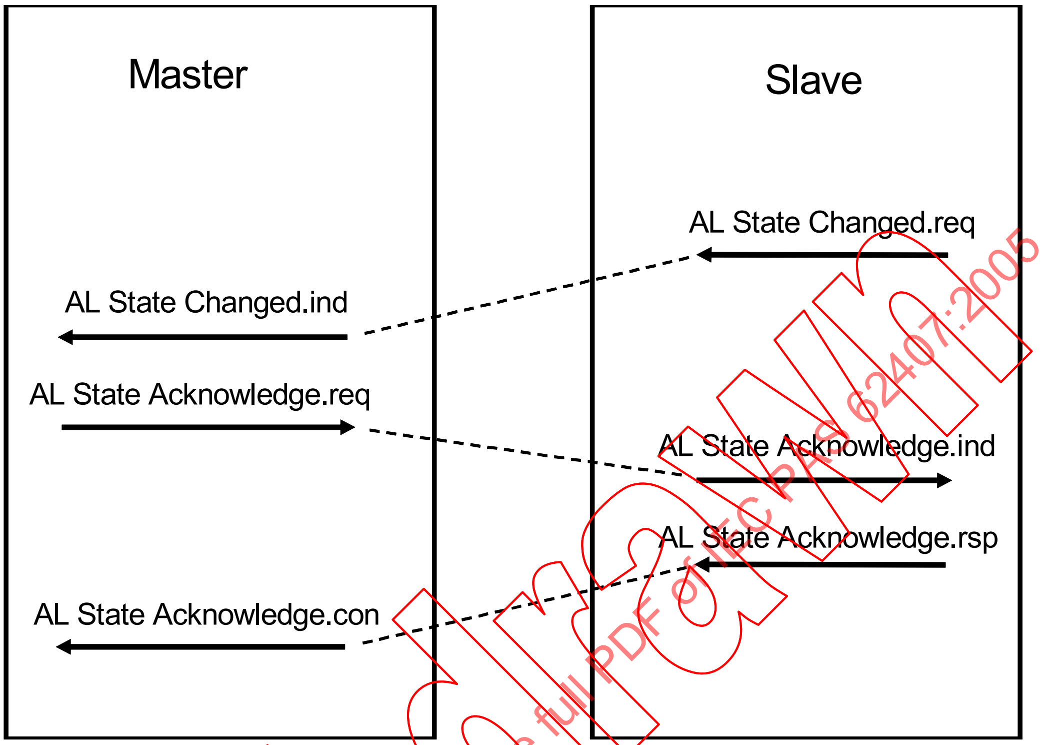


Figure 35: AL State Changed sequence

9.2.4.3.3 AL Control Service

The AL Control service as specified in Table 57 shall be used from the master to request a state transition in the slave's application by writing the AL Control register. The slave shall confirm the state transition by writing the AL Status register, which will be read from the master to get the confirmation.

In case of an unsuccessful state transition (parameter Error Flag = TRUE), the master shall acknowledge this service with the AL State Acknowledge service.

Table 57 – AL Control Service

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
AL Control	Mandatory	
Result		Mandatory
AL State		Mandatory
Error Flag		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

AL Control

This parameter indicates the state which is requested by the master in the slave's application.

Result

The result shall convey the service specific parameters of the service response.

AL State

This parameter indicates the actual state of the slave's application.

Error Flag

This parameter shall be TRUE, if the parameter AL State of the response is unequal the requested AL State.

9.2.4.3.4 AL State Changed Service

Additionally the slave's application can indicate a local state transition by writing the AL Status register as specified in Table 58. The master shall acknowledge the state transition with AL State Acknowledge Service.

Table 58 – AL State Changed Service

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
AL State	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

AL State

This parameter indicates the changed state of the slave's application.

9.2.4.3.5 AL State Acknowledge Service

The AL State Acknowledge service as specified in Table 59 shall be used from the master to acknowledge an unexpected state transition in the slave's application by writing the AL Control register. The slave shall confirm the service by writing the Device Status register, which will be read from the master to get the confirmation.

Table 59 – AL State Acknowledge Service

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
AL Control	Mandatory	
Result		Mandatory
AL State		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

AL Control

This parameter indicates the state to be acknowledged.

Result

The result shall convey the service specific parameters of the service response.

AL State

This parameter indicates the actual state of the slave's application.

9.2.4.4 Local Management services

The local management services describe the behaviour of the master and slave in case of the allowed state transitions mentioned above.

9.2.4.4.1 Start Mailbox Communication

With the Start Mailbox Communication service the master's application requests the 'Pre-Operational' state of the slave's application, if the slave's application is in the state 'Init'.

The master shall configure the DLL registers and the sync manager channels for the mailbox before requesting this service. Then the master shall wait for the confirmation of the slave before sending other commands to the slave.

If the slave supports the EtherCAT mailbox, the slave's application shall read the settings of the sync manager channels 0 and 1 and initialize its mailbox handler appropriately before confirming the state change by writing the AL Status register of the EtherCAT Slave Controller. If the configuration of the sync manager channels 0 and 1 are correct the slave shall confirm this service with success.

If the slave does not support an EtherCAT mailbox, the EtherCAT Slave Controller shall be configured from the master to confirm the state change in the AL Status register immediately.

9.2.4.4.2 Stop Mailbox Communication

With the Stop Mailbox Communication service the master's application requests the 'Init' state of the slave's application, if the slave's is in the state 'Pre-Operational'.

If the slave supports the EtherCAT mailbox, the slave's application has to stop its mailbox handler before confirming the state change by writing the AL Status register of the EtherCAT Slave Controller. The slave shall confirm this service always with success.

If the slave does not support an EtherCAT mailbox, the EtherCAT Slave Controller shall be configured from the master to confirm the state change in the AL Status register immediately.

9.2.4.4.3 Start Input Update

With the Start Input Update service the master's application requests the 'Safe-Operational' state of the slave's application, if the slave's application is in the state 'Pre-Operational'.

The master shall configure the sync manager channels for the process data and the FMMU channels before requesting this service. After requesting the state transition the master shall also start with the transmission of the process data services, but the master shall ignore the input data until the slave has confirmed the state transition.

If the slave supports the EtherCAT mailbox, the slave's application shall read the configuration of the sync manager channels configured for process data transfer and shall check if these settings match to its local process data configuration. If the checking was successful, the slave's application shall deliver valid input data before confirming the state transition. The output data of the slave shall stay in safe state. If the checking of the sync manager configuration was unsuccessful, the slave shall confirm this service with the 'Pre-Operational' state and sets the Error Flag parameter TRUE.

If the slave does not support an EtherCAT mailbox, the EtherCAT Slave Controller shall be configured from the master to confirm the state change in the AL Status register immediately.

9.2.4.4.4 Stop Input Update

With the Stop Input Update service the master's application requests the 'Pre-Operational' state of the slave's application, if the slave's application is in the state 'Safe-Operational'.

The master shall stop transmission of process data requests before requesting the state transition.

If the slave supports the EtherCAT mailbox, the slave's application shall stop updating the input data before confirming the state transition. The slave shall confirm this service always with success.

If the slave does not support an EtherCAT mailbox, the EtherCAT Slave Controller shall be configured from the master to confirm the state change in the AL Status register immediately.

The slave's application can use this service to indicate a local state transition, for example because of an unexpected error.

9.2.4.4.5 Start Output Update

With the Start Output Update service the master's application requests the 'Operational' state of the slave's application, if the slave's application is in the state 'Safe-Operational'.

The master shall deliver valid output data in the process data services before requesting the state transition.

If the slave supports the EtherCAT mailbox, the slave's application shall activate the valid output data received with the process data service before confirming the state transition. If the activation of the output data is not possible, the slave shall confirm this service with the 'Safe-Operational' state and sets the Error Flag parameter TRUE.

If the slave does not support an EtherCAT mailbox, the EtherCAT Slave Controller shall be configured from the master to confirm the state change in the AL Status register immediately.

9.2.4.4.6 Stop Output Update

With the Stop Output Update service the master's application requests the 'Safe-Operational' state of the slave's application, if the slave's application is in the state 'Operational'.

If the slave supports the EtherCAT mailbox, the slave's application shall set the output in the safe state before confirming the state transition. The slave shall confirm this service always with success.

If the slave does not support an EtherCAT mailbox, the EtherCAT Slave Controller shall be configured from the master to confirm the state change in the AL Status register immediately.

The slave's application can use this service to indicate a local state transition, for example because of an unexpected error.

9.2.4.4.7 Start Bootstrap Mode

With the Start Bootstrap Mode service the master's application requests the 'Bootstrap' state of the slave's application, if the slave's application is in the state 'Init'.

The master shall configure the sync manager channels for the mailbox before requesting this service. The sync manager configuration for the mailbox in Bootstrap state can differ from the configuration in the other states. Then the master shall wait for the confirmation of the slave before sending other commands to the slave.

The slave's application shall read the settings of the sync manager channels 0 and 1 and initialize its mailbox handler appropriately before confirming the state change by writing the AL Status register of the EtherCAT Slave Controller. If the slave's application supports the Bootstrap state and the configuration of the sync manager channels 0 and 1 are correct for the Bootstrap state the slave shall confirm this service with success.

9.2.5 Mailbox

9.2.5.1 Overview

The EtherCAT Mailbox is mandatory for each complex slave device.

The mailbox works in both directions – from the master to a slave and from a slave to the master. It supports full duplex, independent communication in both directions and multiple protocols. Slave to slave communication is managed by the master, operating as router. The mailbox header contains an address field that allows the master to redirect appropriate messages.

The mailbox uses the two first sync manager channels, one per each direction (sync manager channel 0 from the master to the slave and sync manager channel 1 from the slave to the master). The sync manager channels are configured to queued mode to prevent the other side from an overrun. Normally the mailbox communication is non cyclic and addresses a single slave. Therefore the physical addressing without the need of a FMMU is used instead of the logical addressing.

9.2.5.1.1 Communication from Master to Slave

The master has to check the working counter after sending a mailbox command to a slave. If the working counter did not increments (normally because of the slave has not completely read the last command) the master has to retransmit the mailbox command until the slave accepted it. No further error correction has to be done, this lies in the responsibility of higher protocols.

9.2.5.1.2 Communication from Slave to Master

The master has to determine that a slave has filled the sync manager 1 with a mailbox command and to send an appropriate read command as quickly as possible.

There are different ways to determine that a slave has filled its sync manager. A clever solution is to configure the "written bit" of the configuration header of sync manager 1 to a logical address and to read this bit cyclically. Using a logical address enables the possibility to read the bits from several slaves together and to configure each slave on an individual bit address. The drawback of this solution is that one FMMU per slave is needed.

Another solution is to simply poll the sync manager data area. The working counter of that read command will only be incremented once if the slave has filled the area with a new command.

9.2.5.2 Mailbox Transmission Services

The primitives of the Mailbox services are mapped to the queued type application memory primitives described in the DL:

Mailbox Write.ind:	Write Memory Event, Read Memory Local
Mailbox Read.req:	Write Memory Local, Read Memory Event

The following diagram shows the primitives between master and slave in case of a successful Mailbox Write sequence.

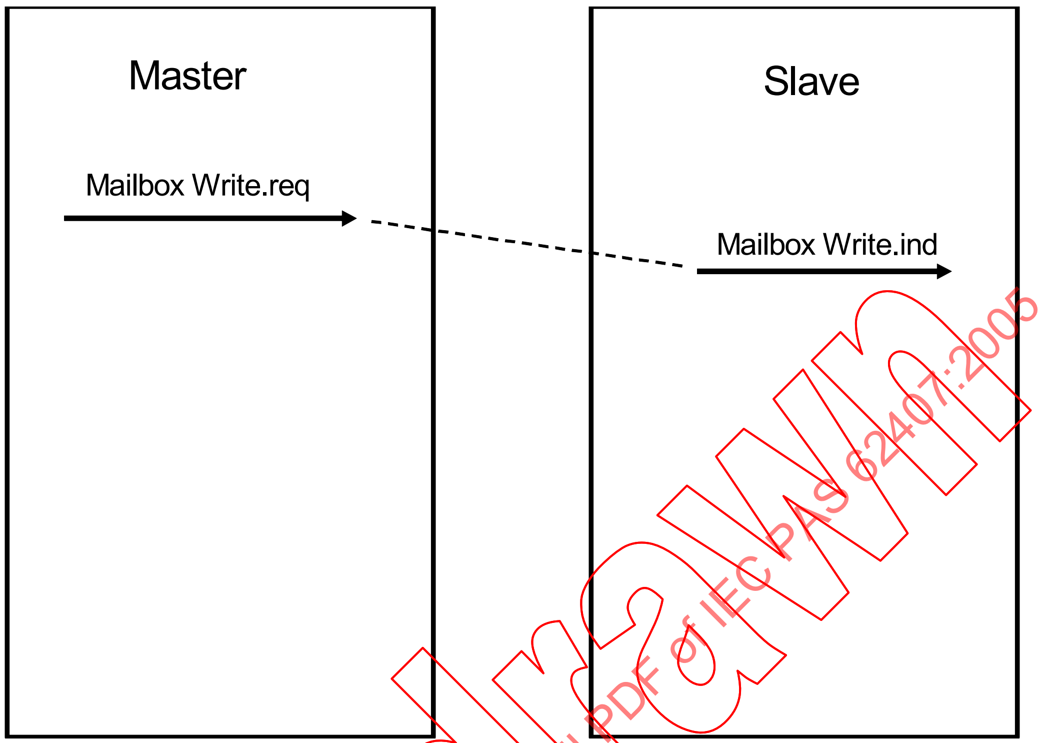


Figure 36: Successful Mailbox Write sequence

The following diagram shows the primitives between master and slave in case of a successful Mailbox Read sequence.

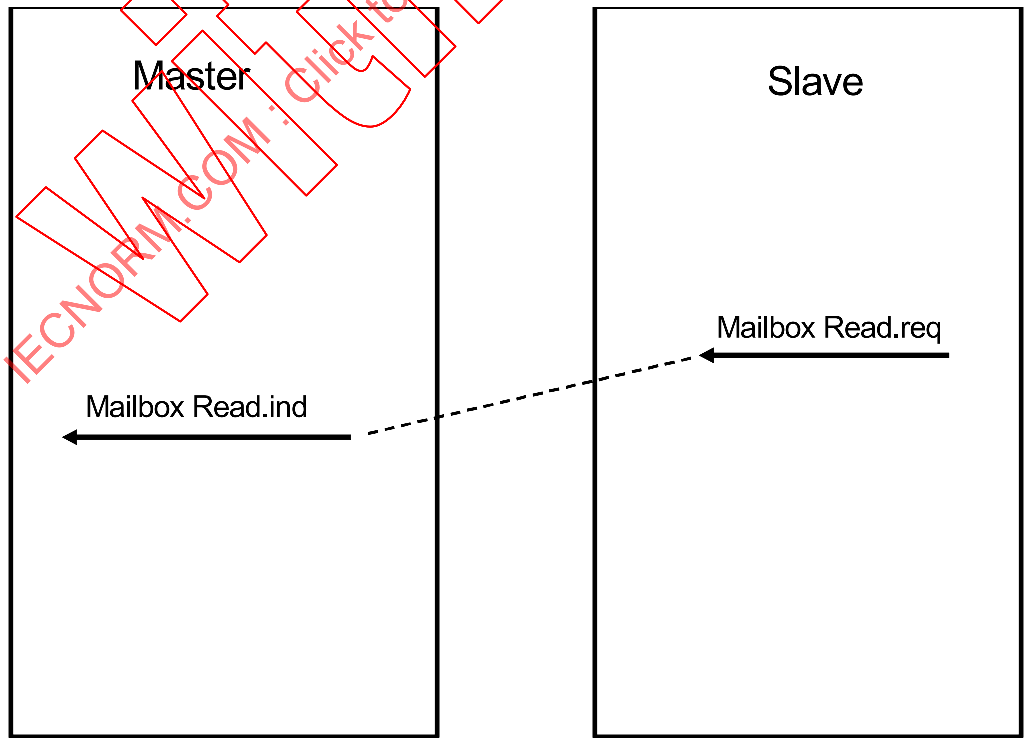


Figure 37: Successful Mailbox Read sequence

9.2.5.2.1 Mailbox Write

The Mailbox Write service as specified in Table 60 is based on writing (transmission from master to slave) queued type memory to get an acknowledged transmission of data.

Table 60 – Mailbox Write

Parameter	Request/Indication	Confirmation
Argument	Mandatory	
Length	Mandatory	
Address	Mandatory	
Channel	Mandatory	
Priority	Mandatory	
Type	Mandatory	
Service Data	Mandatory	
Result		Mandatory
Success		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Length

This parameter contains the size of the parameter Service Data.

Address

This parameter contains the station address of the source station to allow slave to slave communication.

Channel

This parameter contains the communication channel.

Priority

This parameter contains a communication priority.

Type

This parameter contains the protocol type of the used mailbox service.

Service Data

This parameter contains the service parameter of the used mailbox service.

Result

The result shall convey the service specific parameters of the service acknowledge.

Success

This parameter contains the information if the transmission was successful.

9.2.5.2.2 Mailbox Read

The Mailbox Read service as specified in Table 61 is based on reading (transmission from slave to master) queued type memory to get an acknowledged transmission of data.

Table 61 – Mailbox Read

Parameter	Request/Indication	Confirmation
Argument	Mandatory	
Length	Mandatory	
Address	Mandatory	
Channel	Mandatory	
Priority	Mandatory	
Type	Mandatory	
Service Data	Mandatory	
Result		Mandatory
Success		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Length

This parameter contains the size of the parameter Service Data.

Address

This parameter contains the station address of the destination to allow slave to slave communication.

Channel

This parameter contains the communication channel.

Priority

This parameter contains a communication priority.

Type

This parameter contains the protocol type of the used mailbox service.

Service Data

This parameter contains the service parameter of the used mailbox service.

Result

The result shall convey the service specific parameters of the service acknowledge.

Success

This parameter contains the information if the transmission was successful.

9.2.5.3 Mailbox protocols

The following protocol types are defined so far:

- CANopen over EtherCAT (CoE)
- Ethernet over EtherCAT (EoE)
- File Access over EtherCAT (FoE)

They are introduced within the next paragraphs.

9.2.6 EtherCAT Process Data

9.2.6.1 Overview

For the process data communication usually the application memory of the buffered type is used that master and slave always have access to the EtherCAT Process Data.

For complex slave devices the contents of the EtherCAT Process Data shall be described by the PDO Mapping and the Sync Manager PDO Assign objects of the CoE interface

For simple slave devices the EtherCAT Process Data is fixed and shall be defined in the device description file.

Although EtherCAT uses a master/slave communication model, communication between slaves can be created very easily using the features of the Fieldbus Memory Management Unit (FMMU). To this end, memory areas from the 4 GB logical address space are allocated for slave to slave communication and cyclically exchanged by the master. The master alternately issues a read query and, in the next cycle, a write command for the respective memory area. All slaves that are configured accordingly insert their slave to slave communication data or retrieve them during the next cycle. For the master, these data are transparent - it merely deals with the cyclic exchange.

Compared with party line bus systems, in which all devices are connected to the same communication medium, one cycle is 'wasted'. However, this is more than compensated for by the outstanding usable data rate and the associated short cycle times. The strategy described above also has the advantage, that the slave to slave communication data are collected from several sources and then simultaneously arrive at all addressed destinations during the next cycle. At a cycle time of, for example, 100 µs, approximately 1000 bytes can be sent from almost any number of sources to the same number of destinations.

Even more important is the fact that the slave to slave communication planned and initiated by the master is absolutely deterministic and therefore does not interfere with the cyclic process data exchange. The allocation of unnecessary bandwidth is thus avoided.

9.2.6.2 Process Output Data

The primitives of the Process Output Data services are mapped to the buffered type application memory primitives described in the DL:

Process Output Data Event:	Write Memory Event
Update Process Output Data Local:	Read Memory Local

The following diagram shows the primitives between master and the slaves for a Process Output Data sequence.

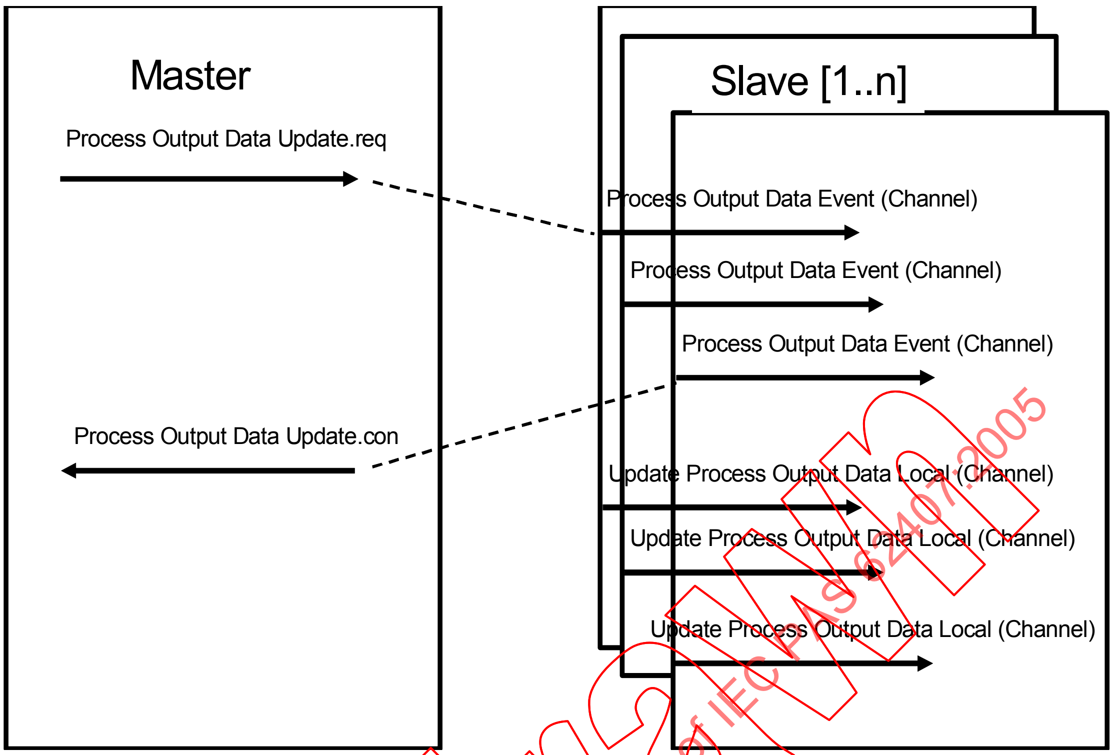

Master
Process Output Data Update.req
Process Output Data Update.con
Slave [1..n]
Process Output Data Event (Channel)
Process Output Data Event (Channel)
Process Output Data Event (Channel)
Update Process Output Data Local (Channel)
Update Process Output Data Local (Channel)
Update Process Output Data Local (Channel)

Figure 38: Process Output Data Sequence

The master usually sends process output data to several slaves. Each slave gets the AL Event of the corresponding Sync Manager channel(s). The slave's AL controller may read the process output data at any time from the related application memory.

9.2.6.3 Process Input Data

The primitives of the Process Input Data services are mapped to the buffered type application memory primitives described in the DL:

Update Process Input Data Local:	Write Memory Local
Process Input Data Event:	Read Memory Event

The following diagram shows the primitives between master and the slaves for a Process Input Data sequence.

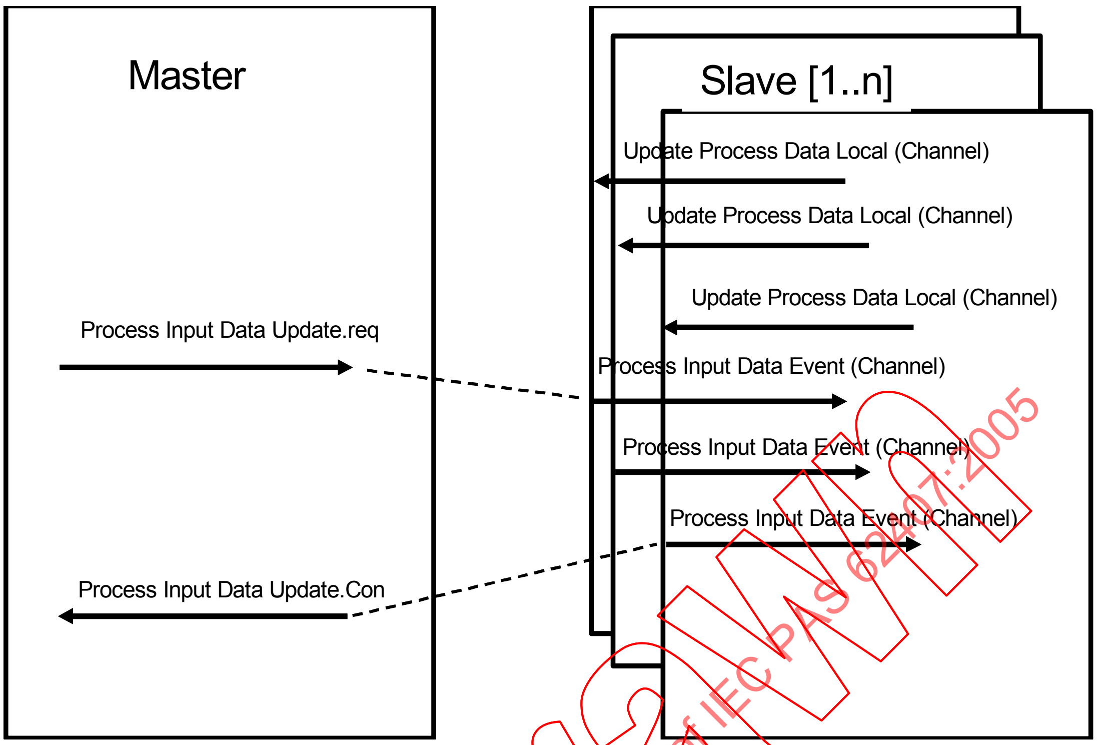

Master
Slave [1..n]
Update Process Data Local (Channel)
Update Process Data Local (Channel)
Update Process Data Local (Channel)
Process Input Data Update.req
Process Input Data Event (Channel)
Process Input Data Event (Channel)
Process Input Data Event (Channel)
Process Input Data Update.Con

Figure 39: Process Input Data Sequence

The master usually reads process input data from several slaves. Each slave gets the AL Event of the corresponding Sync Manager channel(s). The slave's AL controller may write the process input data at any time from the related application memory.

9.2.7 CANopen over EtherCAT

9.2.7.1 Overview

CANopen is a communication protocol originally developed for CAN-based systems. It is a standardized embedded network with highly flexible configuration capabilities. CANopen was designed for motion-oriented machine control networks, such as handling systems. By now it is used in a variety of applications areas, such as medical equipment, off-road vehicles, maritime electronics, public transportation, building automation, etc.

CANopen was pre-developed in an Esprit project. In 1995, the CANopen specification was handed over to the CAN in Automation (CiA) international users' and manufacturers' group. Version 4 of CANopen (CiA DS 301) is standardized as EN 50325-4.

Standardized profiles (device and application profiles) developed by CiA members simplify the system designer job of integrating a CANopen network system. CAN in Automation e.V. has made available those profiles to be used with EtherCAT.

The EtherCAT specification replaces the CANopen application layer and communication profile (DS301) and specifies an interface to the CANopen device and application profiles. These profiles are beyond of the scope of this specification.

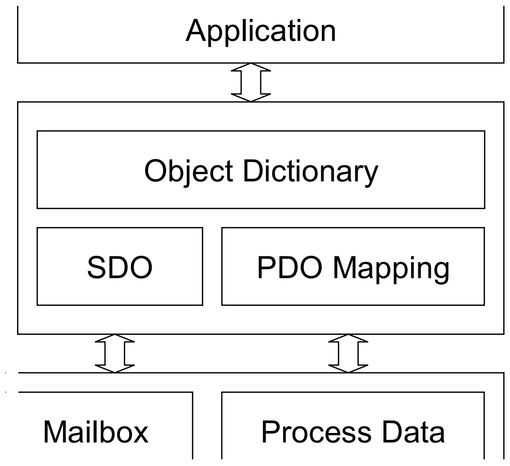


Figure 40: Server Model (CANopen part)

The Object Dictionary contains parameters, application data and the mapping information between process data interface and application date (PDO mapping). Its entries can be accessed via Service Data Objects (SDO), as noted in Figure 40.

9.2.7.2 Object Dictionary

9.2.7.2.1 Overview

The Object Dictionary contains all the CANopen related data objects of the device in a standardized way. It is a collection of the device parameters data structures that can be accessed with the SDO Upload and SDO Download services.

With the SDO Information services the available entries of the object dictionary and the entry description of an entry in the object dictionary can be read.

The object dictionary consists of the following areas:

Data Type Area:	Definition of the data types
CoE Communication Area:	Definition of the variables which may be used for all servers
Manufacturer Specific Area:	Definition of manufacturer specific variables
Device Profile Area:	Definition of the variables defined in a device profile
Reserved Area:	reserved for future use

9.2.7.2.2 Data Type Area

The Data Type Area consists of the following parts:

Static Data Types

  Definition of general simple data types

Complex Data Types

  Definition of general structured data types

Manufacturer Specific Complex Data Types

  Definition of manufacturer specific structured data types

Device Profile Specific Static Data Types

Definition of device profile specific simple data types

Device Profile Specific Complex Data Types

Definition of device profile specific structured data types

9.2.7.2.3 CoE Communication Area

9.2.7.2.3.1 Device Type

The Device Type object has the following parameter:

Parameter

Device Profile

This parameter contains the device profile that is used of the device.

Additional Information

This parameter contains an additional information defined by the device profile.

9.2.7.2.3.2 Manufacturer Device Name

The Manufacturer Device Name object has the following parameter:

Parameter

Device Name

This parameter contains the device name of the device.

9.2.7.2.3.3 Hardware Version

The Hardware Version object has the following parameter:

Parameter

Hardware Version

This parameter contains the hardware version of the device.

9.2.7.2.3.4 Software Version

The Software Version object has the following parameter:

Parameter

Device Name

This parameter contains the software version of the device.

9.2.7.2.3.5 Identity

The Identity object has the following parameter:

Parameter

Vendor ID

This parameter contains the vendor ID of the device assigned by the CiA.

Product Code

This parameter contains the product code of the device.

Major Revision Number

This parameter contains the major revision number of the device which shall identify the functionality of the device.

Minor Revision Number

This parameter contains the minor revision number of the device which shall identify different version with the same functionality.

Serial Number

This parameter contains the serial number of the device.

9.2.7.2.3.6 EtherCAT Fixed Station Address

The EtherCAT Fixed Station Address object has the following parameter:

Parameter

EtherCAT Fixed Station Address

This parameter contains the fixed station address of the DLL assigned by the master.

9.2.7.2.3.7 Virtual MAC Address

The Virtual MAC Address object has the following parameter:

Parameter

Virtual MAC Address

This parameter contains the virtual MAC address of the EoE interface.

9.2.7.2.3.8 IP Address Info

The IP Address Info object has the following parameter:

Parameter

IP Address

This parameter contains the IP address of the EoE interface.

Subnet mask

This parameter contains the subnet mask of the EoE interface.

Default Gateway

This parameter contains the default gateway of the EoE interface.

DNS Server

This parameter contains the DNS server of the EoE interface.

DNS Name

This parameter contains the DNS name of the EoE interface.

9.2.7.2.3.9 Receive PDO Mapping

The Receive PDO Mapping object has the following parameter:

Parameter

PDO Number

This parameter contains the number of the PDO.

Number of mapping entries

This parameter contains the number of mapping entries.

List of mapping entries

This parameter contains a list of mapping entries.

9.2.7.2.3.10 Transmit PDO Mapping

The Transmit PDO Mapping object has the following parameter:

Parameter

PDO Number

This parameter contains the number of the PDO.

Number of mapping entries

This parameter contains the number of mapping entries.

List of mapping entries

This parameter contains a list of mapping entries.

9.2.7.2.3.11 Sync Manager Communication Type

The Sync Manager Communication Type object has the following parameter:

Parameter

Number of Used Sync Manager Channels

This parameter contains the number of used Sync Manager Channels.

List of Sync Manager Communication Types

This parameter contains the communication type for each used Sync Manager Channel.

9.2.7.2.3.12 Sync Manager PDO Assignment

The Sync Manager PDO Assignement object has the following parameter:

Parameter

Channel Number

This parameter contains the channel number of the Sync Manager Channel.

Number of Assigned PDOs

This parameter contains the number of assigned PDOs.

List of assigned PDOs

This parameter contains the list of assigned PDOs.

9.2.7.2.3.13 EtherCAT Physical Memory Access

The EtherCAT Physical Memory Access object has the following parameter:

Parameter

Physical Memory Offset

This parameter contains the offset of the related physical memory of the EtherCAT slave.

Number of Physical Memory Words

This parameter contains the number of words of the related physical memory of the EtherCAT slave.

List of Physical Memory Words

This parameter contains the list of words of the related physical memory of the EtherCAT slave.

9.2.7.3 SDO Services

9.2.7.3.1 Overview

With the SDO services entries of the Object Dictionary can be read or written. The SDO transport protocol allows transmitting objects of any size. The EtherCAT SDO protocol is equivalent to the CANopen SDO protocol in order to support re-use of existing protocol stacks.

The first byte of the first segment contains the necessary flow control information. The next three byte of the first segment contain index and sub-index of the Object Dictionary entry to be read or written. The last four byte of the first segment are available for user data. The second and the following segments contain the control byte and user data. The receiver confirms each segment or a block of segments, so that a peer-to-peer communication (client/server) takes place.

In legacy mode the SDO protocol consists of 8 bytes only to match the CAN data size. In enhanced mode the payload data is simply enlarged without changing the protocol headers. In this way the larger data size of the EtherCAT mailbox is applied to the SDO protocol, the transmission of large data is accelerated accordingly.

Furthermore, a mode is added that allows one to transfer the entire data located at one index of the object dictionary at one go. The data of all sub-indices is then transferred subsequently.

The confirmed services (Initiate SDO Upload, Initiate SDO Download, Download SDO Segment, and Upload SDO Segment) and the unconfirmed service (Abort SDO Transfer) are used for Service Data Objects performing the segmented/expedited transfer.

The primitives of the SDO services are mapped to the primitives of the Mailbox Transmission services:

Slave is Server:

SDO Download.req/.ind:	Mailbox Write.req/.ind
SDO Download.rsp/.con:	Mailbox Read.req/.ind
Download SDO Segment.req/.ind:	Mailbox Write.req/.ind
Download SDO Segment.rsp/.con:	Mailbox Read.req/.ind
SDO Upload.req/.ind:	Mailbox Write.req/.ind
SDO Upload.rsp/.con:	Mailbox Read.req/.ind
Upload SDO Segment.req/.ind:	Mailbox Write.req/.ind
Upload SDO Segment.rsp/.con:	Mailbox Read.req/.ind
Abort SDO Transfer.req:	Mailbox Read.req/.ind

Slave is Client:

SDO Download.req/.ind:	Mailbox Read.req/.ind
SDO Download.rsp/.con:	Mailbox Write.req/.ind
Download SDO Segment.req/.ind:	Mailbox Read.req/.ind
Download SDO Segment.rsp/.con:	Mailbox Write.req/.ind
SDO Upload.req/.ind:	Mailbox Read.req/.ind
SDO Upload.rsp/.con:	Mailbox Write.req/.ind
Upload SDO Segment.req/.ind:	Mailbox Read.req/.ind
Upload SDO Segment.rsp/.con:	Mailbox Write.req/.ind
Abort SDO Transfer.req:	Mailbox Write.req/.ind

9.2.7.3.2 SDO Download

9.2.7.3.2.1.1 SDO Download Sequence

The following diagram shows the primitives between client and server in case of a successful single SDO Download sequence. The service parameter Address, Index and Subindex shall be the same in the request and the response.

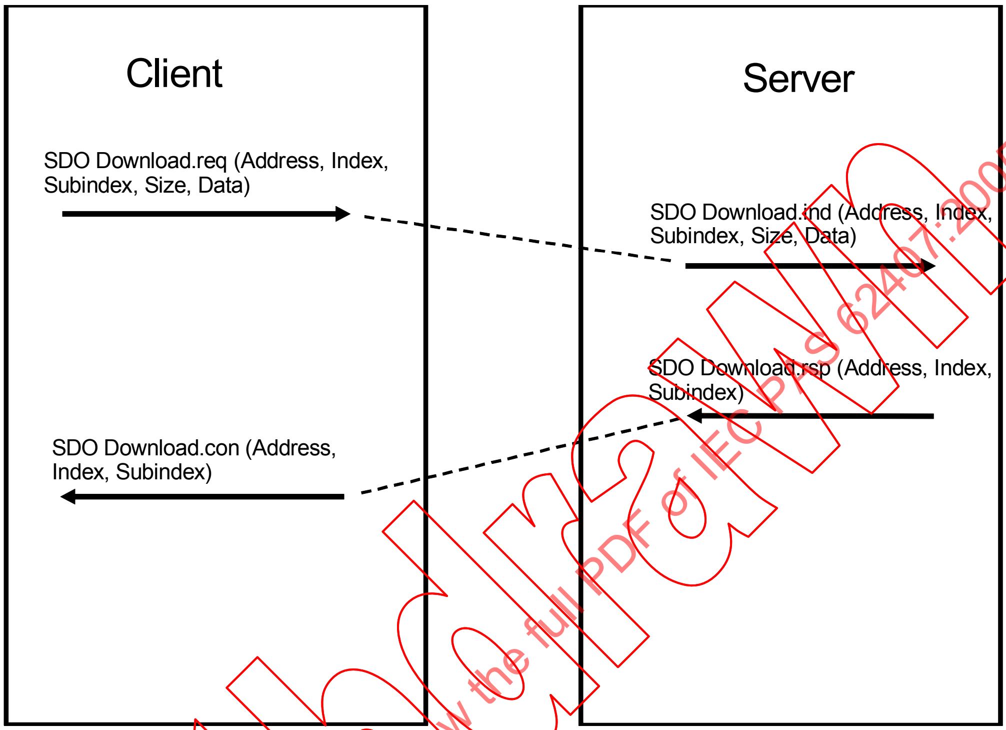

Client
Server
SDO Download.req (Address, Index, Subindex, Size, Data)
SDO Download.ind (Address, Index, Subindex, Size, Data)
SDO Download.rsp (Address, Index, Subindex)
SDO Download.con (Address, Index, Subindex)

Figure 41: Successful single SDO-Download sequence

The following diagram shows the primitives between client and server in case of a unsuccessful SDO Download sequence. The service parameter Address, Index and Subindex shall be the same in the Download request and the Abort request.

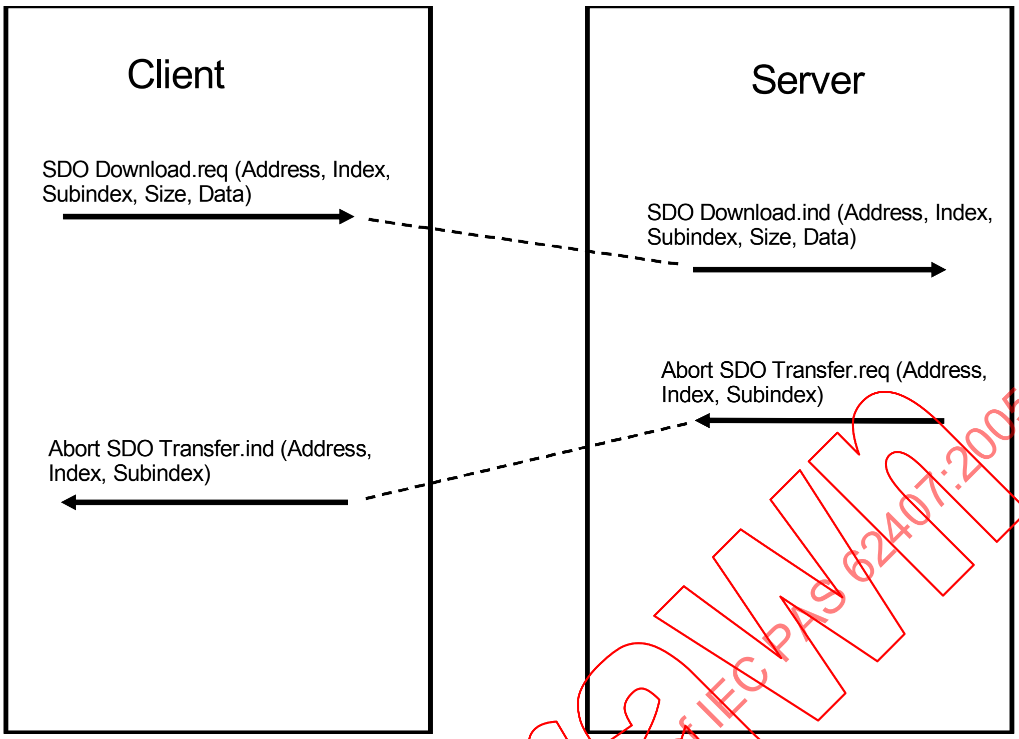

Client
SDO Download.req (Address, Index, Subindex, Size, Data)
Abort SDO Transfer.ind (Address, Index, Subindex)
Server
SDO Download.ind (Address, Index, Subindex, Size, Data)
Abort SDO Transfer.req (Address, Index, Subindex)

Figure 42: Unsuccessful single SDO-Download sequence

The following diagram shows the primitives between client and server in case of a successful segmented SDO Download sequence. The service parameter Address, Index and Subindex shall be the same in the request and the response.

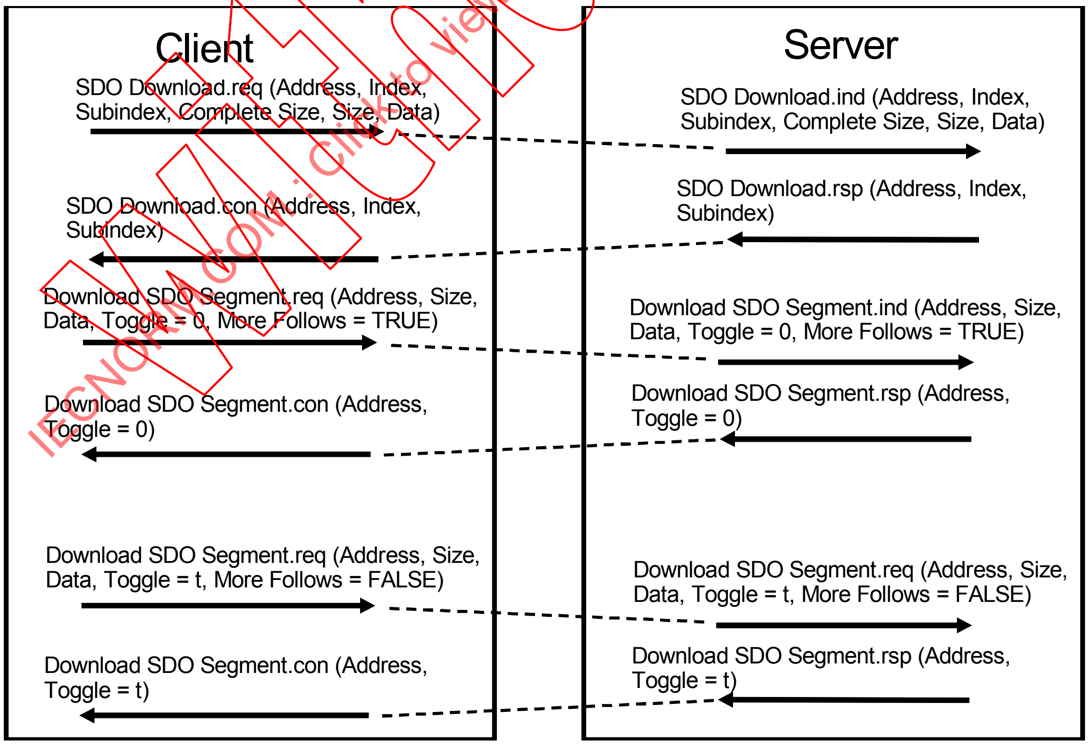

Client
SDO Download.req (Address, Index, Subindex, Complete Size, Size, Data)
SDO Download.con (Address, Index, Subindex)
Download SDO Segment.req (Address, Size, Data, Toggle = 0, More Follows = TRUE)
Download SDO Segment.con (Address, Toggle = 0)
Download SDO Segment.req (Address, Size, Data, Toggle = t, More Follows = FALSE)
Download SDO Segment.con (Address, Toggle = t)
Server
SDO Download.ind (Address, Index, Subindex, Complete Size, Size, Data)
SDO Download.rsp (Address, Index, Subindex)
Download SDO Segment.ind (Address, Size, Data, Toggle = 0, More Follows = TRUE)
Download SDO Segment.rsp (Address, Toggle = 0)
Download SDO Segment.req (Address, Size, Data, Toggle = t, More Follows = FALSE)
Download SDO Segment.rsp (Address, Toggle = t)

Figure 43: Successful segmented SDO-Download sequence

9.2.7.3.2.1.2 Initiate SDO Download Expedited

The Expedited SDO Download service shall be used for an expedited transfer of up to 4 bytes of data from the client to server. The server shall answer with the result of the download operation.

Table 62 – Initiate SDO Download Expedited

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Subindex	Mandatory	
Size	Mandatory	
Data	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Index		Mandatory
Subindex		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This parameter contains the index in the object dictionary of the object which should be downloaded.

Subindex

This parameter contains the subindex in the object dictionary of the object which should be downloaded.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the data to be downloaded.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Expedited SDO Download service was successful. In case of an error an Abort SDO Transfer service will be send from the server.

Address

This parameter shall be the same as the parameter Address in the request.

Index

This parameter shall be the same as the parameter Index in the request.

Subindex

This parameter shall be the same as the parameter Subindex in the request.

9.2.7.3.2.1.3 Initiate SDO Download Normal

The Initiate SDO Download service will be used for a single normal transfer of data from the client to the server, if the number of bytes to be downloaded fit in the mailbox, or to start a segmented transfer with more bytes.

Table 63 – Initiate SDO Download Normal

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Subindex	Mandatory	
Complete Size	Mandatory	
Size	Mandatory	
Data	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Index		Mandatory
Subindex		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This parameter contains the index in the object dictionary of the object which should be downloaded.

Subindex

This parameter contains the subindex in the object dictionary of the object which should be downloaded.

Complete Size

The parameter Complete Size indicates the number of bytes to be downloaded to the server. If the Complete Size is bigger than the Size parameter the difference number of bytes will be downloaded in the next Download SDO Segment services.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the data to be downloaded.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Initiate SDO Download service was successful. In case of an error an Abort SDO Transfer service will be send from the server.

Address

This parameter shall be the same as the parameter Address in the request.

Index

This parameter shall be the same as the parameter Index in the request.

Subindex

This parameter shall be the same as the parameter Subindex in the request.

9.2.7.3.2.1.4 Download SDO Segment

The SDO Download Segment service shall be used to download the additional data from the client to the server, which did not fit in the mailbox with Initiate SDO Download service. The master will start so many Download SDO Segment services until all data is downloaded to the server.

Table 64 – Download SDO Segment

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Toggle	Mandatory	
More Follows	Mandatory	
Size	Mandatory	
Data	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Toggle		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Toggle

This parameter shall be toggeled with every Download SDO Segmenet service, starting by zero.

More Follows

This parameter indicates, if this service is the last Download SDO Segment service to download the data for the object or if at least one Download SDO Segment service is following.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the data to be downloaded.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Initiate SDO Download service was successful. In case of an error an Abort SDO Transfer service will be send from the server.

Address

This parameter shall be the same as the parameter Address in the request.

Toggle

This parameter shall be the same as the parameter Toggle in the request.

9.2.7.3.3 SDO Upload

9.2.7.3.3.1.1 SDO Upload Sequence

The following diagram shows the primitives between client and server in case of a successful single SDO Upload sequence. The service parameter Address, Index and Subindex shall be the same in the request and the response.

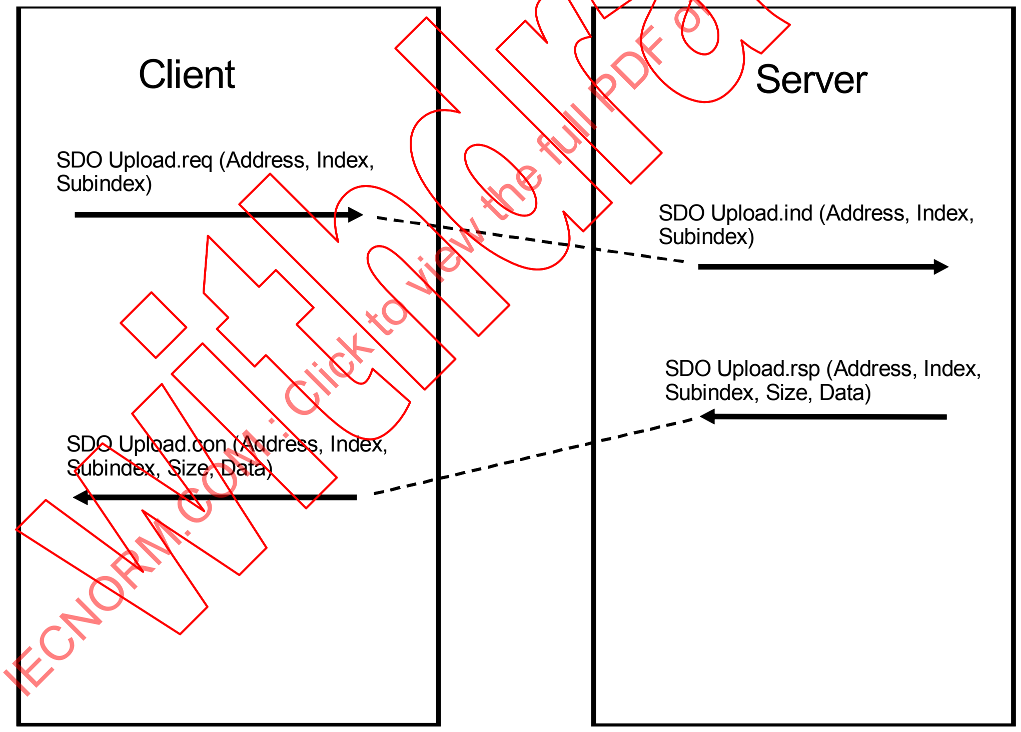

Client
Server
SDO Upload.req (Address, Index, Subindex)
SDO Upload.ind (Address, Index, Subindex)
SDO Upload.rsp (Address, Index, Subindex, Size, Data)
SDO Upload.con (Address, Index, Subindex, Size, Data)

Figure 44: Successful single SDO-Upload sequence

The following diagram shows the primitives between client and server in case of a unsuccessful SDO Upload sequence. The service parameter Address, Index and Subindex shall be the same in the Upload request and the Abort request.

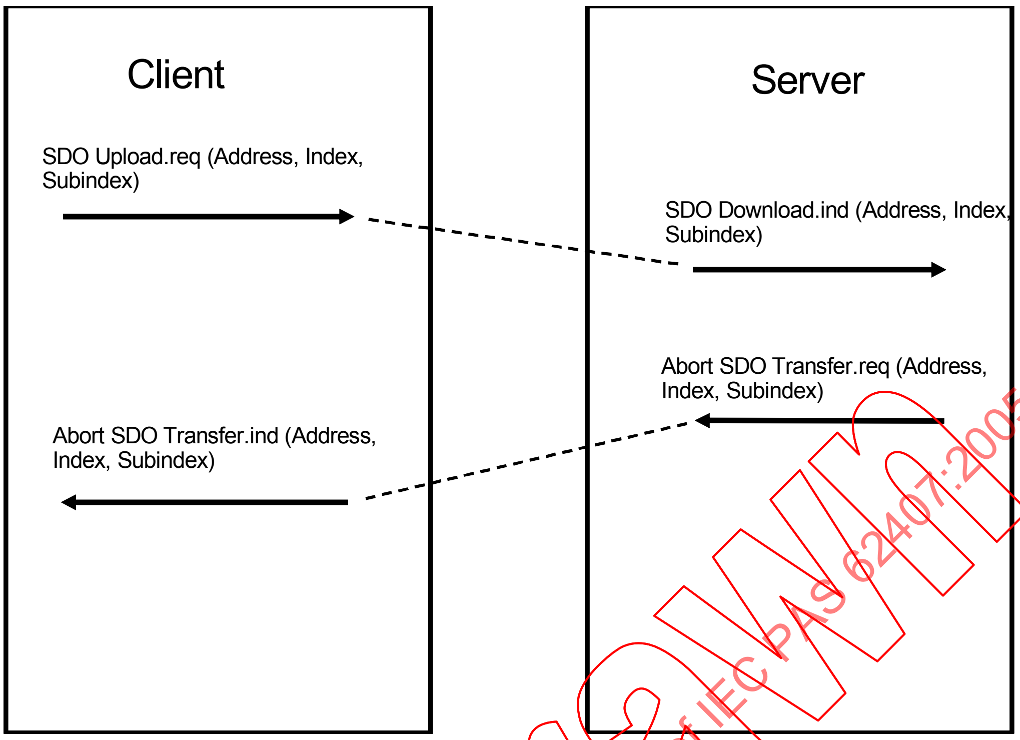

Client
SDO Upload.req (Address, Index, Subindex)
Abort SDO Transfer.ind (Address, Index, Subindex)
Server
SDO Download.ind (Address, Index, Subindex)
Abort SDO Transfer.req (Address, Index, Subindex)

Figure 45: Unsuccessful single SDO-Upload sequence

The following diagram shows the primitives between client and server in case of a successful segmented SDO Upload sequence. The service parameter Address, Index and Subindex shall be the same in the request and the response.

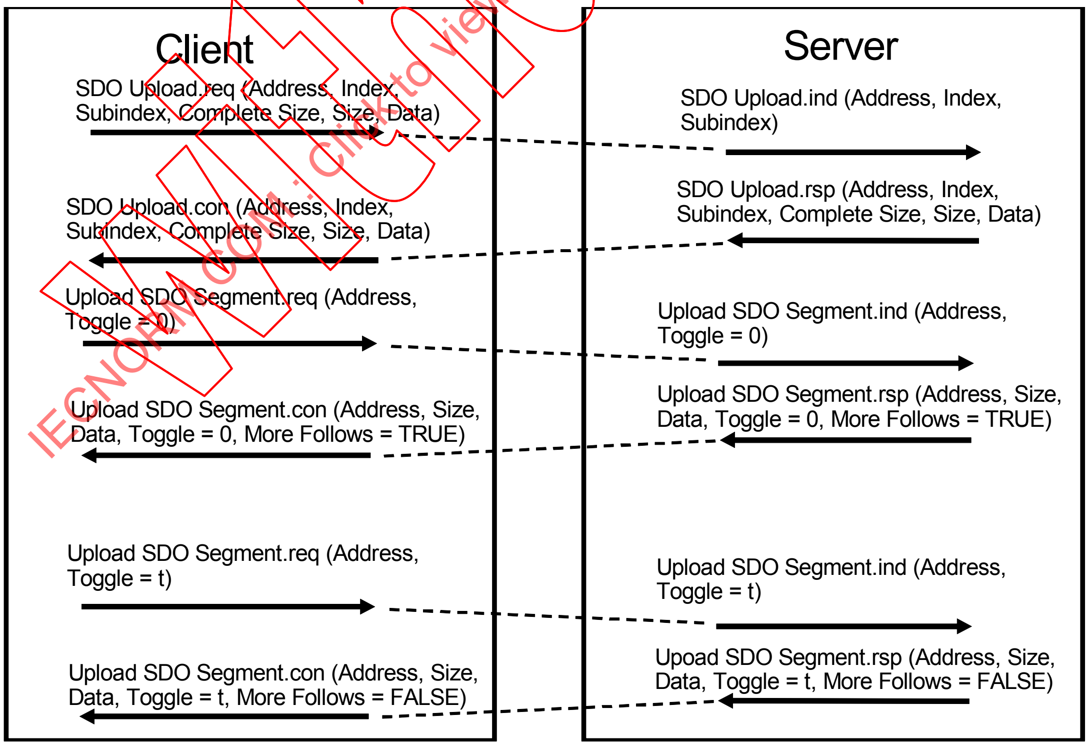

Client
SDO Upload.req (Address, Index, Subindex, Complete Size, Size, Data)
SDO Upload.con (Address, Index, Subindex, Complete Size, Size, Data)
Upload SDO Segment.req (Address, Toggle = 0)
Upload SDO Segment.con (Address, Size, Data, Toggle = 0, More Follows = TRUE)
Upload SDO Segment.req (Address, Toggle = t)
Upload SDO Segment.con (Address, Size, Data, Toggle = t, More Follows = FALSE)
Server
SDO Upload.ind (Address, Index, Subindex)
SDO Upload.rsp (Address, Index, Subindex, Complete Size, Size, Data)
Upload SDO Segment.ind (Address, Toggle = 0)
Upload SDO Segment.rsp (Address, Size, Data, Toggle = 0, More Follows = TRUE)
Upload SDO Segment.ind (Address, Toggle = t)
Upoad SDO Segment.rsp (Address, Size, Data, Toggle = t, More Follows = FALSE)

Figure 46: Successful segmented SDO-Upload sequence

9.2.7.3.3.1.2 Initiate SDO Upload Expedited

The Expedited SDO Upload service shall be used for an expedited transfer of up to 4 bytes of data from the server to client. The server shall answer with the result of the upload operation and the requested data in case of a successful operation.

Table 65 – Initiate SDO Upload Expedited

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Subindex	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Index		Mandatory
Subindex		Mandatory
Size		Mandatory
Data		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This parameter contains the index in the object dictionary of the object which should be uploaded.

Subindex

This parameter contains the subindex in the object dictionary of the object which should be uploaded.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Expedited SDO Upload service was successful. In case of an error an Abort SDO Transfer service will be send from the server.

Address

This parameter shall be the same as the parameter Address in the request.

Index

This parameter shall be the same as the parameter Index in the request.

Subindex

This parameter shall be the same as the parameter Subindex in the request.

Size

This parameter indicates the number of bytes in the Data parameter.

Data

This parameter contains the requested data.

9.2.7.3.3.1.3 Initiate SDO Upload Normal

The Initiate SDO Upload service shall be used for a single normal transfer of data from the server to the client, if the number of bytes to be uploaded fit in the mailbox, or to start a segmented transfer with more bytes. The server shall answer with the result of the upload operation and the requested data in case of a successful operation.

Table 66 – Initiate SDO Upload Normal

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Subindex	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Index		Mandatory
Subindex		Mandatory
Complete Size		Mandatory
Size		Mandatory
Data		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This parameter contains the index in the object dictionary of the object which should be uploaded.

Subindex

This parameter contains the subindex in the object dictionary of the object which should be uploaded.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Expedited SDO Upload service was successful. In case of an error an Abort SDO Transfer service will be send from the server.

Address

This parameter shall be the same as the parameter Address in the request.

Index

This parameter shall be the same as the parameter Index in the request.

Subindex

This parameter shall be the same as the parameter Subindex in the request.

Complete Size

This parameter indicates the number of bytes to be uploaded from the server. If the Complete Size is bigger than the Size parameter of the response the difference number of bytes will be send in the next Upload SDO Segment services from the server to the client.

Size

This parameter indicates the number of bytes in the Data parameter.

Data

This parameter contains the requested data.

9.2.7.3.3.1.4 Upload SDO Segment

The Upload SDO Segment service shall be used to upload the additional data from the server to the client, which did not fit in the mailbox with the Initiate SDO Upload service response. The client will start so many Upload SDO Segment services until all data is uploaded from the server.

Table 67 – Upload SDO Segment

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Toggle	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Toggle		Mandatory
More Follows		Mandatory
Size		Mandatory
Data		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Toggle

This parameter shall be toggled with every Download SDO Segmenet service, starting by zero.

More Follows

This parameter indicates, if this service is the last Download SDO Segment service to download the data for the object or if at least one Download SDO Segment service is following.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the data to be downloaded.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Initiate SDO Download service was successful. In case of an error an Abort SDO Transfer service will be send from the server.

Address

This parameter shall be the same as the parameter Address in the request.

Toggle

This parameter shall be the same as the parameter Toggle in the request.

More Follows

This parameter indicates, if this service shall be the last Upload SDO Segment service to download the data for the object or if at least one Upload SDO Segment service shall be following.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the data to be downloaded.

9.2.7.3.4 Abort SDO Transfer

The Abort SDO Transfer service shall be used from the server instead of a service response to a download or upload service in case of an error.

Table 68 – Abort SDO Transfer

Parameter	**Request/Indication**	**Response/Confirmation**
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Subindex	Mandatory	
Reason	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This parameter contains the parameter Index of the related SDO Download or SDO Upload service.

Subindex

This parameter contains the parameter Subindex of the related SDO Download or SDO Upload service.

Reason

This parameter contains the abort reason.

9.2.7.4 SDO Information Services

9.2.7.4.1 Overview

With the SDO Information services the object dictionary of a server can be read by a client. The primitives of the SDO Information services are mapped to the primitives of the Mailbox Transmission services:

Slave is Server:

Get OD List.req/.ind:	Mailbox Write.req/.ind
Get OD List.rsp/.con:	Mailbox Read.req/.ind
Get Object Description.req/.ind:	Mailbox Write.req/.ind
Get Object Description.rsp/.con:	Mailbox Read.req/.ind
Get Entry Description.req/.ind:	Mailbox Write.req/.ind
Get Entry Description.rsp/.con:	Mailbox Read.req/.ind

Slave is Client:

Get OD List.req/.ind:	Mailbox Read.req/.ind
Get OD List.rsp/.con:	Mailbox Write.req/.ind
Get Object Description.req/.ind:	Mailbox Read.req/.ind
Get Object Description.rsp/.con:	Mailbox Write.req/.ind
Get Entry Description.req/.ind:	Mailbox Read.req/.ind
Get Entry Description.rsp/.con:	Mailbox Write.req/.ind

The following diagram shows the primitives between client and server which is the same for all OD Information services:

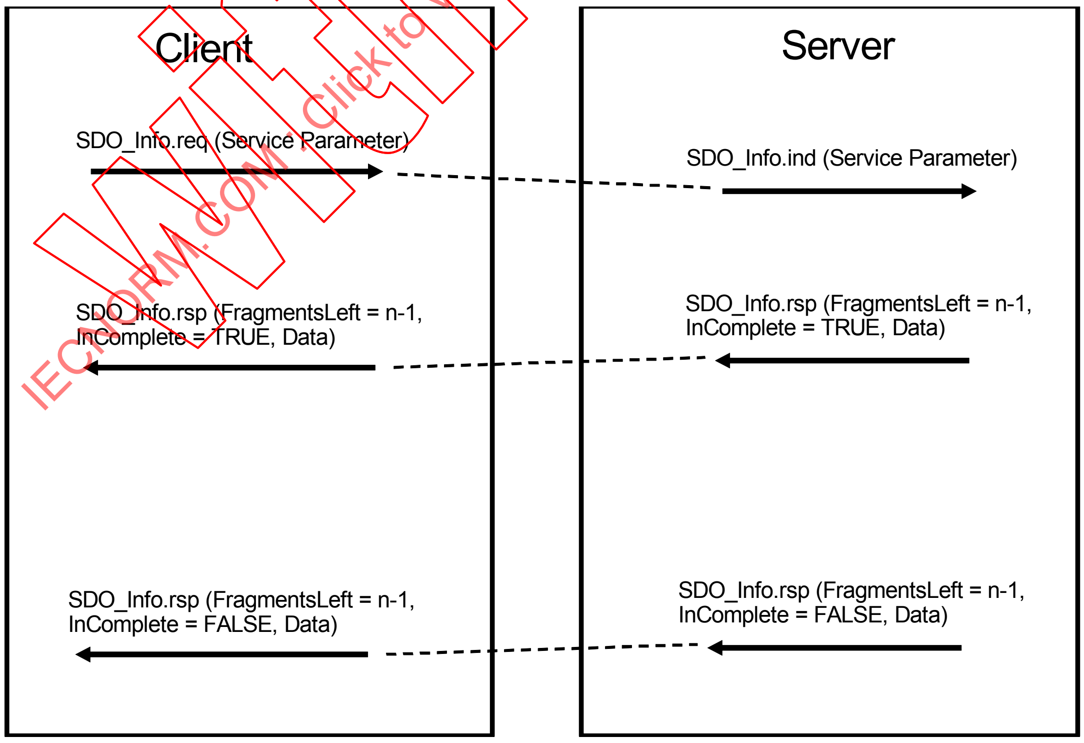

Client
Server
SDO_Info.req (Service Parameter)
SDO_Info.ind (Service Parameter)
SDO_Info.rsp (FragmentsLeft = n-1, InComplete = TRUE, Data)
SDO_Info.rsp (FragmentsLeft = n-1, InComplete = TRUE, Data)
SDO_Info.rsp (FragmentsLeft = n-1, InComplete = FALSE, Data)
SDO_Info.rsp (FragmentsLeft = n-1, InComplete = FALSE, Data)

Figure 47: SDO Information service sequence

9.2.7.4.2 SDO Information Service

Table 69 – SDO Information Service

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Opcode	Mandatory	
Size	Mandatory	
Data	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Opcode		Mandatory
Incomplete		Mandatory
Fragments Left		Mandatory
Size		Mandatory
Data		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Opcode

This parameter describes the requested SDO Information Service.

Size

This parameter contains the size of the parameter Data.

Data

This parameter depends on the requested SDO Information Service.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Get OD List service was successful.

Address

This parameter shall be the same as the parameter Address in the request.

Opcode

This parameter describes the requested SDO Information Service.

Incomplete

This parameter contains the information if fragments will follow to get the complete response.

Fragments Left

This parameter contains the information how much fragments will follow to get the complete response.

Size

This parameter contains the size of the parameter Data.

Data

This parameter depends on the requested SDO Information Service.

9.2.7.4.3 Get OD List

With the Get OD List service lists of objects existing in the object dictionary of the server can be read. If the response does not fit in the mailbox the response shall be sent with multiple fragments as described above.

Table 70 – Get OD List

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
List Type	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
List Type		Mandatory
Size		Mandatory
Index List		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

List Type

This contains the type of the list to be read.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Get OD List service was successful.

Address

This parameter shall be the same as the parameter Address in the request.

List Type

This parameter shall be the same as the parameter ListType in the request.

Size

This parameter contains the size of the parameter IndexList.

Index List

This parameter contains the read index list.

9.2.7.4.4 Get Object Description

With the Get Object Description service an object description existing in the object dictionary of the server can be read. If the response does not fit in the mailbox the response shall be sent with multiple fragments as described above.

Table 71 – Get Object Description

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Index		Mandatory
Data Type		Mandatory
Object Code		Mandatory
Object Category		Mandatory
Name		Mandatory

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This contains the index of the object description to be read.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Get Object Description service was successful.

Address

This parameter shall be the same as the parameter Address in the request.

Index

This parameter shall be the same as the parameter Index in the request.

Object Code

This parameter contains the object code of the requested object.

Object Category

This parameter contains the object category of the requested object.

Name

This parameter contains the name of the requested object.

9.2.7.4.5 Get Entry Description

With the Get Entry Description service an entry description existing in the object dictionary and addressed by index and subindex of the server can be read. If the response does not fit in the mailbox the response shall be sent with multiple fragments as described above.

Table 72 – Get Entry Description

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Index	Mandatory	
Subindex	Mandatory	
Request Default Value	Mandatory	
Request Min Value	Mandatory	
Request Max Value	Mandatory	
Result		Mandatory
Success		Mandatory
Address		Mandatory
Index		Mandatory
Subindex		Mandatory
Response Default Value		Mandatory
Response Min Value		Mandatory
Response Max Value		Mandatory
Data Type		Mandatory
Unit Type		Mandatory
Bit Length		Mandatory
Object Category		Mandatory
Read Access		Mandatory
Write Access		Mandatory
Constant Value		Mandatory
Write Access PreOp		Mandatory
Write Access SafeOp		Mandatory
Write Access Op		Mandatory
PDO Mapping		Mandatory
Default PDO Mapping		Mandatory
Default Value		Optional
Min Value		Optional
Max Value		Optional
Description		Optional

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Index

This contains the index of the entry description to be read.

Subindex

This contains the subindex of the entry description to be read.

Request Default Value

This contains the information, if the parameter Default Value is requested to be included in the response.

Request Min Value

This contains the information, if the parameter Min Value is requested to be included in the response.

Request Max Value

This contains the information, if the parameter Max Value is requested to be included in the response.

Result

The result shall convey the service specific parameters of the service response.

Success

This parameter indicates that the Get Object Description service was successful.

Address

This parameter shall be the same as the parameter Address in the request.

Index

This parameter shall be the same as the parameter Index in the request.

Subindex

This parameter shall be the same as the parameter Subindex in the request.

Response Default Value

This contains the information, if the parameter Default Value is included in the response.

Response Min Value

This contains the information, if the parameter Min Value is included in the response.

Response Max Value

This contains the information, if the parameter Max Value is included in the response.

Data Type

This parameter contains the data type of the object.

Unit Type

This parameter contains the unit type of the object.

Bit Length

This parameter contains the bit length of the object.

Object Category

This parameter contains the object category of the object.

Read Access

This parameter contains the information if the object is readable.

Write Access

This parameter contains the information if the object is writable.

Constant Value

This parameter contains the information if the object has a constant value.

Write Access PreOp

This parameter contains the information if the object is writable in the Pre-Operational state.

Write Access SafeOp

This parameter contains the information if the object is writable in the Safe-Operational state.

Write Access Op

This parameter contains the information if the object is writable in the Operational state.

PDO Mapping

This parameter contains the information if the object is mappable in a PDO.

Default PDO Mapping

This parameter contains the information if the object is included in the Default PDO Mapping.

Default Value

This parameter contains the default value of the object.

Min Value

This parameter contains the minimum value of the object.

Max Value

This parameter contains the maximum value of the object.

Description

This parameter contains the description of the object.

9.2.7.5 Process Data

9.2.7.5.1 Overview

The Process Data of an EtherCAT slave consists of Sync Manager Channel objects. Each Sync Manager Channel object describes a consistent area inside the EtherCAT Process Data and consists of several Process Data objects.

Process Data Objects (PDO) consist of objects in the object dictionary which are PDO mappable. The PDO Mapping objects describes how these objects are related to a PDO.

Every EtherCAT slave with an application controller shall support the reading of the PDO Mapping objects.

PDO mapping refers to mapping of the application objects (real time process data) from the object dictionary to the PDOs. The device profiles provide a default mapping for every device type, and this is appropriate for most applications. E.g. the default mapping for digital I/O simply represents the inputs and outputs in their physical sequence in the transmit and receive PDOs.

The current mapping can be read by means of corresponding entries in the object directory. These are known as the mapping tables. The first location in the mapping table (sub-index 0) contains the number of mapped objects that are listed after it. The tables are located in the object directory at index 0x1600ff for the RxPDOs and at 0x1A00ff for the TxPDOs.

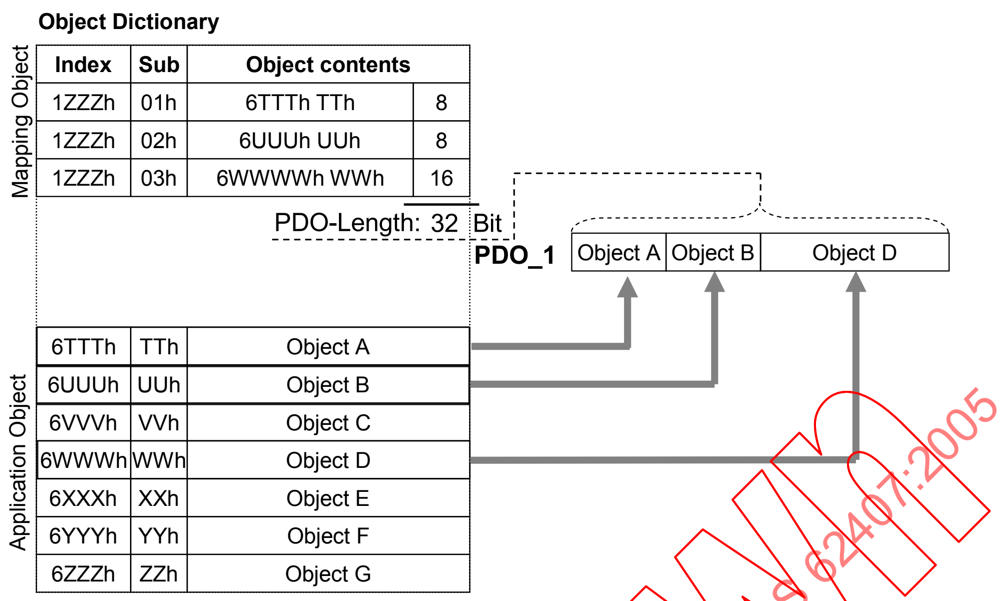


Figure 48: PDO Mapping Example

Sync Manager Channel Objects (SMCO) consist of several PDOs. The Sync Manager PDO Assign objects describes how these PDOs are related to a SMCO.

Every EtherCAT slave with an application controller shall support the reading of the Sync Manager PDO Assign objects.

Sync Manager PDO Assign refers to the assignment of the PDOs to the SMCOs. For devices with a configurable mapping or devices allowing an access to the process data by different master tasks the Sync Manager PDO Assign objects shall be writable.

The current assignment can be read by means of corresponding entries in the object directory. The first location in the Sync Manager PDO Assignment table (sub-index 0) contains the number of assigned PDOs that are listed after it. The tables are located in the object directory at index 0x1C10ff.

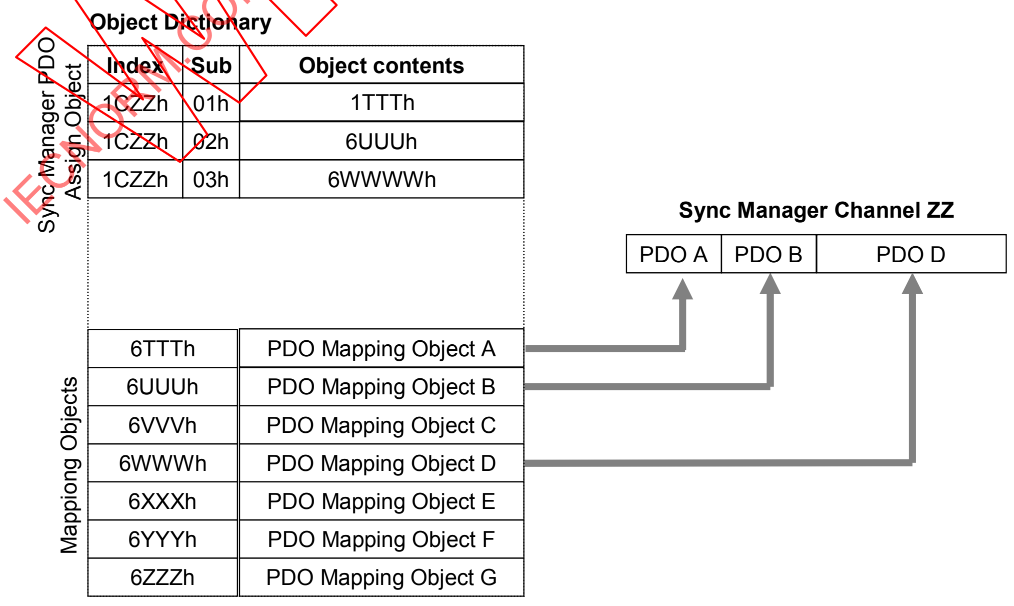

Object Dictionary
Sync Manager PDO Assign Object
Index | Sub | Object contents
1CZZh | 01h | 1TTTh
1CZZh | 02h | 6UUUh
1CZZh | 03h | 6WWWWh
Sync Manager Channel ZZ
PDO A | PDO B | PDO D
Mappiong Objects
6TTTh | PDO Mapping Object A
6UUUh | PDO Mapping Object B
6VVVh | PDO Mapping Object C
6WWWh | PDO Mapping Object D
6XXXh | PDO Mapping Object E
6YYYh | PDO Mapping Object F
6ZZZh | PDO Mapping Object G

Figure 49: Sync Manager PDO Assigment

Every EtherCAT slave with an application controller shall support the reading of the PDO Mapping and the Sync Manager PDO Assign objects. For slaves with a configurable mapping there are two possibilities how this configurable mapping shall be supported. In the first possibility the slave supports different mappings described in the PDO Mapping objects which are only readable. The master configures the actual mapping by selecting the desired PDO Mapping objects when writing the Sync Manager PDO Assign objects. The second possibility for more complex devices allows the master to write the PDO Mapping objects to get a more flexible mapping configuration possibility.

9.2.7.5.2 Process Data Objects (PDO)

Each Process Data object has the following parameters.

Parameter

Number

This parameter contains the number to identify the PDO.

Direction

This parameter contains the direction Rx (master to slave, client to server) or Tx (slave to master, server to client) of the PDO.

Number of Mapped Objects

This parameter contains the number of mapped objects which are related to the PDO.

List of mapped Objects

This parameter contains the related objects which define the mapping of the PDO.

9.2.7.5.3 Sync Manager Channel Objects (SMCO)

Each Sync Manager Channel object has the following parameters.

Parameter

Number

This parameter contains the number to identify the Sync Manager channel.

Direction

This parameter contains the direction Output (master to slave) or Input (slave to master) of the Sync Manager channel.

Number of Assigned PDOs

This parameter contains the number of assigned PDOs which are related to the Sync Manager channel.

List of assigned PDOs

This parameter contains the assigned PDOs to the Sync Manager channel.

9.2.7.5.4 PDO transmission via Mailbox

With the PDO services Process Data Objects can be transmit via the Mailbox Interface acyclicly from the client to the server (RxPDO) or from the server to the client (TxPDO)

The primitives of the PDO services are mapped to the primitives of the Mailbox Transmission services:

RxPDO.req:	Mailbox Write.req
RxPDO.ind:	Mailbox Write.ind
TxPDO.req:	Mailbox Read.req
TxPDO.ind:	Mailbox Read.ind
RxPDO_RT.req:	Mailbox Read.req

RxPDO_RT.ind:	Mailbox Read.ind
TxPDO_RT.req:	Mailbox Write.req
TxPDO_RT.ind:	Mailbox Write.ind

9.2.7.5.4.1 RxPDO

The following diagram shows the primitives between client and server in case of a RxPDO service.

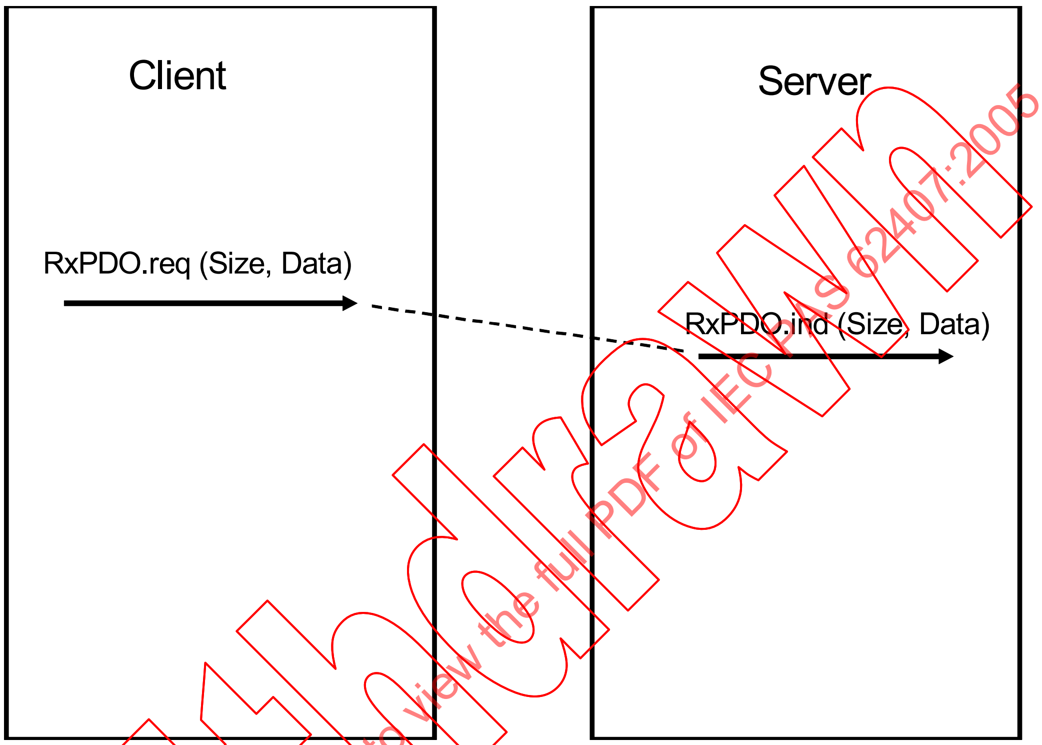


Figure 50: RxPDO service

The RxPDO service shall be used from the client to transmit output data to the server. The mapping and the size of the data is defined in the PDO mapping objects

Table 73 – RxPDO Service

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Number	Mandatory	
Size	Mandatory	
Data	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the server.

Number

This parameter contains the PDO number which is related to the PDO mapping objects.

Size

This parameter contains the size of the Data parameter.

Data

This parameter contains the data of the PDO.

9.2.7.5.4.2 TxPDO

The following diagram shows the primitives between server and client in case of a TxPDO service.

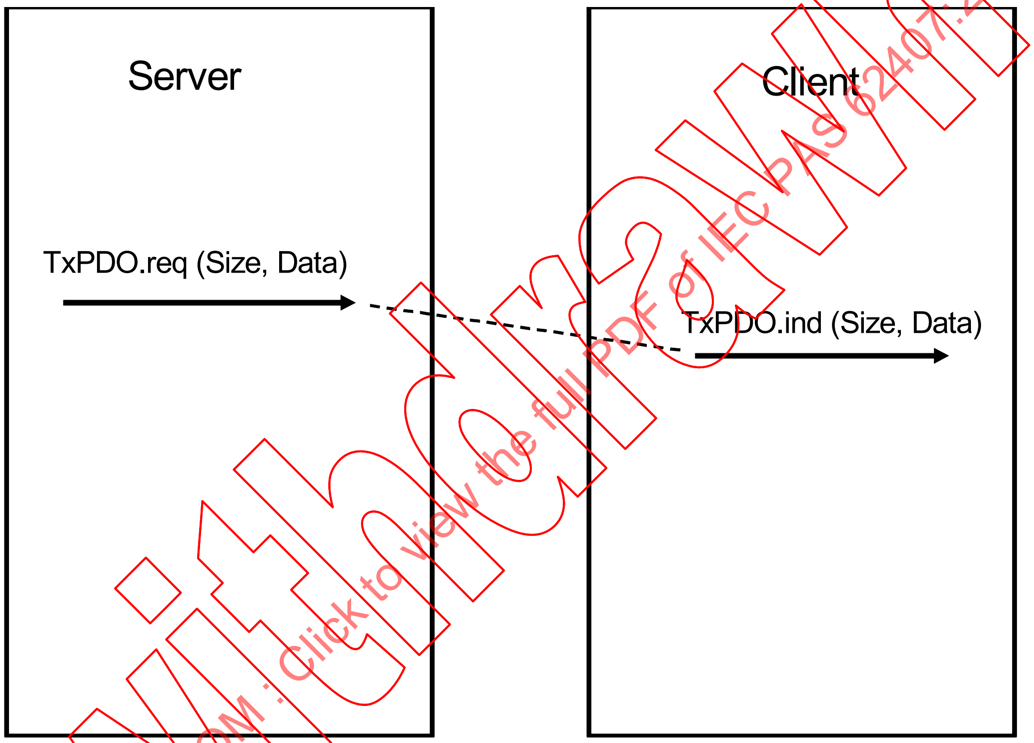


Figure 51: TxPDO service

The TxPDO service shall be used from the server to transmit input data to the client. The mapping and the size of the data is defined in the PDO mapping objects

Table 74 – TxPDO Service

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Number	Mandatory	
Size	Mandatory	
Data	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the server.

Number

This parameter contains the PDO number which is related to the PDO mapping objects.

Size

This parameter contains the size of the Data parameter.

Data

This parameter contains the data of the PDO.

9.2.7.5.4.3 RxPDO Remote Transmission Request

The following diagram shows the primitives between client and server in case of a RxPDO Remote Transmission Request service.

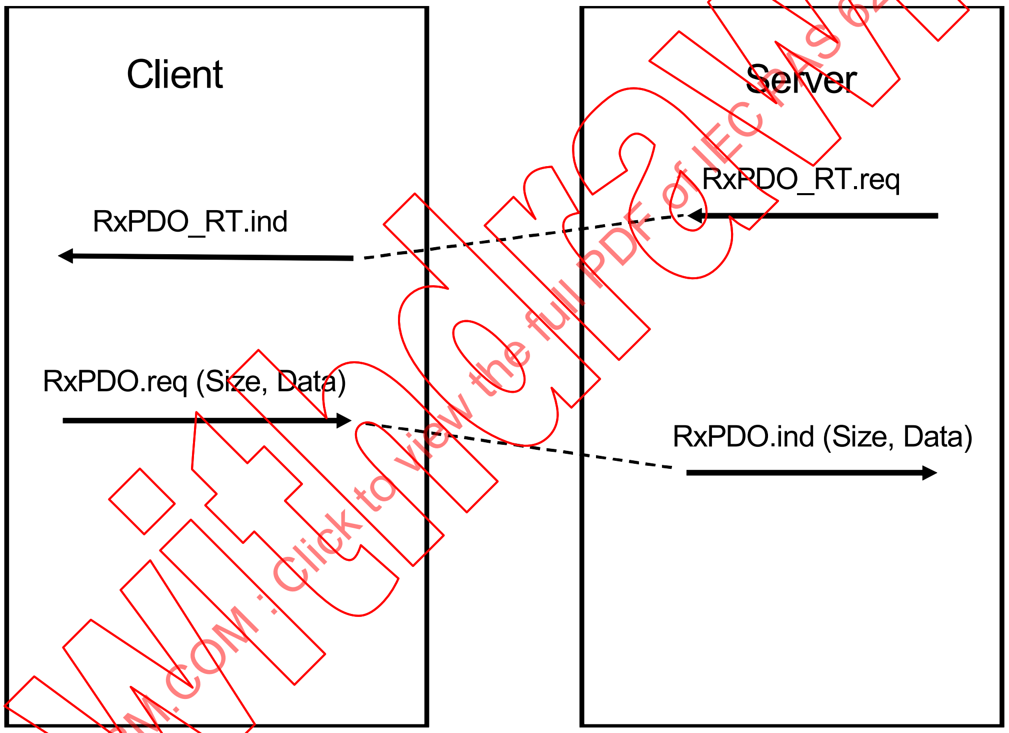


Figure 52: RxPDO Remote Transmission Request service sequence

The RxPDO_RT service shall be used from the server to request output data from the client.

Table 75 – RxPDO Remote Transmission Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Number	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the destination.

Number

This parameter contains the PDO number which is related to the PDO mapping objects.

9.2.7.5.4.4 TxPDO Remote Transmission Request

The following diagram shows the primitives between client and server in case of a TxPDO Remote Transmission Request service.

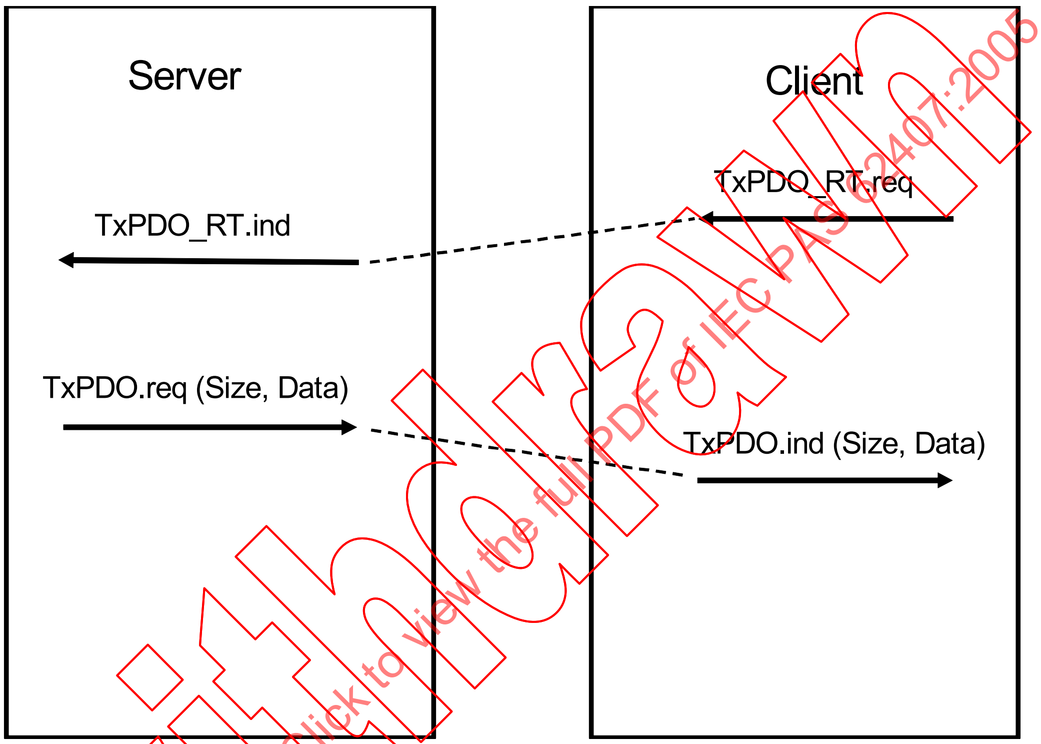


Figure 53: TxPDO Remote Transmission Request service sequence

The TxPDO_RT service shall be used from the client to request input data from the server.

Table 76 – TxPDO Remote Transmission Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Number	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the destination.

Number

This parameter contains the PDO number which is related to the PDO mapping objects.

9.2.7.6 Command

Command objects can be used for operations in a server which need some time before a response is available or for operations where data has to be sent in the request and the response.

An operation will be started in the server by a writing subindex 1 of the command object with the command request data by a SDO Download service. The status and the response data of the operation will be read with a SDO Upload to subindex 3 of the command object.

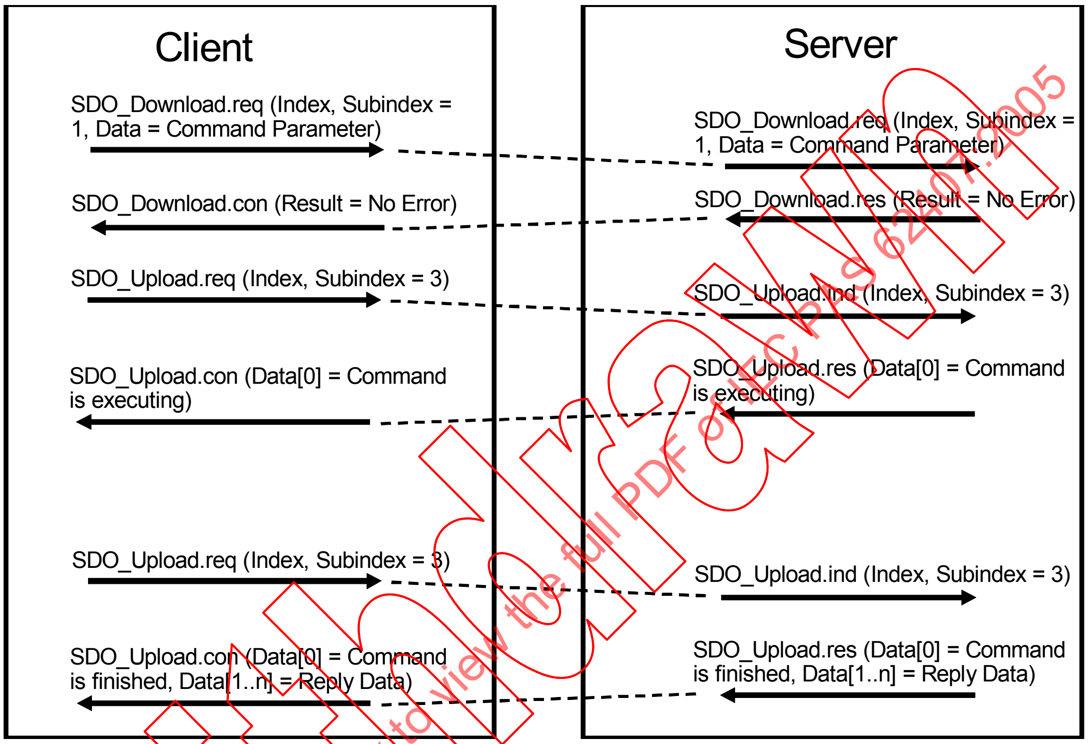

Client
Server
SDO_Download.req (Index, Subindex = 1, Data = Command Parameter)
SDO_Download.ind (Index, Subindex = 1, Data = Command Parameter)
SDO_Download.con (Result = No Error)
SDO_Download.res (Result = No Error)
SDO_Upload.req (Index, Subindex = 3)
SDO_Upload.ind (Index, Subindex = 3)
SDO_Upload.con (Data[0] = Command is executing)
SDO_Upload.res (Data[0] = Command is executing)
SDO_Upload.req (Index, Subindex = 3)
SDO_Upload.ind (Index, Subindex = 3)
SDO_Upload.con (Data[0] = Command is finished, Data[1..n] = Reply Data)
SDO_Upload.res (Data[0] = Command is finished, Data[1..n] = Reply Data)

Figure 54: Command sequence

9.2.7.7 Emergency

Emergency messages are triggered by the occurrence of a device internal error situation. The transmission is executed via the mailbox interface.

The primitives of the Emergency service are mapped to the primitives of the Mailbox Transmission services:

Emergency.req:	Mailbox Read.req
Emergency.ind:	Mailbox Read.ind

The following diagram shows the primitives between server and client in case of an Emergency service.

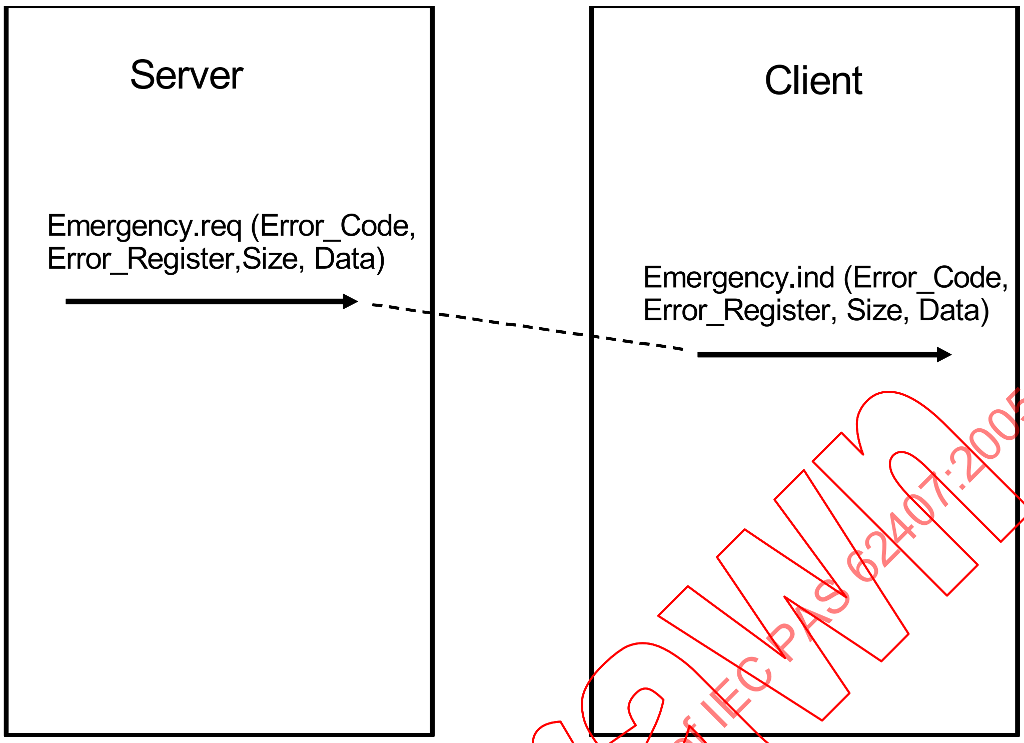

Server
Client
Emergency.req (Error_Code, Error_Register,Size, Data)
Emergency.ind (Error_Code, Error_Register, Size, Data)

Figure 55: Emergency service

The Emergency service shall be used from the server to transmit diagnostic messages to the client. Each diagnostic event which was transmitted by the server to the client shall be transmitted again with the reset error code when the diagnostic event disappears.

Table 77 – Emergency Service

Parameter	**Request/Indication**	**Response/Confirmation**
Argument	Mandatory	
Address	Mandatory	
Error Code	Mandatory	
Error Register	Mandatory	
Size	Mandatory	
Data	Optional	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the server.

Error Code

This parameter contains the emergency error code.

Error Register

This parameter contains the emergency error register.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the manufacturer specific diagnostic data to be transmitted.

9.2.8 Ethernet over EtherCAT (EoE)

9.2.8.1 Overview

In addition to the already described EtherCAT addressing mode for the communication with the EtherCAT devices, an Ethernet fieldbus is also expected to feature standard IP-based protocols such as TCP/IP, UDP/IP and all higher protocols based on these (HTTP, FTP, SNMP etc.). Ideally, individual Ethernet frames should be transferred transparently, since this avoids restrictions with regard to the protocols to be transferred.

There are two different basic approaches for transferring acyclic Ethernet telegrams in cyclic fieldbus mode. In the first variant, an appropriate time slice is allocated, in which the acyclic Ethernet frames can be embedded. This time slice has to be chosen sufficiently large to be able to accommodate complete Ethernet telegrams. The common MTU (Maximum Transmission Unit) is 1514 bytes, corresponding to approximately 125 µs at 100 MBaud. The minimum resulting cycle time for systems using this variant is approximately 200-250 µs. A reduction of the MTU often leads to problems: While the IP protocol allows fragmentation in principle, this is often not recommended, and it will no longer be available in the next generation (IPv6). Data consistency problems may also occur, particularly in UDP/IP transfers.

EtherCAT utilizes the second variant, in which the Ethernet telegrams are tunnelled and re-assembled in the associated device, before being relayed as complete Ethernet telegrams. This procedure does not restrict the achievable cycle time, since the fragments can be optimized according to the available bandwidth (EtherCAT instead of IP fragmentation). In this case, EtherCAT defines a standard channel, which in principle enables any EtherCAT device to participate in the normal Ethernet traffic. In an intelligent drive controller that exchanges process data with a cycle time of 100 µs, for example, an HTTP server can be integrated that features its own diagnostics interface in the form of web pages.

Another application for transferred Ethernet telegrams will be switchport devices. They offer standard Ethernet ports at any location within the EtherCAT system, through which any Ethernet device may be connected. For example, this may be a service computer that communicates directly with the control and queries the web page of an intelligent EtherCAT device, or simply routes it to the intranet or internet via the control. The EtherCAT master also features software-integrated switch functionality, which is responsible for the routing of the individual Ethernet frames from and to the EtherCAT devices and the IP stack of the host operating system. The switch functionality is identical with that of a standard layer 2 switch and responds to the Ethernet addresses used irrespective of the protocol

9.2.8.2 EoE Services

The primitives of the EoE services are mapped to the primitives of the Mailbox Transmission services:

Slave is Server:

Initiate EoE.req/.ind:	Mailbox Write.req/.ind
Initiate EoE.rsp/.con:	Mailbox Read.req/.ind
EoE Fragment.req/.ind:	Mailbox Write.req/.ind
EoE Fragment.rsp/.con:	Mailbox Read.req/.ind

Slave is Client:

Initiate EoE.req/.ind:	Mailbox Read.req/.ind

Initiate EoE.rsp/.con:	Mailbox Write.req/.ind
EoE Fragment.req/.ind:	Mailbox Read.req/.ind
EoE Fragment.rsp/.con:	Mailbox Write.req/.ind

The following diagram shows the primitives between client and server in case of a successful Ethernet over EtherCAT sequence. The Frame Number has to be the same for all Ethernet fragments in an EoE sequence

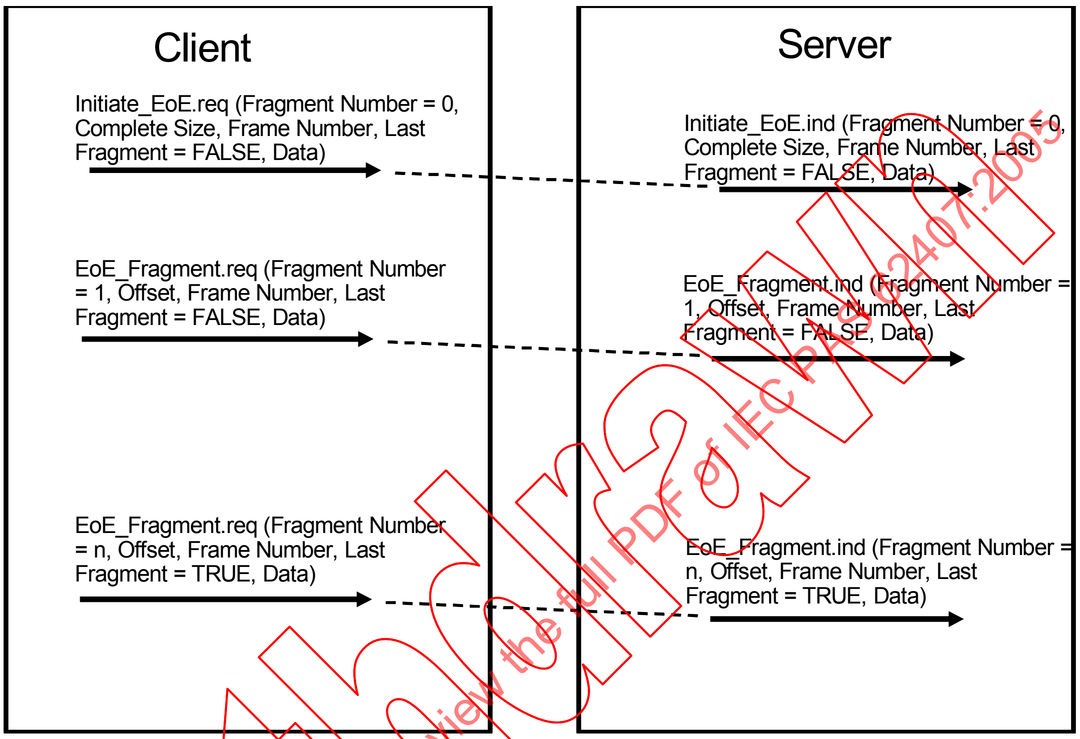

Client
Server
Initiate_EoE.req (Fragment Number = 0, Complete Size, Frame Number, Last Fragment = FALSE, Data)
Initiate_EoE.ind (Fragment Number = 0, Complete Size, Frame Number, Last Fragment = FALSE, Data)
EoE_Fragment.req (Fragment Number = 1, Offset, Frame Number, Last Fragment = FALSE, Data)
EoE_Fragment.ind (Fragment Number = 1, Offset, Frame Number, Last Fragment = FALSE, Data)
EoE_Fragment.req (Fragment Number = n, Offset, Frame Number, Last Fragment = TRUE, Data)
EoE_Fragment.ind (Fragment Number = n, Offset, Frame Number, Last Fragment = TRUE, Data)

Figure 56: Ethernet over EtherCAT sequence

9.2.8.3 Initiate Ethernet over EtherCAT

The Initiate EoE service shall be used to transmit the first fragment of an Ethernet frame.

Table 78 – Initiate Ethernet over EtherCAT

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Frame Number	Mandatory	
Complete Size	Mandatory	
Last Fragment	Mandatory	
Size	Mandatory	
Data	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Frame Number

This parameter shall be incremented with every new Initiate EoE service.

Complete Size

The parameter Complete Size indicates the number of bytes of the complete Ethernet frame.

Last Fragment

This parameter contains the information, if the complete Ethernet frame is transmitted with this service or if additional EoE Fragment services shall be sent.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains the first data part of the Ethernet frame.

9.2.8.4 Ethernet over EtherCAT Fragment

The EoE Fragment service shall be used to transmit the fragments of an Ethernet frame following the Initiate EoE service.

Table 79 – Ethernet over EtherCAT Fragment

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Frame Number	Mandatory	
Offset	Mandatory	
Last Fragment	Mandatory	
Size	Mandatory	
Data	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Frame Number

This parameter shall be the same as the parameter Frame Number in the related Initiate EoE service.

Offset

The parameter Offset contains the offset inside the complete Ethernet frame where the Data of this service is related.

Last Fragment

This parameter contains the information, if the complete Ethernet frame is transmitted with this service or if additional EoE Fragment services shall be sent.

Size

This parameter indicates the number of used bytes in the Data parameter.

Data

This parameter contains a data part of the Ethernet frame.

9.2.9 File Access over EtherCAT

9.2.9.1 Overview

The file access mailbox command specifies a standard way to download a firmware or any other files from a client to a server or to upload a firmware or any other files from a server to a client. The protocol is similar to the TFTP protocol (trivial file transfer protocol). Both sides can initiate a read or write request via a read or write request command.

9.2.9.2 FoE Services

The primitives of the FoE services are mapped to the primitives of the Mailbox Transmission services:

Slave is Server, Read Request:

FoE Read.req/.ind:	Mailbox Write.req/.ind
FoE Data.req/.ind:	Mailbox Read.req/.ind
FoE Ack.req/.ind:	Mailbox Write.req/.ind
FoE Error.req/.ind:	Mailbox Read.req/.ind

Slave is Server, Write Request:

FoE Write.req/.ind:	Mailbox Write.req/.ind
FoE Data.req/.ind:	Mailbox Write.req/.ind
FoE Ack.req/.ind:	Mailbox Read.req/.ind
FoE Busy.req/.ind:	Mailbox Read.req/.ind
FoE Error.req/.ind:	Mailbox Read.req/.ind

Slave is Client, Read Request:

FoE Read.req/.ind:	Mailbox Read.req/.ind
FoE Data.req/.ind:	Mailbox Write.req/.ind
FoE Ack.req/.ind:	Mailbox Read.req/.ind
FoE Error.req/.ind:	Mailbox Write.req/.ind

Slave is Client, Write Request:

FoE Write.req/.ind:	Mailbox Read.req/.ind
FoE Data.req/.ind:	Mailbox Read.req/.ind
FoE Ack.req/.ind:	Mailbox Write.req/.ind
FoE Busy.req/.ind:	Mailbox Write.req/.ind
FoE Error.req/.ind:	Mailbox Write.req/.ind

9.2.9.3 FoE Read Sequence

9.2.9.3.1 FoE Read with Success

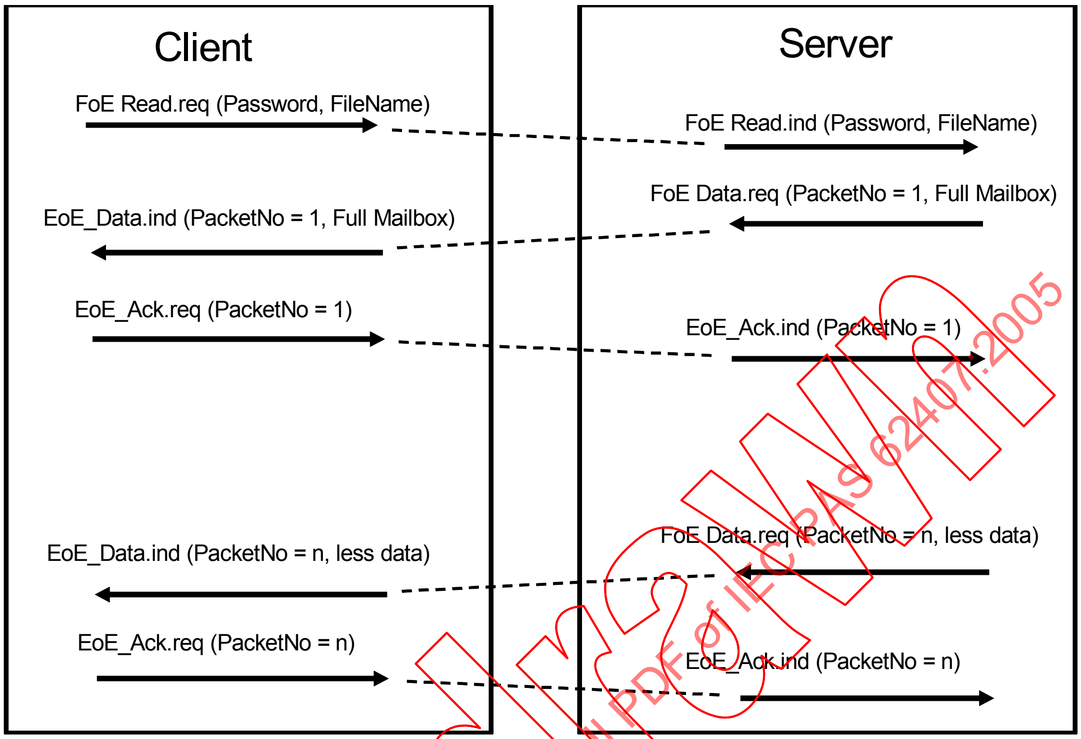

Client
Server
FoE Read.req (Password, FileName)
FoE Read.ind (Password, FileName)
FoE Data.req (PacketNo = 1, Full Mailbox)
EoE_Data.ind (PacketNo = 1, Full Mailbox)
EoE_Ack.req (PacketNo = 1)
EoE_Ack.ind (PacketNo = 1)
FoE Data.req (PacketNo = n, less data)
EoE_Data.ind (PacketNo = n, less data)
EoE_Ack.req (PacketNo = n)
EoE_Ack.ind (PacketNo = n)

Figure 57: FoE Read sequence with success

9.2.9.3.2 FoE Read with Error

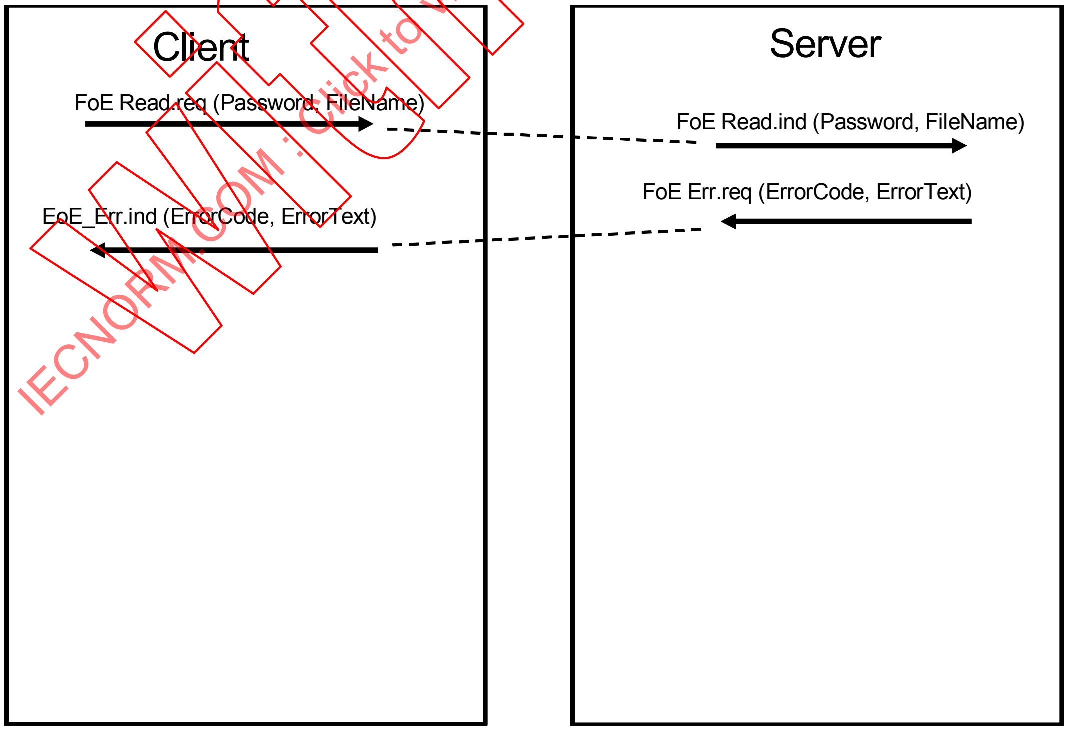

Client
Server
FoE Read.req (Password, FileName)
FoE Read.ind (Password, FileName)
FoE Err.req (ErrorCode, ErrorText)
EoE_Err.ind (ErrorCode, ErrorText)

Figure 58: FoE Read sequence with error

9.2.9.4 FoE Write Sequence

9.2.9.4.1 FoE Write with Success

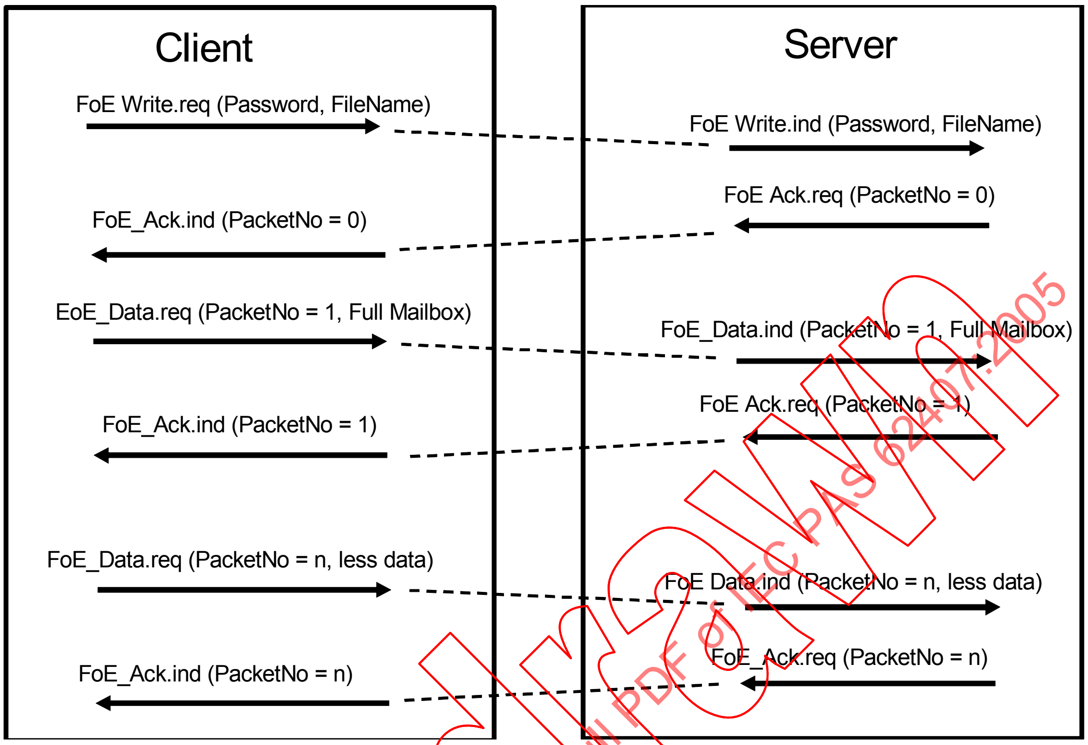

Client
FoE Write.req (Password, FileName)
FoE_Ack.ind (PacketNo = 0)
EoE_Data.req (PacketNo = 1, Full Mailbox)
FoE_Ack.ind (PacketNo = 1)
FoE_Data.req (PacketNo = n, less data)
FoE_Ack.ind (PacketNo = n)
Server
FoE Write.ind (Password, FileName)
FoE Ack.req (PacketNo = 0)
FoE_Data.ind (PacketNo = 1, Full Mailbox)
FoE Ack.req (PacketNo = 1)
FoE Data.ind (PacketNo = n, less data)
FoE_Ack.req (PacketNo = n)

Figure 59: FoE Write sequence with success

9.2.9.4.2 FoE Write with Error

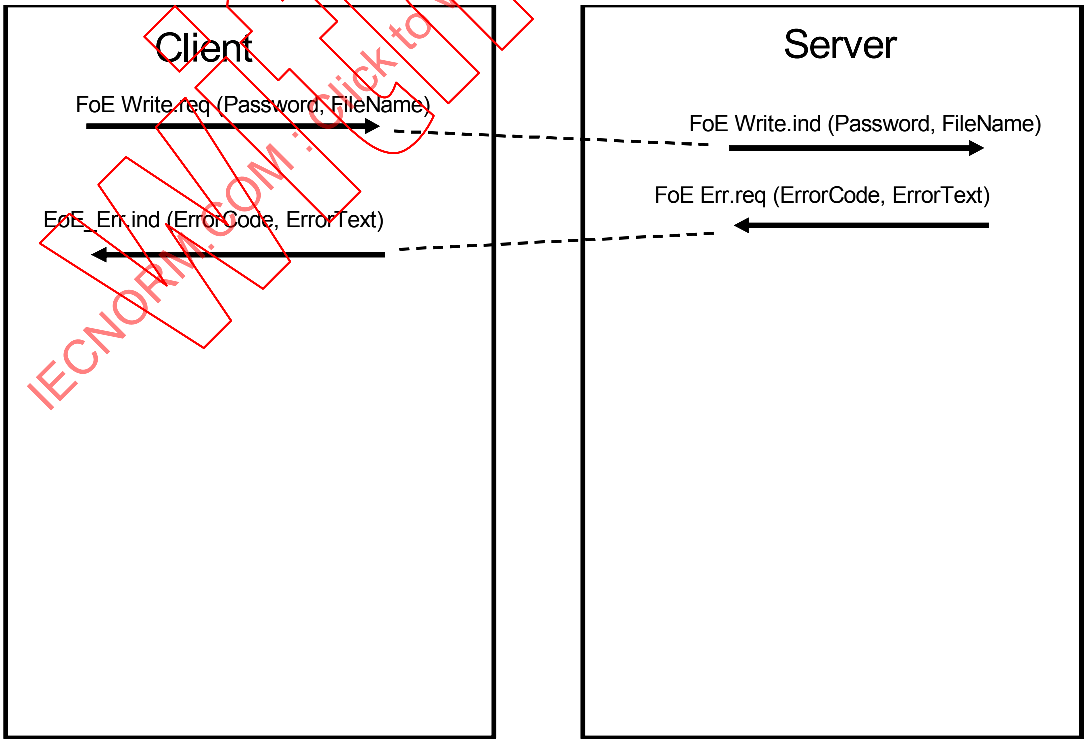

Client
FoE Write.req (Password, FileName)
EoE_Err.ind (ErrorCode, ErrorText)
Server
FoE Write.ind (Password, FileName)
FoE Err.req (ErrorCode, ErrorText)

Figure 60: FoE Read sequence with error

9.2.9.4.3 FoE Write with Busy

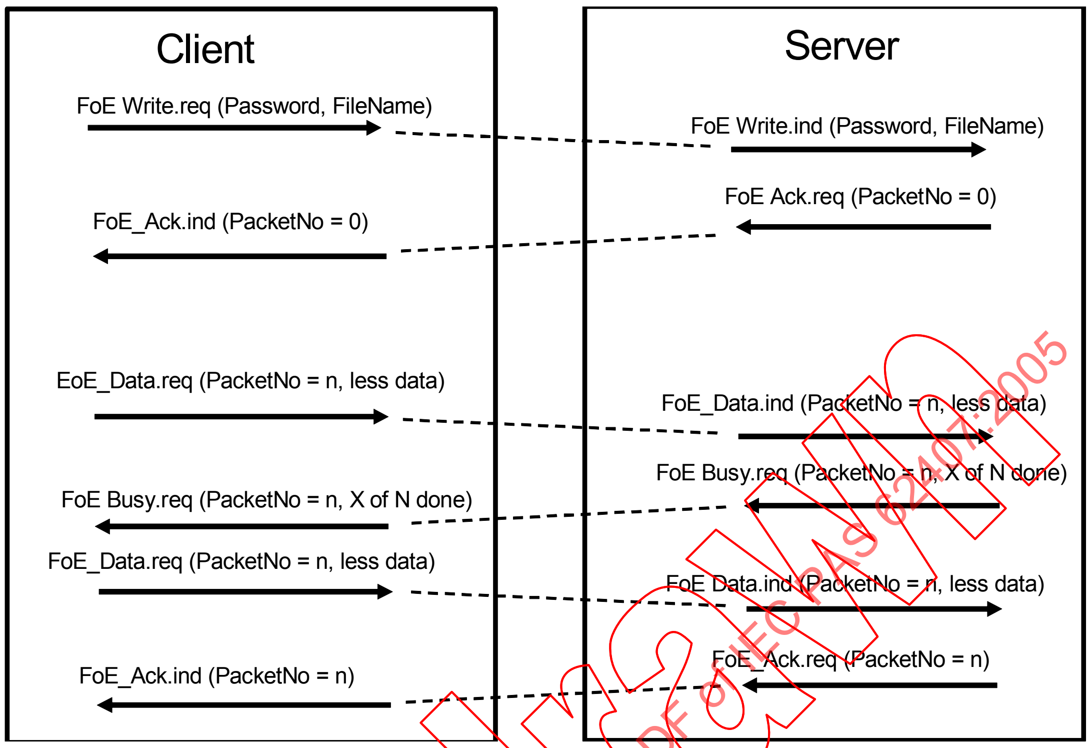

Client
Server
FoE Write.req (Password, FileName)
FoE Write.ind (Password, FileName)
FoE Ack.req (PacketNo = 0)
FoE_Ack.ind (PacketNo = 0)
EoE_Data.req (PacketNo = n, less data)
FoE_Data.ind (PacketNo = n, less data)
FoE Busy.req (PacketNo = n, X of N done)
FoE Busy.req (PacketNo = n, X of N done)
FoE_Data.req (PacketNo = n, less data)
FoE Data.ind (PacketNo = n, less data)
FoE_Ack.req (PacketNo = n)
FoE_Ack.ind (PacketNo = n)

Figure 61: FoE Write sequence with busy

9.2.9.5 FoE Read Request

The FoE Read Request service shall be used to read a file from a server.

Table 80 – FoE Read Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Password	Optional	
File Name	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Password

This parameter contains an optional password for the read access.

File Name

The parameter contains the file name of the file to be read.

9.2.9.6 FoE Write Request

The FoE Write Request service shall be used to write a file to a server.

Table 81 – FoE Write Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Password	Optional	
File Name	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Password

This parameter contains an optional password for the write access.

File Name

The parameter contains the file name of the file to be written.

9.2.9.7 FoE Data Request

The FoE Data Request service shall be used to transmit the file data from the server in case of a read request or from the client in case of a write request. The first Foe Data Request always starts with a Packet Number of one incrementing with every following Foe Data Request.

Table 82 – FoE Data Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Packet Number	Mandatory	
Size	Mandatory	
Data	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Packet Number

This parameter contains a number for flow control. It always starts with one incrementing with every following FoE Data Request.

Size

The parameter contains the size of the parameter Data.

Data

The parameter contains the data to be read or written.

9.2.9.8 FoE Ack Request

The FoE Ack Request service shall be used to acknowledge a FoE Data Request. The Packet Number will be always the same as send with the corresponding FoE Data Request before. A FoE Write Request will be acknowledged with a FoE Ack Request with a Packet Number of zero before the client starts with the first FoE Data Request.

Table 83 – FoE Ack Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Packet Number	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Packet Number

This parameter contains a number for flow control. It contains the sent packet number of the corresponding FoE Data Request or zero if a FoE Write Request will be acknowledged.

9.2.9.9 FoE Busy Request

The FoE Busy Request service shall be used to indicate the client, that the server is busy to store the written file data. The client will repeat the corresponding FoE Data Request until the server answers with a FoE Ack Request.

Table 84 – FoE Busy Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Done	Mandatory	
Entire	Mandatory	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Done

This parameter contains an information how much of the storing is finished.

Entire

This parameter contains an information how much of the storing is finished.

9.2.9.10 FoE Error Request

The FoE Error Request service shall be used to indicate the client that an error has happened during file operation.

Table 85 – FoE Error Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Error Code	Mandatory	
Error Text	Optional	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Error Code

This parameter contains an information about the error during file operation.

Error Text

This parameter contains an optional description of the error reason.

9.2.9.11 FoE Busy Request

The FoE Busy Request service shall be used to indicate the client, that the server is busy to store the written file data. The client will repeat the corresponding FoE Data Request until the server answers with a FoE Ack Request.

Table 86 – FoE Busy Request

Parameter	Request/Indication	Response/Confirmation
Argument	Mandatory	
Address	Mandatory	
Done	Mandatory	
Entire	Mandatory	
Busy Text	Optional	

Argument

The argument shall convey the service specific parameters of the service request.

Address

This parameter contains the station address of the source station if a master is the client or the station address of the destination if a slave is the client to allow slave to slave communication.

Done

This parameter contains an information how much of the storing is finished.

Entire

This parameter contains an information how much of the storing is finished.

Busy Text

This parameter contains an optional description of the busy reason.

9.3 Master

9.3.1 Overview

The master communicates with the slaves by using the services described in the slave chapter. Additionally there is a Slave Handler for each slave defined in the master to control the ESM of the slave and a Router which shall allow the slave to slave communication via the mailbox, as noted in Figure 62.

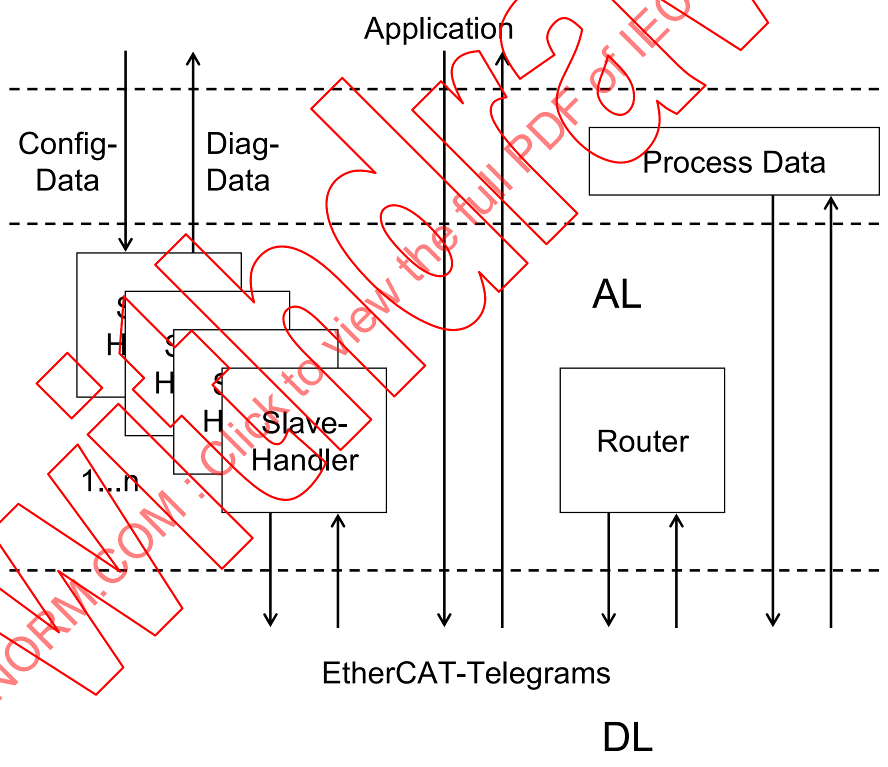


Figure 62: Master Functional Overview

9.3.2 Management

9.3.2.1 Slave Handler

The master shall support a Slave Handler for each slave using the EtherCAT State Services to control the ESM of the slave. The Slave Handler is the image of the slave's ESM in the master. Additionally the Slave Handler may send SDO Services before changing the state of the slave's ESM

Parameter

Ring Position

This parameter shall contain the position in the logical ring which is used to address the slave when reading the Identification and writing the Station Address. It is mandatory for all slaves.

Expected Identification

This parameter shall contain the expected identification of the slave which shall be read and compared by the master in the Init state. It is mandatory for all slaves.

Station Address

This parameter shall contain the station address which shall be assigned in the Init state to the slave. All further services will use this station address to address the slave. It is mandatory for all slaves.

Mailbox Configuration

This parameter contains the configuration of the Sync Manager channels 0 and 1 for the mailbox which shall be written in the Init state to the slave. It is mandatory for complex slaves.

FMMU Configuration

This parameter contains the configuration of the FMMU channels which should be written in the Pre-Operational state to the slave.

Process Data Configuration

This parameter contains the configuration of the Sync Manager channels which should be used for process data and should be written in the Pre-Operational state to the slave.

PDO Mapping

This parameter contains the PDO mapping objects which may be written in the Pre-Operational state to the slave.

Sync Manager PDO Assign

This parameter contains the Sync Manager PDO Assign objects which may be written in the Pre-Operational state to the slave.

Start Up Objects

This parameter contains the objects from the object dictionary of the slave which may be written to the slave from the Slave Handler during start up.

9.3.3 Router

The Router shall route mailbox service requests from the client slave to the server slave and mailbox service responses from the server slave to the client slave. Therefore the master shall overwrite the address field of the mailbox service request with the station address of the client slave before routing the mailbox service request to the server slave addressed by the original address field. The address field of the mailbox service response shall be overwritten by the master with the station address of the server slave before routing the mailbox service response to the client slave addressed by the original address field.

10 Contents of Part 6: Application Layer Protocol definition

10.1 Management

10.1.1 EtherCAT State Services

10.1.1.1 AL Control Service

10.1.1.1.1 AL Control Request (Indication)

10.1.1.1.1.1 Coding

```
typedef struct
{
  unsigned          State:          4;
  unsigned          Acknowledge:    1;
  unsigned          Reserved:       3;
  unsigned          ApplSpecific:   8;
} TALCONTROL;
```

10.1.1.1.1.2 Description

The Indication Description protocol is specified in Table 87.

Table 87 – AL Control Request Indication Description

Parameter	Data Type	Value
State	unsigned:4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Acknowledge	unsigned:1	0: Parameter Change of the AL Status Register will be unchanged 1: Parameter Change of the AL Status Register will be reset
Reserved	unsigned:3	Shall be zero
Application Specific	unsigned:8	

10.1.1.1.2 AL Control Response (Confirmation)

10.1.1.1.2.1 Coding

```
typedef struct
{
  unsigned          State:          4;
  unsigned          Change:         1;
  unsigned          Reserved:       3;
  unsigned          ApplSpecific:   8;
} TALSTATUS;
```

10.1.1.1.2.2 Description

The AL Control Request Confirmation protocol is specified inTable 88.

Table 88 – AL Control Request Confirmation

Parameter	Data Type	Value
State	unsigned:4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Change	unsigned:1	0: State transition successful 1: State transition unsuccessful
Reserved	unsigned:3	Shall be zero
Application Specific	unsigned:8	

10.1.1.2 AL State Changed Service

10.1.1.2.1 AL State Changed Request (Indication)

10.1.1.2.1.1 Coding

```
typedef struct
{
  unsigned           State:            4;
  unsigned           Change:           1;
  unsigned           Reserved:         3;
  unsigned           ApplSpecific:     8;
} TALSTATUS;
```

10.1.1.2.1.2 Description

The AL State Changed Request Indication protocol is specified in Table 89.

Table 89 – AL State Changed Request Indication

Parameter	Data Type	Value
State	unsigned:4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Change	unsigned:1	Shall be one
Reserved	unsigned:3	Shall be zero
Application Specific	unsigned:8	

10.1.1.3 AL State Acknowledge Service

10.1.1.3.1 AL State Acknowledge Request (Indication)

10.1.1.3.1.1 Coding

```
typedef struct
{
  unsigned           State:            4;
  unsigned           Acknowledge:      1;
  unsigned          Reserved:         3;
  unsigned          ApplSpecific:     8;
} TALCONTROL;
```

10.1.1.3.1.2 Description

The AL State Acknowledge Request Indication protocol is specified in Table 90.

Table 90 - AL State Acknowledge Request Indication

Parameter	Data Type	Value
State	unsigned:4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Acknowledge	unsigned:1	Shall be one
Reserved	unsigned:3	Shall be zero
Application Specific	unsigned:8	

10.1.1.3.2 AL State Response (Confirmation)

10.1.1.3.2.1 Coding

```
typedef struct
{
  unsigned          State:            4;
  unsigned          Change:           1;
  unsigned          Reserved:         3;
  unsigned          ApplSpecific:     8;
} TALSTATUS;
```

10.1.1.3.2.2 Description

The AL State Response Confirmation protocol is specified in Table 91.

Table 91 - AL State Response Confirmation

Parameter	Data Type	Value
State	unsigned:4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Change	unsigned:1	Shall be zero
Reserved	unsigned:3	Shall be zero
Application Specific	unsigned:8	

10.1.2 EtherCAT State Machine

10.1.2.1 Start Mailbox Communication

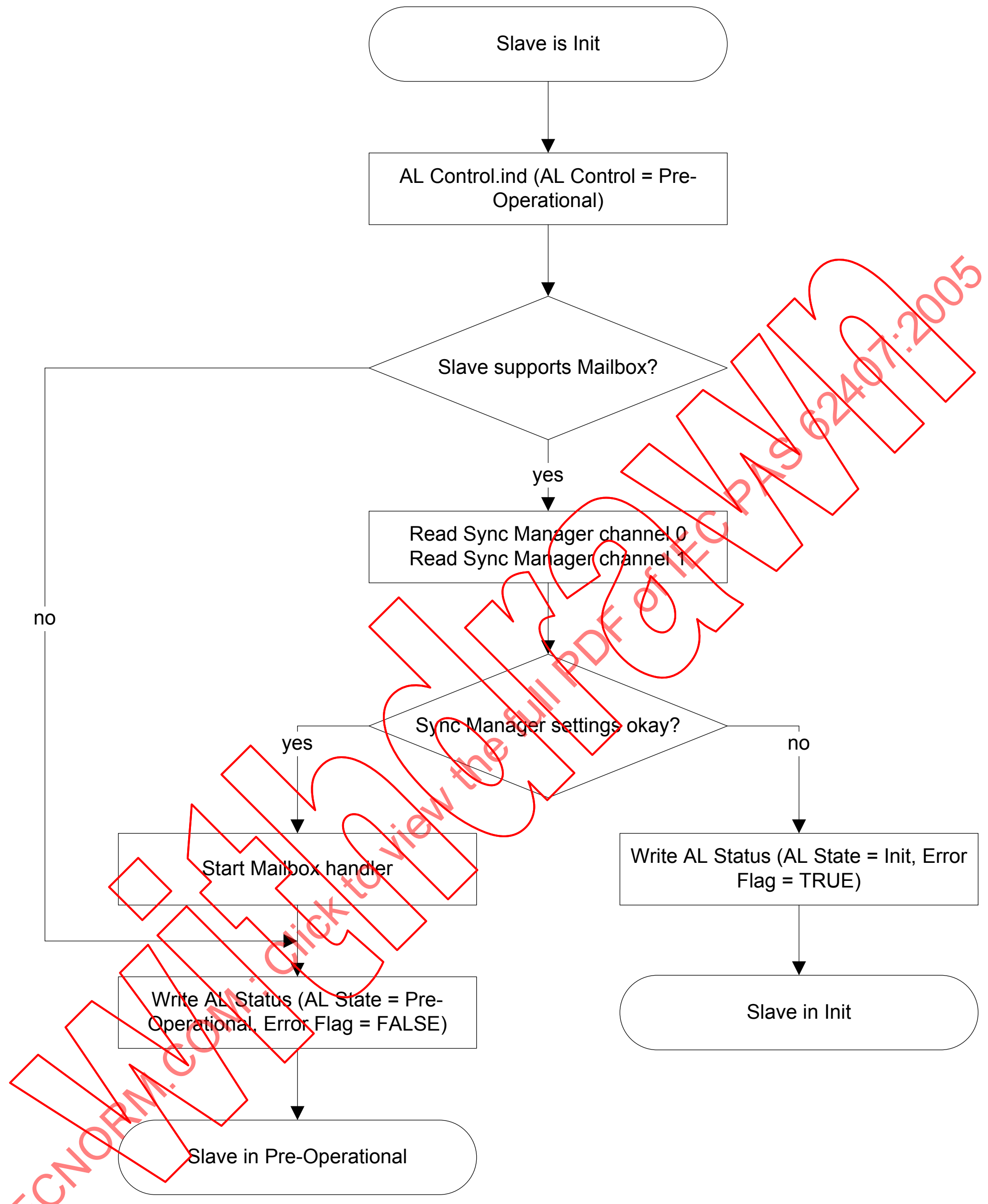

Slave is Init
AL Control.ind (AL Control = Pre-Operational)
Slave supports Mailbox?
yes
no
Read Sync Manager channel 0
Read Sync Manager channel 1
Sync Manager settings okay?
yes
no
Start Mailbox handler
Write AL Status (AL State = Init, Error Flag = TRUE)
Write AL Status (AL State = Pre-Operational, Error Flag = FALSE)
Slave in Init
Slave in Pre-Operational

Figure 63: Processing of Start Mailbox Communication in the Slave

10.1.2.2 Stop Mailbox Communication

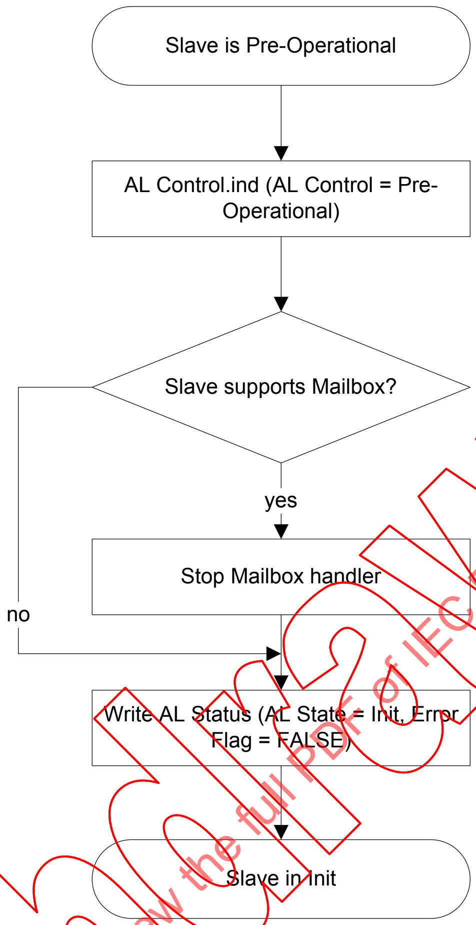

Slave is Pre-Operational
AL Control.ind (AL Control = Pre-Operational)
Slave supports Mailbox?
yes
no
Stop Mailbox handler
Write AL Status (AL State = Init, Error Flag = FALSE)
Slave in Init

Figure 64: Processing of Stop Mailbox Communication in the Slave

10.1.2.3 Start Input Update

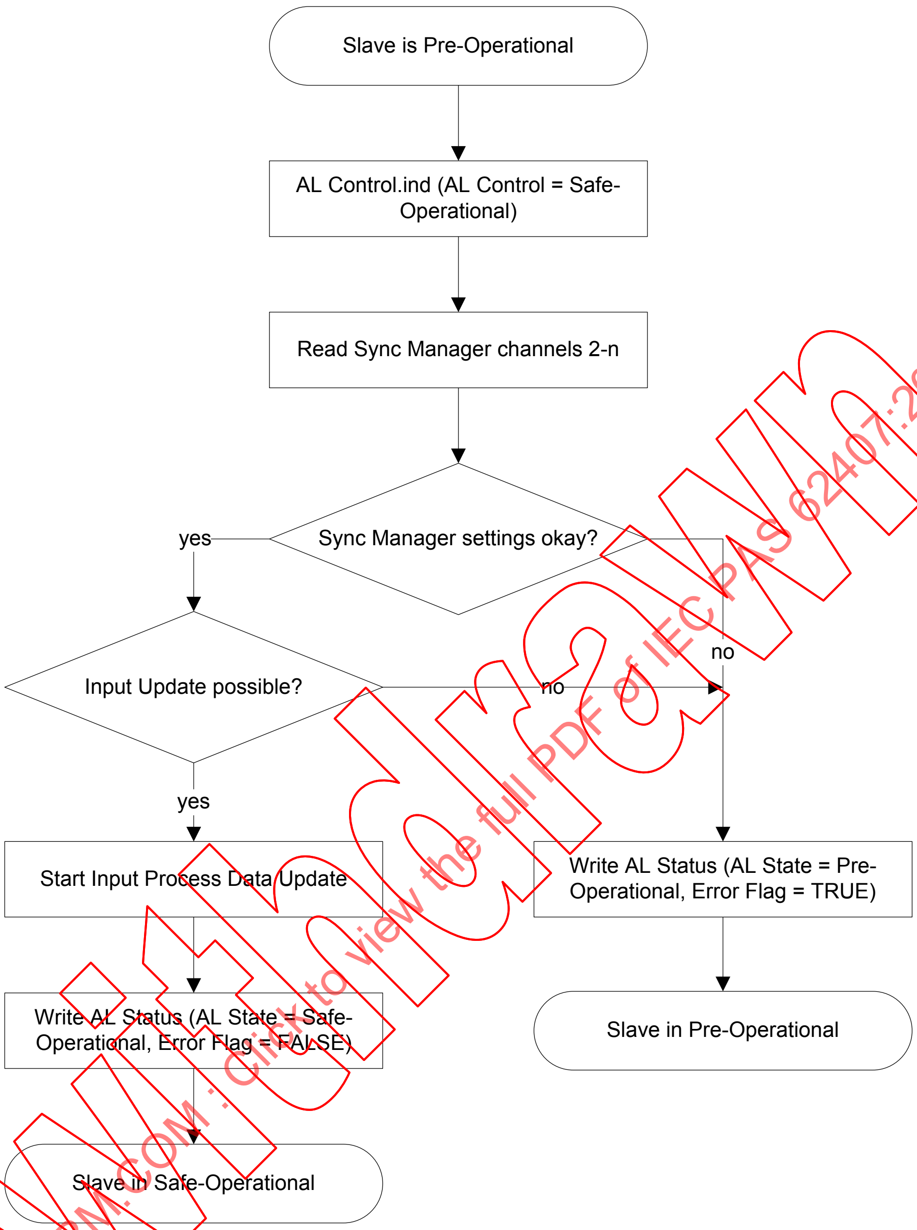

Slave is Pre-Operational
AL Control.ind (AL Control = Safe-Operational)
Read Sync Manager channels 2-n
Sync Manager settings okay?
yes
no
Input Update possible?
no
yes
Start Input Process Data Update
Write AL Status (AL State = Safe-Operational, Error Flag = FALSE)
Slave in Safe-Operational
Write AL Status (AL State = Pre-Operational, Error Flag = TRUE)
Slave in Pre-Operational

Figure 65: Processing of Start Input Update in the Slave

10.1.2.4 Stop Input Update

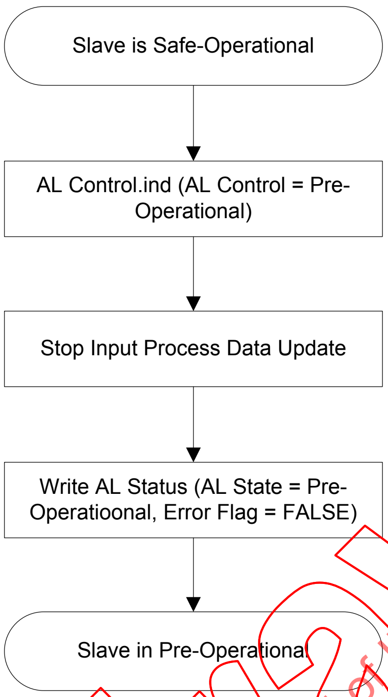

Slave is Safe-Operational
AL Control.ind (AL Control = Pre-Operational)
Stop Input Process Data Update
Write AL Status (AL State = Pre-Operatioonal, Error Flag = FALSE)
Slave in Pre-Operational

Figure 66: Processing of Stop Input Update in the Slave

10.1.2.5 Start Output Update

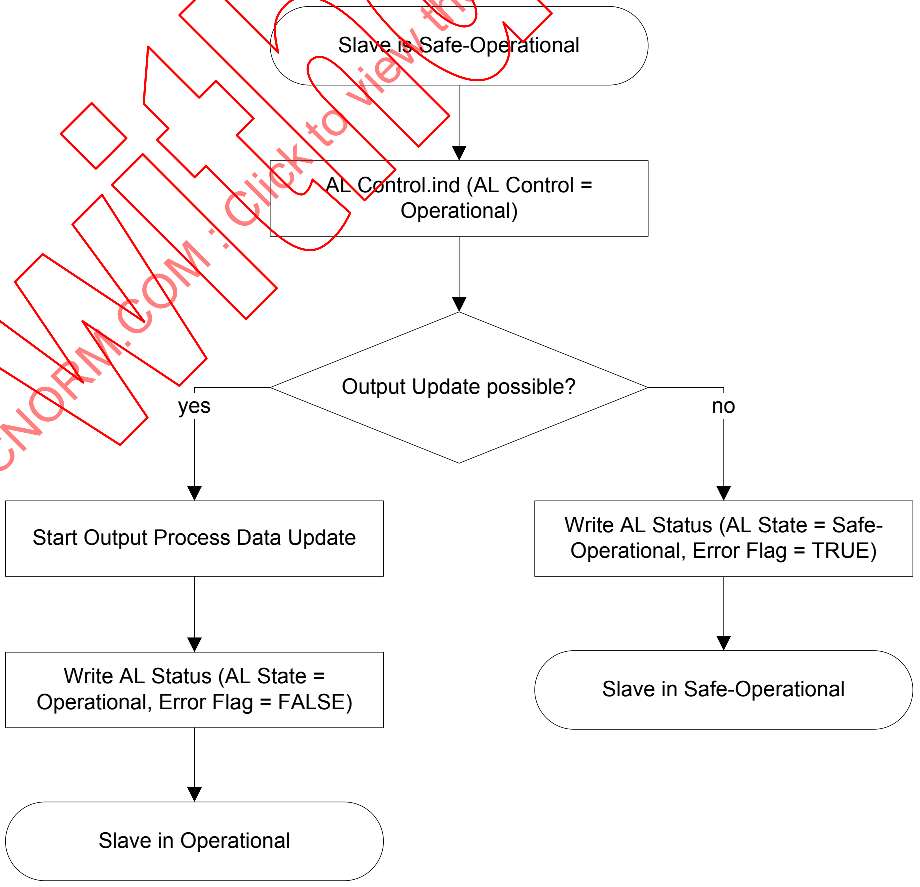

Slave is Safe-Operational
AL Control.ind (AL Control = Operational)
Output Update possible?
yes
no
Start Output Process Data Update
Write AL Status (AL State = Operational, Error Flag = FALSE)
Slave in Operational
Write AL Status (AL State = Safe-Operational, Error Flag = TRUE)
Slave in Safe-Operational

Figure 67: Processing of Start Output Update in the Slave

10.1.2.6 Stop Output Update

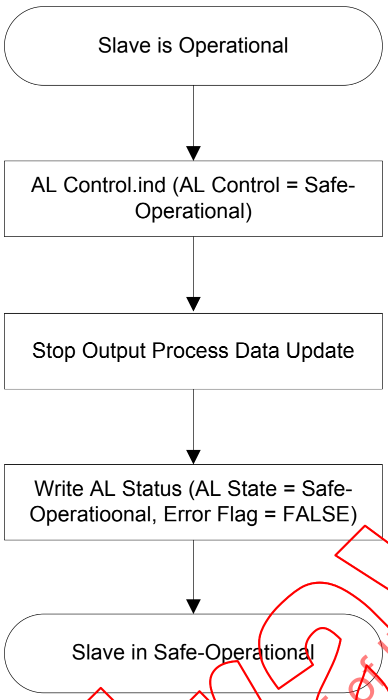

Slave is Operational
AL Control.ind (AL Control = Safe-Operational)
Stop Output Process Data Update
Write AL Status (AL State = Safe-Operatioonal, Error Flag = FALSE)
Slave in Safe-Operational

Figure 68: Processing of Stop Output Update in the Slave

10.1.2.7 Acknowledge State Change

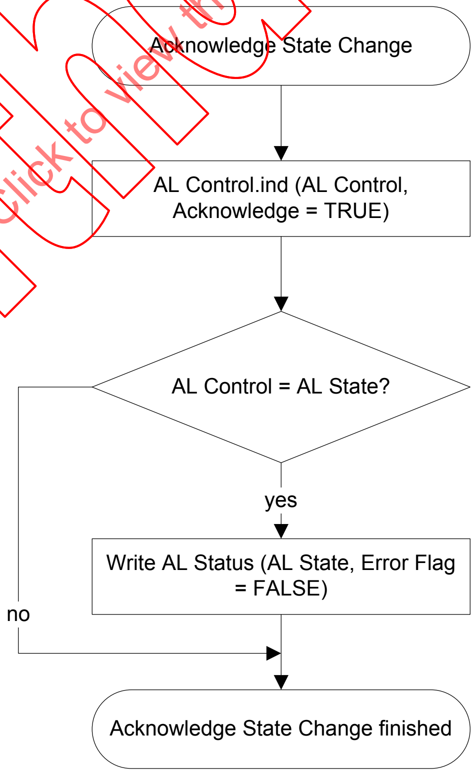

Acknowledge State Change
AL Control.ind (AL Control, Acknowledge = TRUE)
AL Control = AL State?
yes
no
Write AL Status (AL State, Error Flag = FALSE)
Acknowledge State Change finished

Figure 69: Processing of Acknowledge State Change in the Slave

10.1.3 Slave Handler

10.1.3.1 Start Mailbox Communication

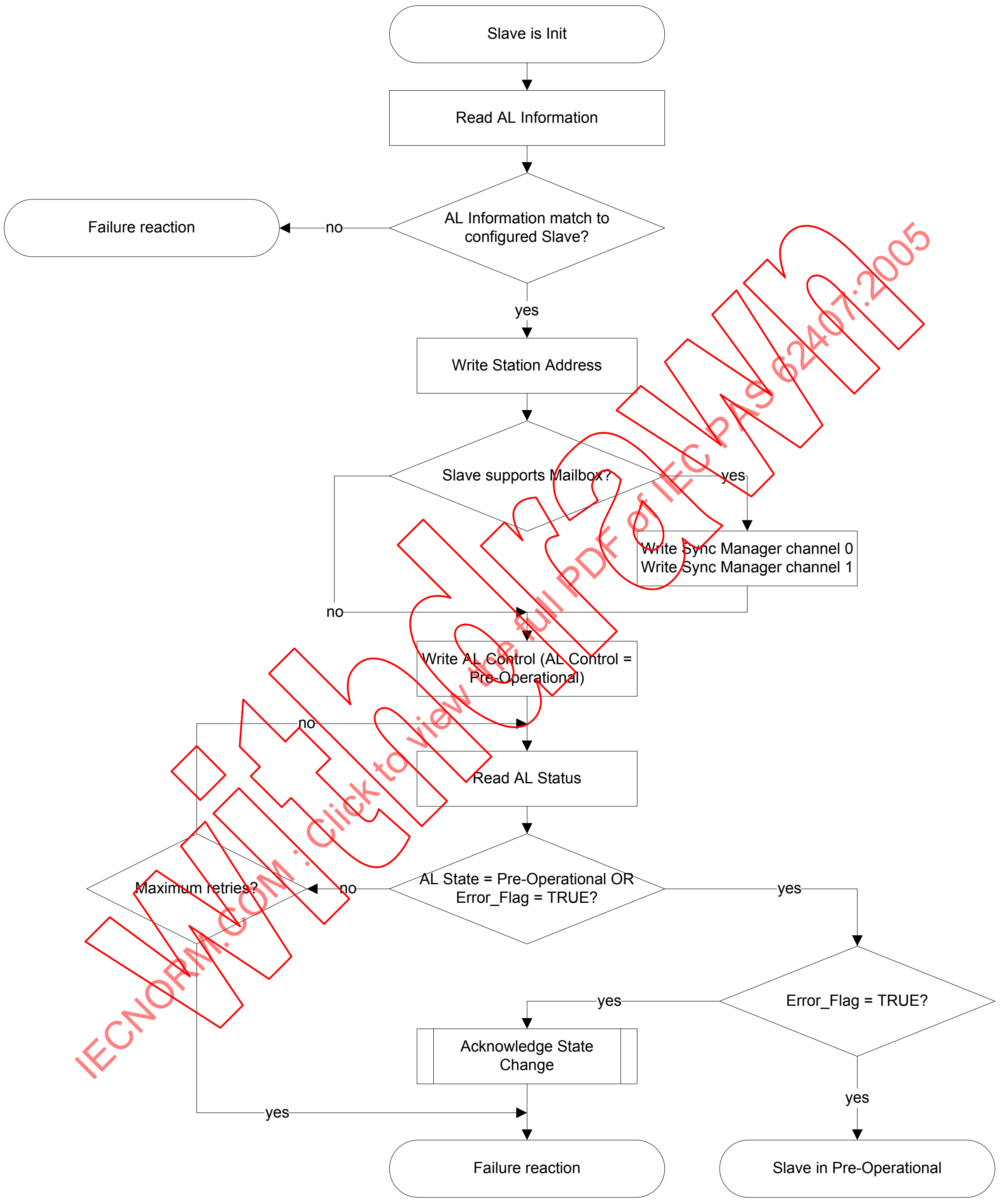


Figure 70: Processing of Start Mailbox Communication in the Master

10.1.3.2 Stop Mailbox Communication

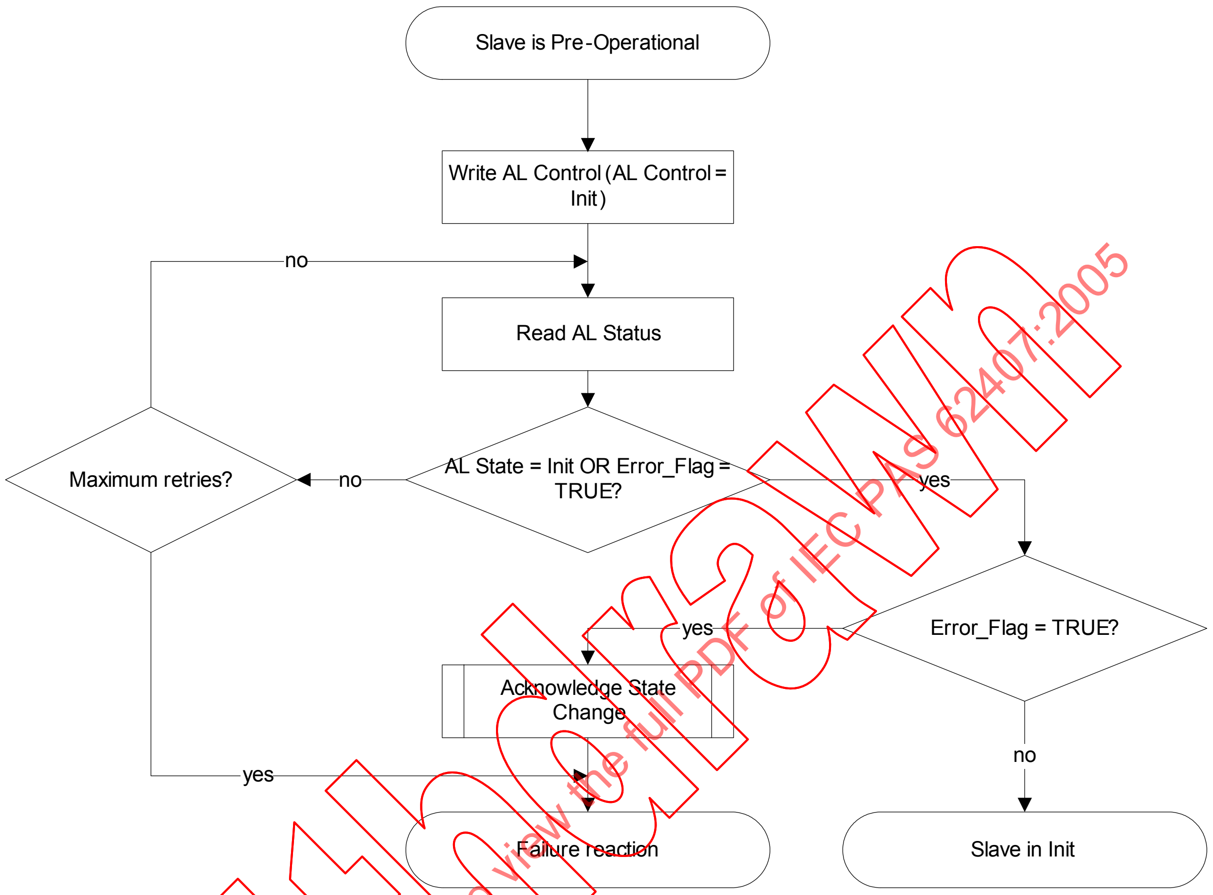
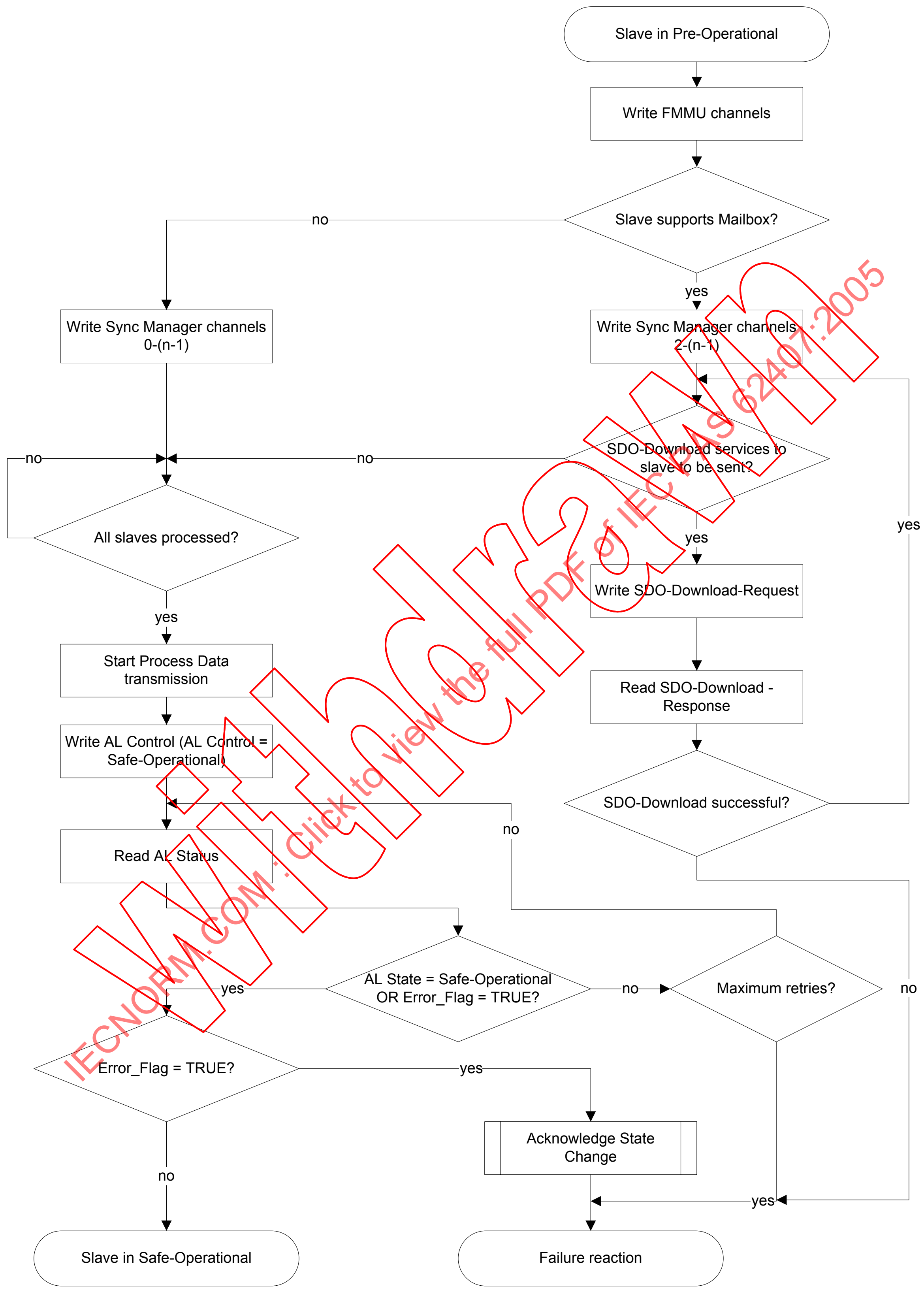


Figure 71: Processing of Stop Mailbox Communication in the Master

10.1.3.3 Start Input Update



Slave in Pre-Operational
Write FMMU channels
Slave supports Mailbox?
no
yes
Write Sync Manager channels 0-(n-1)
Write Sync Manager channels 2-(n-1)
SDO-Download services to slave to be sent?
no
no
yes
yes
All slaves processed?
yes
Write SDO-Download-Request
Read SDO-Download - Response
SDO-Download successful?
Start Process Data transmission
Write AL Control (AL Control = Safe-Operational)
Read AL Status
no
AL State = Safe-Operational OR Error_Flag = TRUE?
yes
no
Maximum retries?
no
Error_Flag = TRUE?
yes
Acknowledge State Change
no
yes
Slave in Safe-Operational
Failure reaction

Figure 72: Processing of Start Input Update in the Master

10.1.3.4 Stop Input Update

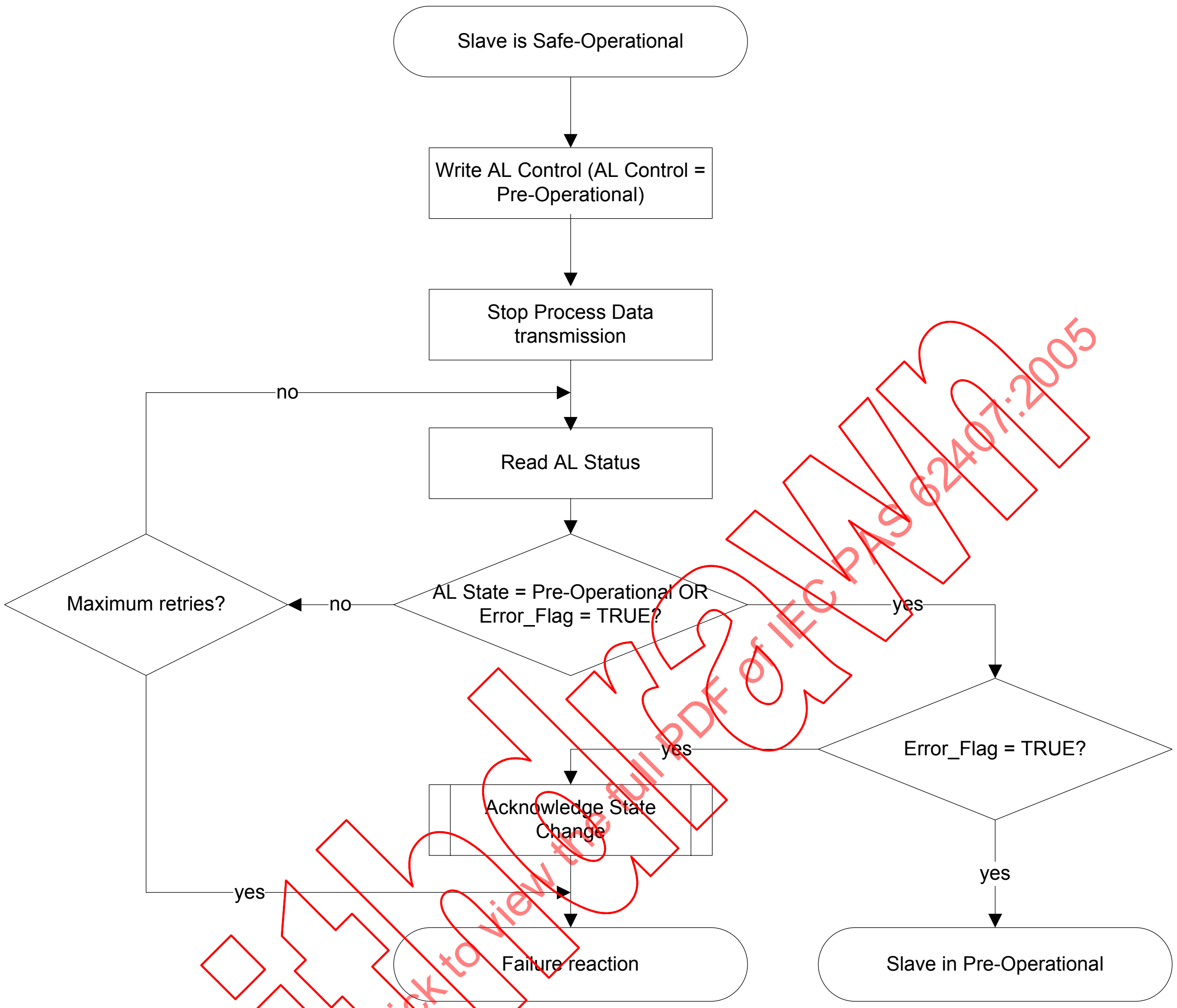

Slave is Safe-Operational
Write AL Control (AL Control = Pre-Operational)
Stop Process Data transmission
no
Read AL Status
Maximum retries?
no
AL State = Pre-Operational OR Error_Flag = TRUE?
yes
Error_Flag = TRUE?
yes
Acknowledge State Change
yes
yes
Failure reaction
Slave in Pre-Operational

Figure 73: Processing of Stop Input Update in the Master

10.1.3.5 Start Output Update

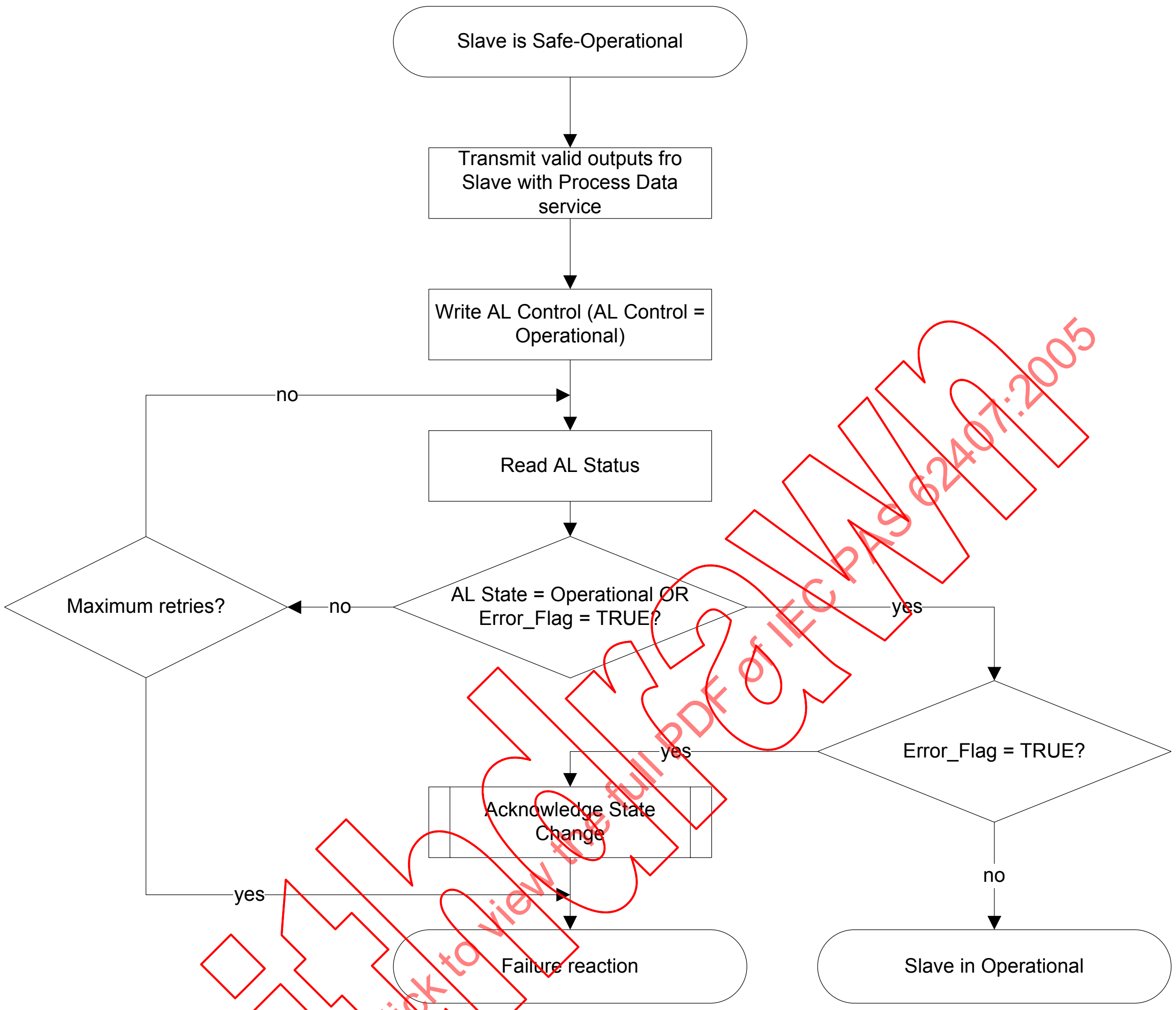

Slave is Safe-Operational
Transmit valid outputs fro Slave with Process Data service
Write AL Control (AL Control = Operational)
no
Read AL Status
Maximum retries?
no
AL State = Operational OR Error_Flag = TRUE?
yes
Error_Flag = TRUE?
yes
Acknowledge State Change
yes
no
Failure reaction
Slave in Operational

Figure 74: Processing of Start Output Update in the Master

10.1.3.6 Stop Output Update

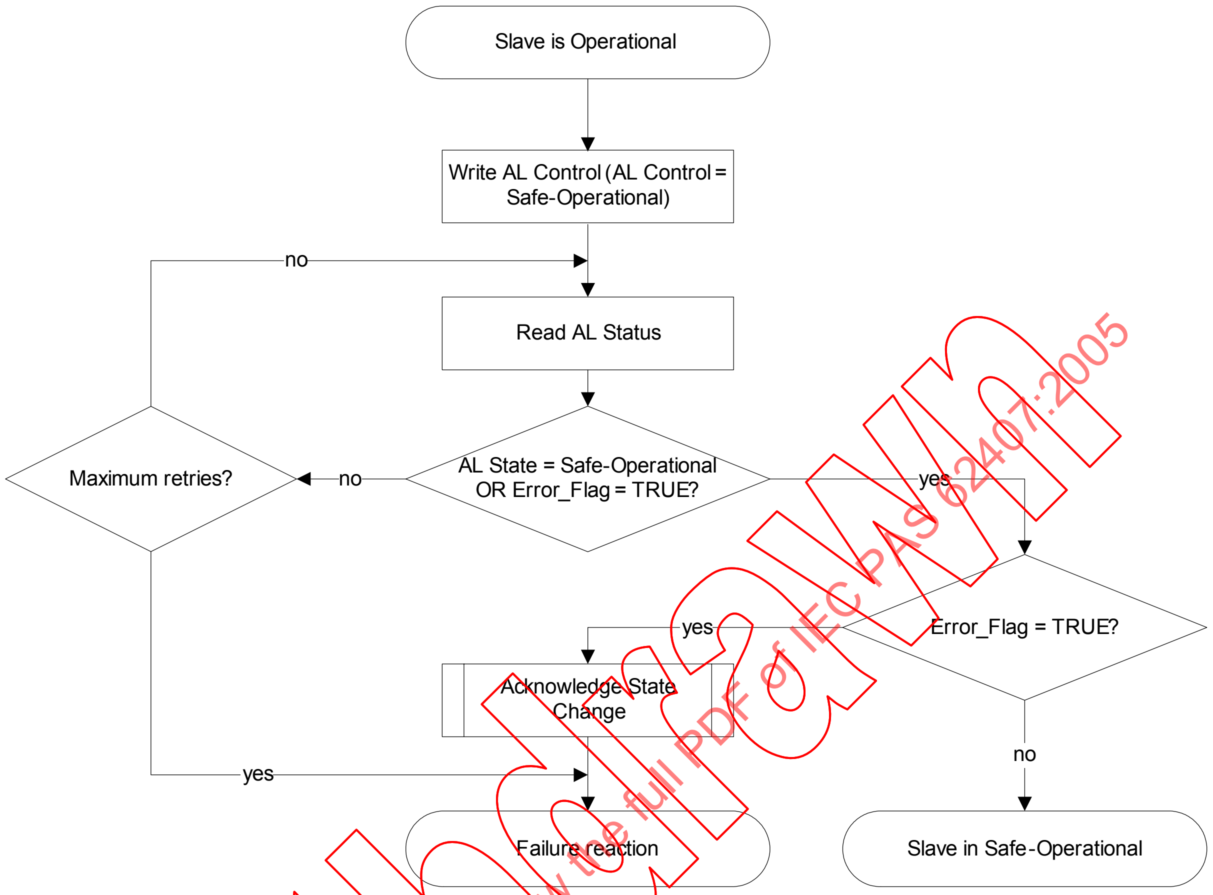


Figure 75: Processing of Stop Output Update in the Master

10.1.3.7 Acknowledge State Change

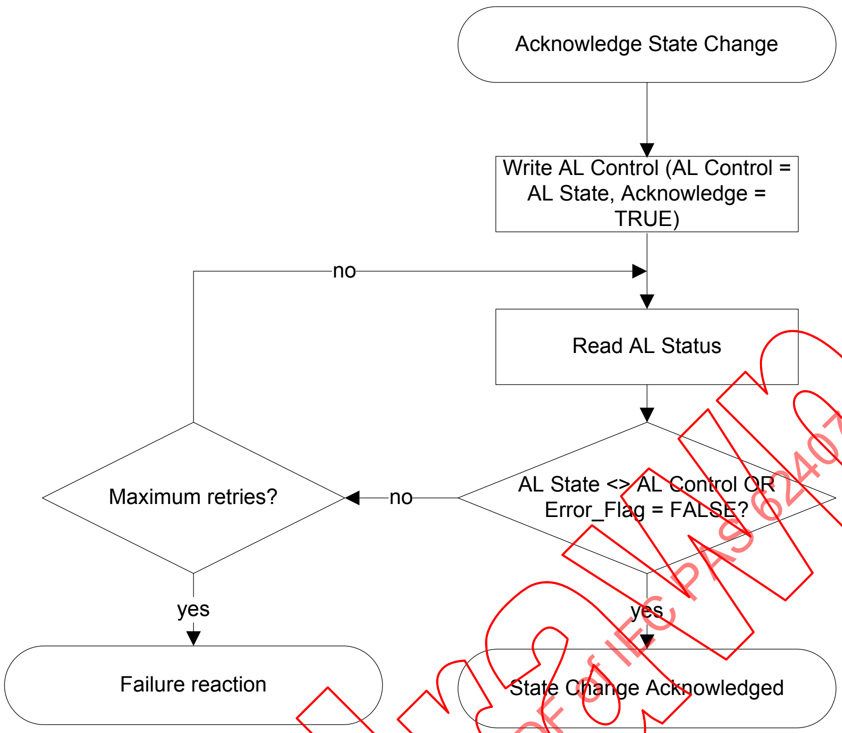

Acknowledge State Change
Write AL Control (AL Control = AL State, Acknowledge = TRUE)
no
Read AL Status
Maximum retries?
no
AL State <> AL Control OR Error_Flag = FALSE?
yes
yes
Failure reaction
State Change Acknowledged

Figure 76: Processing of Acknowledge State Change in the Master

10.2 Mailbox

10.2.1 General Mailbox Header

10.2.1.1 Coding

```
typedef struct
{
  WORD                Length;
  WORD                Address;
  unsigned            Channel:            6;
  unsigned            Priority:           2;
  unsigned            Type:               4;
  unsigned            Reserved:           4;
} TMBXHEADER;

typedef struct
{
  TMBXHEADER          MbxHeader;
  BYTE                Data[MBX_DATA_SIZE];
} TMBX;
```

10.2.1.2 Description

The General Mailbox Header protocol is specified in Table 92.

Table 92 – General Mailbox Header

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x01: ADS over EtherCAT (AoE) 0x02: Ethernet over EtherCAT (EoE) 0x03: CANopen over EtherCAT (CoE) 0x04: File Access over EtherCAT (FoE)
	Reserved	unsigned:4	0x00
Mailbox Service	Service Data	BYTE[n]	Mailbox Service Data

10.3 CANopen over EtherCAT

10.3.1 Coding

```
typedef struct
{
  unsigned          NumberLo:         8;
  unsigned          NumberHi:         1;
  unsigned          Reserved:         3;
  unsigned          Service:          4;
} TCOEHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  BYTE              Data[MBX_DATA_SIZE-2];
} TCOPMBX;
```

10.3.2 Description

The CANopen over EtherCAT protocol is specified in Table 93.

Table 93 – CANopen over EtherCAT

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x03: CANopen over EtherCAT (CoE)
	Reserved	unsigned:4	0x00
CANopen Header	Number	unsigned:9	Depending on the CANopen service
	Reserved	unsigned:3	0x00
	Service	unsigned:4	0x01: Emergency 0x02: SDO Request 0x03: SDO Response 0x04: TxPDO 0x05: RxPDO 0x06: TxPDO remote request 0x07: RxPDO remote request 0x08: SDO Information

10.3.3 SDO

10.3.3.1 Initiate SDO Download Expedited

10.3.3.1.1 Initiate SDO Download Expedited Request

10.3.3.1.1.1 Coding

```
typedef struct
{
  unsigned            SizeIndicator:     1;
  unsigned            TransferType:      1;
  unsigned            DataSetSize:       2;
  unsigned            CompleteAccess:    1;
  unsigned            Command:           3;
  BYTE                IndexLo;
  BYTE                IndexHi;
  BYTE                SubIndex;
} TINITSDOHEADER;

typedef struct
{
  TMBXHEADER          MbxHeader;
  TCOPHEADER          CopHeader;
  TINITSDOHEADER      SdoHeader;
  BYTE                Data[4];
} TINITSDODOWNLOADEXPREQMBX;
```

10.3.3.1.1.2 Description

The CANopen Initiate SDO Download Expedited Request protocol is specified in Table 94.

Table 94 – CANopen Initiate SDO Download Expedited Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x03: CANopen over EtherCAT (CoE)
	Reserved	unsigned:4	0x00
CANopen Header	Number	unsigned:9	0x00
	Reserved	unsigned:3	0x00
	Service	unsigned:4	0x02: SDO Request
SDO	Size Indicator	unsigned:1	0x00: size of Data (1..4) unspecified 0x01: size of Data in Data Set Size specified
	Transfer Type	unsigned:1	0x01: Expedited transfer
	Data Set Size	unsigned:2	0x00: 4 byte Data 0x01: 3 byte Data 0x02: 2 byte Data 0x03: 1 byte Data
	Complete Access	unsigned:1	0x00
	Command Specifier	Unsigned:3	0x01: Initiate Download Request
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object
	Data	BYTE[4]	Data of the Object,

10.3.3.1.2 Initiate SDO Download Expedited Response

10.3.3.1.2.1 Coding

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TCOPHEADER          CopHeader;
  TINITSDOHEADER      SdoHeader;
} TINITSDODOWNLOADEXPRESMBX;
```

10.3.3.1.2.2 Description

The Initiate SDO Download Expedited protocol is specified in Table 95.

Table 95 – Initiate SDO Download Expedited

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x03: CANopen over EtherCAT (CoE)
	Reserved	unsigned:4	0x00
CANopen Header	Number	unsigned:9	0x00
	Reserved	unsigned:3	0x00
	Service	unsigned:4	0x03: SDO Response
SDO	Size Indicator	unsigned:1	0x00
	Transfer Type	unsigned:1	0x00
	Data Set Size	unsigned:2	0x00
	Complete Access	unsigned:1	0x00
	Command Specifier	unsigned:3	0x03: Initiate Download Response
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object

10.3.3.2 Initiate SDO Download Normal

10.3.3.2.1 Initiate SDO Download Normal Request

10.3.3.2.1.1 Coding

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TCOPHEADER          CopHeader;
  TINITSDOHEADER      SdoHeader;
  DWORD               CompleteSize;
  BYTE                Data[MBX_DATA_SIZE-10];
} TINITSDODOWNLOADNORMREQMBX;
```

10.3.3.2.1.2 Description

The Initiate SDO Download Normal Request protocol is specified in Table 96.

Table 96 – Initiate SDO Download Normal Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n > 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x03: CANopen over EtherCAT (CoE)
	Reserved	unsigned:4	0x00
CANopen Header	Number	unsigned:9	0x00
	Reserved	unsigned:3	0x00
	Service	unsigned:4	0x02: SDO Request
SDO	Size Indicator	unsigned:1	0x01
	Transfer Type	unsigned:1	0x00: Normal transfer
	Data Set Size	unsigned:2	0x00
	Complete Access	unsigned:1	0x00: entry addressed with index and subindex will be downloaded 0x01: complete object will be downloaded, subindex shall be zero
	Command Specifier	unsigned:3	0x01: Initiate Download Request
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero if Complete Access = 0x01
	Complete Size	DWORD	Complete Data Size of the Object
	Data	BYTE[n-10]	If ((Length-10) >= Complete Size): Data of the Object If ((Length-10) < Complete Size): First Data part of the Object, Download SDO Segment is following

10.3.3.2.2 Initiate SDO Download Normal Response

10.3.3.2.2.1 Coding

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TCOPHEADER          CopHeader;
  TINITSDOHEADER      SdoHeader;
} TINITSDODOWNLOADNORMRESMBX;
```

10.3.3.2.2.2 Description

The Initiate SDO Download Normal Response protocol is specified in Table 97.

Table 97 – Initiate SDO Download Normal Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x03: CANopen over EtherCAT (CoE)
	Reserved	unsigned:4	0x00
CANopen Header	Number	unsigned:9	0x00
	Reserved	unsigned:3	0x00
	Service	unsigned:4	0x03: SDO Response
SDO	Size Indicator	unsigned:1	0x00
	Transfer Type	unsigned:1	0x00
	Data Set Size	unsigned:2	0x00
	Complete Access	unsigned:1	0x00
	Command Specifier	unsigned:3	0x03: Initiate Download Response
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object

10.3.3.3 Download SDO Segment

10.3.3.3.1 Download SDO Segment Request

10.3.3.3.1.1 Coding

```
typedef struct
{
  unsigned           MoreFollows:       1;
  unsigned           SegDataSize:       3;
  unsigned           Toggle:            1;
  unsigned           Command:           3;
} TSDOSEGHEADER;

typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOPHEADER         CopHeader;
  TSDOSEGHEADER      SdoHeader;
  BYTE               Data[MBX_DATA_SIZE-3];
} TDOWNLOADSDOSEGREQMBX;
```

10.3.3.3.1.2 Description

The Download SDO Segment Request protocol is specified in Table 98.

Table 98 – Download SDO Segment Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	unsigned:6	0x00 (Reserved for future)
	Priority	unsigned:2	0x00: lowest priority … 0x03: highest priority
	Type	unsigned:4	0x03: CANopen over EtherCAT (CoE)
	Reserved	unsigned:4	0x00
CANopen Header	Number	unsigned:9	0x00
	Reserved	unsigned:3	0x00
	Service	unsigned:4	0x02: SDO Request
SDO	More Follows	unsigned:1	0x00: Download SDO Segment is following 0x01: last Download SDO Segment
	SegData Size	unsigned:3	Defines how much of the last 7 data bytes (which always has to be send) contain data: 0x00: 7 byte Data 0x01: 6 byte Data 0x02: 5 byte Data 0x03: 4 byte Data 0x04: 3 byte Data 0x05: 2 byte Data 0x06: 1 byte Data 0x07: 0 byte Data
	Toggle	unsigned:1	Shall toggle with every Download SDO Segment Request, starting with 0x00
	Command specifier	unsigned:3	0x00: Download Segment Request
	Data	BYTE[n-2]	Data part of the Object

10.3.3.3.2 Download SDO Segment Response

10.3.3.3.2.1 Coding

```
typedef struct
{
   TMBXHEADER          MbxHeader;
   TCOPHEADER          CopHeader;
   TSDOSEGHEADER       SdoHeader;
} TDOWNLOADSDOSEGRESMBX;
```

10.3.3.3.2.2 Description

The Download SDO Segment Response protocol is specified in Table 99.