# EtherCAT Device Protocol

## ESM

### EtherCAT State Machine (ESM)

The EtherCAT State Machine defines a step-by-step set-up of each individual EtherCAT SubDevice and indicates available functionalities. A device can reject a state request from the MainDevice and signal an error with an error indication (error flag in AL Status register) and a related error code (AL Status Code register). The state request, state response and error response is exchanged via the ESC AL Control register ($_{reg}$0x0120), AL Status register ($_{reg}$0x0130) and AL Status Code register ($_{reg}$0x0134). The ESM specifies the SubDevice behavior, while the MainDevice adopts it respectively. The ESM description below outlines the basic actions for each state transition: commands sent by the MainDevice and (local) behavior of the connected devices.

**INIT** ($_{reg}$0x0120/$_{reg}$0x0130 = 0x1): Only access to ESC registers

MainDevice
- Clear: Configuration registers, e.g.:
  - FMMUs ($_{reg}$0x0600 – 0x06FF)
  - SyncManagers (SM) ($_{reg}$0x0800 – 0x087F)
- Set: Fixed Physical Address ($_{reg}$0x0010)
  - SM0 for Mailbox Out ($_{reg}$0x0800 – 0x0807)
  - SM1 for Mailbox In ($_{reg}$0x0808 – 0x080F)
- If DC: DC System Time setup:
  - Delay compensation ($_{reg}$0x0900, $_{reg}$0x0928)
  - Offset compensation ($_{reg}$0x0918, $_{reg}$0x0920)
  - Static drift compensation (~15.000 times) ($_{reg}$0x0910)

SubDevice
- Verify: Mailbox SyncManager settings
- Start: Mailbox communication (START_MBX_HANDLER)
- Confirm: State request to PRE-OPERATIONAL

SubDevice
- Clear: SM0 for Mailbox Out ($_{reg}$0x0800 – 0x0807); SM1 for Mailbox In ($_{reg}$0x0808 – 0x080F)
- Stop: Mailbox communication (STOP_MBX_HANDLER)

**PRE-OPERATIONAL** ($_{reg}$0x0120/$_{reg}$0x0130 = 0x2): Mailbox communication available

**BOOT** ($_{reg}$0x0120/$_{reg}$0x0130 = 0x3): optional

MainDevice
- Set: Configuration objects via SDO, e.g.:
  - RxPDO / TxPDO Assignment ($_{obj}$0x1C12 / $_{obj}$0x1C13)
  - RxPDO / TxPDO Mapping ($_{obj}$0x1A00 – 0x1BFF / $_{obj}$0x1600 – 0x17FF)
  - SM2 for outputs ($_{reg}$0x0810 – 0x0817)
  - SM3 for inputs ($_{reg}$0x0818 – 0x081F)
  - FMMU0 (maps outputs) ($_{reg}$0x0600 – 0x060F)
  - FMMU1 (maps inputs) ($_{reg}$0x0610 – 0x061F)
- If DC: Configure SYNC/LATCH unit:
  - Set SYNC cycle time ($_{reg}$0x09A0 – 0x09A7)
  - Set DC start time ($_{reg}$0x0990 – 0x0997)
  - Set DC SYNC OUT unit ($_{reg}$0x0980 – 0x0981)
  - Set DC LATCH IN unit ($_{reg}$0x09A8 – 0x09A9)
  - Start continuous drift compensation ($_{reg}$0x0910 – 0x0917)
- Start: Cyclic process data

SubDevice
- Verify: Process data SyncManager settings, PDO Mapping/Assignment
- Start: Input update (START_INPUT_HANDLER)
- Provide: Valid Inputs
- Confirm: State request to SAFE-OPERATIONAL

SubDevice
- Stop: Cyclic process data

**SAFE-OPERATIONAL** ($_{reg}$0x0120/$_{reg}$0x0130 = 0x4): Process data available, outputs still in safe state

MainDevice
- Provide: Valid outputs

SubDevice
- Verify: Synchronization with DC, if required
- Start: Output update (START_OUTPUT_HANDLER)
- Confirm: State request to OPERATIONAL

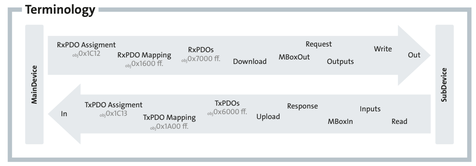
SubDevice
- Stop: Output update (STOP_OUTPUT_HANDLER)

**OPERATIONAL** ($_{reg}$0x0120/$_{reg}$0x0130 = 0x8): Input and output process data available

State transition/default timeout: PreopTimeout (3000ms) → SafeopOPTimeout (10000ms) → BackToSafeopTimeout (200ms) → BackToInitTimeout (5000ms)

References: ETG.1000.5 – AR (Application Relationship); ETG.1000.4 – Attributes; ETG.1000.6 – AL state machine; ETG.2000 – InfoType

## Terminology




## CoE

### CoE Object Dictionary (OD)

The OD is based on CANopen® and provides a structure for any EtherCAT device. Each object is addressed by a 16-bit index and can have up to 255 object entries, addressed by an 8-bit subindex (SI). The following Object Codes are distinguished:

- VARIABLE: SI0 only
- ARRAY*: all entries have the same data type and name except SI0
- RECORD*: entries may have different data types and names

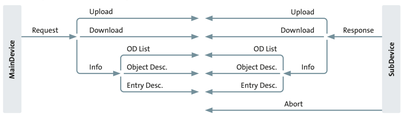
*SI0 defines the highest supported subindex. SI1 always has a 16-bit offset.

### CoE base data types (excerpt)

| Index | Data Type | Index | Data Type | Index | Data Type | Index | Data Type |
|---|---|---|---|---|---|---|---|
| 0x0001 | BOOL/BIT | 0x0006 | UINT | 0x0008 | ARRAY of UINT | 0x001F | WORD |
| 0x0002 | SINT | 0x0007 | UDINT | 0x0011 | LREAL | 0x0020 | DWORD |
| 0x0003 | INT | 0x0008 | REAL | 0x0015 | LINT | 0x0260 | ARRAY of INT |
| 0x0004 | DINT | 0x0009 | STRING(n) | 0x0018 | ULINT | 0x0261 | ARRAY of SINT |
| 0x0005 | USINT | 0x000A | ARRAY of BYTE | 0x001E | BYTE | 0x0262 | ARRAY of DINT |

Reference: ETG.1020 – Base Data Types

### CoE SDO (Service Data Object) services




SDO services are used to read from and write to the online OD of the SubDevice in a confirmed fashion. They are also used to read the OD structure. The MainDevice sends an SDO request, the SubDevice sends either an SDO Response or an SDO Abort. The following SDO Request/Response services are defined:

- SDO Upload: read object/object entry from online OD
- SDO Download: write object/object entry from online OD
- Get OD List: returns all object indexes of the online OD
- Get Object Description: returns object name/code, data type and max. number of entries
- Get Entry Description: returns value info, data type, bit length, access rights of an entry

Reference: ETG.1000.5 – CoE service specification | ETG.1000.6 – CoE coding

| SDO Command Specifier | SDO Services | | |
|---|---|---|---|
| | 0x02: SDO Request | 0x03: SDO Response | 0x08: SDO Info |
| 0x00 | Download Segmented | Upload Segmented | |
| 0x01 | Download | Download Segmented | Get OD List Req. |
| 0x02 | Upload | Upload | Get OD List Resp. |
| 0x03 | Upload Segmented | Download | Get Object Description Req. |
| 0x04 | Abort Transfer | | Get Object Description Resp. |
| 0x05 | | | Get Entry Description Req. |
| 0x06 | | | Get Entry Description Resp. |
| 0x07 | | | SDO Info Error Req. |

### Modular Device Profile (MDP)

The MDP provides a basic structure for any EtherCAT SubDevice to organize physical or logical modules within the device, based on the CoE OD. The SubDevice's data is grouped based on its physical structure and/or logical/software structure. The MDP structure is based on so-called modules. A module has an assigned index range, typically:

- 1 RxPDO, 1 TxPDO (PdoIncrement = 1)
- 16 objects (with up to 255 entries per object) per functional index area (inputs, outputs, configuration, information, diagnosis) (IndexIncrement = 16).

Information which is not specific for a module is organized in the 0xFxxx Device index area.

### CoE Object Dictionary structure, incl. MDP structure

General OD structure, MDP structure (0x6000 ff.) and attributes of the functional index areas:

| Object Dictionary | MDP Device Module 0 | Module 1 | ... | Module n |
|---|---|---|---|---|
| Communication area (0x1000 – 0x1FFF) e.g. object 0x1000, 0x1018, 0x10F3 | | | | |
| RxPDOs (0x1600 – 0x17FF) | 0x1600 | 0x1601 | ... | 0x16nn |
| TxPDOs (0x1A00 – 0x1BFF) | 0x1A00 | 0x1A01 | ... | 0x1Ann |
| Manufacturer specific area (0x2000 – 0x5FFF) | | | | |
| Input area (0x6000 – 0x6FFF) Tx-mappable, read-only | 0x6000 – 0x600F | 0x6010 – 0x601F | ... | 0x6nn0 – 0x6nnF |
| Output area (0x7000 – 0x7FFF) Rx-mappable, read-writeable | 0x7000 – 0x700F | 0x7010 – 0x701F | ... | 0x7nn0 – 0x7nnF |
| Configuration area (0x8000 – 0x8FFF) read-writeable, usually not mappable | 0x8000 – 0x800F | 0x8010 – 0x801F | ... | 0x8nn0 – 0x8nnF |
| Information area (0x9000 – 0x9FFF) read-only, usually not mappable | 0x9000 – 0x900F | 0x9010 – 0x901F | ... | 0x9nn0 – 0x9nnF |
| Diagnosis area (0xA000 – 0xAFFF) | 0xA000 – 0xA00F | 0xA010 – 0xA01F | ... | 0xAnn0 – 0xAnnF |
| Device area (0xF000 – 0xFFFF) e.g. object 0xF000, 0xF010, 0xF030, 0xF050 | | | | |

Reference: ETG.5001 – MDP Device model

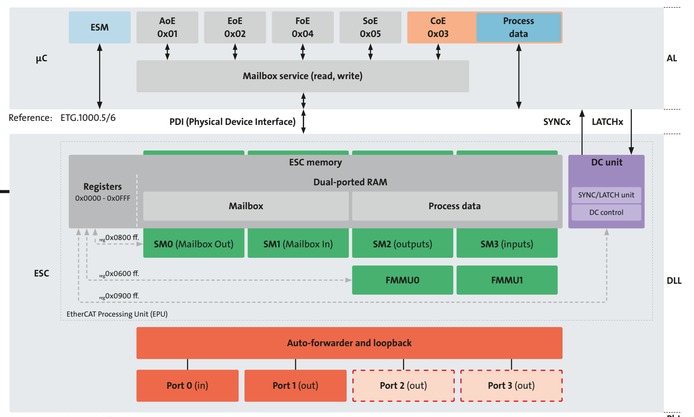
### Profiles

The major part of profiles is based on the CoE OD (incl. range 0x6000 ff.), while there are MDP-based profiles and the IEC61800 drive profiles (incl. "DS402"). The 32-bit profile number is provided by the SubDevice via $_{obj}$0x1000, $_{obj}$0xF010 and via ESI element *Device:Profile* or *Module:Profile*. Bit 0-15 is the device profile number (e.g. 5003 for Semi Device Profile). Bit 16-31 is the module profile number (e.g. 2020 Mass Flow Controller). Important profiles are:

- ETG.5001.1 – General MDP Device Model Specification: Basic structure for any EtherCAT SubDevice
- ETG.5001.3 – MDP Fieldbus Gateway Profile Specification: Incl. profiles for EtherCAT MainDevices, Profibus DP, CAN, CANopen, DeviceNet
- ETG.5001.4 – MDP Safety Module Specification: Incl. profiles for FSoE Digital I/O connection, FSoE Safety Drive Profile, FSoE MainDevice
- ETG.5003 – Semi Device Profile: Based on MDP structures, incl. profiles for mass flow controllers, temperature controllers, pressure gauges, valves, chillers, pumps, RF DC generators
- ETG.6010 – Implementation Directive for CiA402 Drive Profile (IEC61800-7-201)

### SubDevice structure and EtherCAT SubDevice Controller (ESC)

The ESC processes the EtherCAT frames on the fly in hardware and implements the functionalities of the data link layer (DLL). Those include SyncManagers (SM), Fieldbus Memory Management Units (FMMU) and DC unit. Typically, EtherCAT SubDevices implement the application layer (AL) functionalities such as the ESM, parameter handling and process data handling on a µC – such SubDevices are called "complex device". Only for very simple I/O devices without parameters and without necessary AL error handling, the I/O hardware drivers are connected directly to the digital I/O interface of the ESC – such SubDevices are called "simple device". The block diagram combines elements of the SubDevice's hardware structure, functional entities of the ESC, software structure and protocol elements.




Reference: ETG.1000.5/6

Reference: ETG.2010

Reference: ETG.1000.3/4

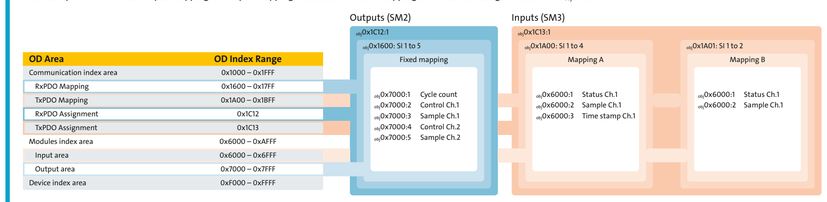
## Process data

### Process data configuration

The EtherCAT process data configuration allows very flexible PDO description. PDO configuration can be either fixed, selectable or configurable. When using MDP objects, inputs ($_{obj}$0x6000 – 0x6FFF) and outputs ($_{obj}$0x7000 – 0x7FFF) are mapped to PDO Mapping objects ($_{obj}$0x1600 – 0x17FF for RxPDO Mapping and $_{obj}$0x1A00 – 0x1BFF for TxPDO Mapping) and assigned to the respective SM via $_{obj}$0x1C12 (SM2) and $_{obj}$0x1C13 (SM3) for PDO Assignment. PDO Mapping and PDO Assignment objects are used by complex devices (with online and offline OD) as well as by simple devices (only in the EtherCAT SubDevice Information file).

### Example: PDO Mapping and PDO Assignment

This example shows a fixed output mapping. The input mapping is selectable; either mapping A or B can be sassigned to SM3 via $_{obj}$0x1C13




| OD Area | OD Index Range |
|---|---|
| Communication index area | 0x1000 – 0x1FFF |
| RxPDO Mapping | 0x1600 – 0x17FF |
| TxPDO Mapping | 0x1A00 – 0x1BFF |
| RxPDO Assignment | 0x1C12 |
| TxPDO Assignment | 0x1C13 |
| Modules index area | 0x6000 – 0xAFFF |
| Input area | 0x6000 – 0x6FFF |
| Output area | 0x7000 – 0x7FFF |
| Device index area | 0xF000 – 0xFFFF |

References: ETG.1000.5 – Process data interaction | ETG.1000.6 – Object Dictionary | ETG.5001.1 – PDO Mapping and PDO Assign

## DC unit and synchronization

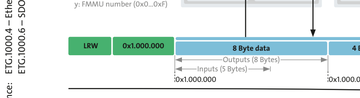
### Distributed Clocks (DC) and synchronization

Synchronization of MainDevice and SubDevice applications is based on a common time in the network (DC System Time). The synchronization modes define how this common time is used to synchronize the local applications. The SYNC/LATCH unit is used to generate SYNC/LATCH events based on the System Time.

### DC System Time

The System Time is a 64-bit ns-based time starting 01.01.2000, 0:00h, or a 32-bit time respectively. After setting up the DC time, MainDevice and SubDevices share the same time base. The first DC SubDevice behind the MainDevice is used as the reference clock. Each SubDevice has a local copy of the System Time stored in $_{reg}$0x0910.

### DC System Time setup (1-3) and continuous drift compensation (4)




1. **Propagation Delay:** write to $_{reg}$0x0900 latches receive times on all ports; read latched times, calculate delays; write individual values to $_{reg}$0x0928
2. **Offset:** read individual local times; calculate offset to time reference; write individual offsets values to $_{reg}$0x0920
3. **Static Drift:** read System Time from reference clock and write to individual DC SubDevices via multiple (~15.000) xRMW datagrams
4. **Continuous Drift:** distribute System Time (xRMW) together with cyclic frames, e.g. every ms to keep deviation of distributed times small

Reference: ETG.1000.4 – Distributed clock

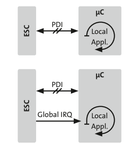



Reference: ETG.1020 – Synchronization

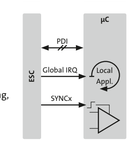
### Synchronization modes

Applications may require different degrees of synchronization which is reflected by the different EtherCAT synchronization modes. The basic operation for DC modes is setup via $_{reg}$0x0980 – 0x0981. Additional information is provided by the SyncManager Parameter objects $_{obj}$0x1C32 for SM2 and $_{obj}$0x1C33 for SM3 (incl. minimum cycle time, calc and copy time, error counters).

**Free Run:** Application is triggered by local clock and runs independently from EtherCAT cycle. ESI element *Device:Dc* is not available and $_{obj}$0x1C32/3 is optional if only Free Run is supported.

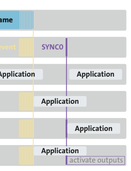


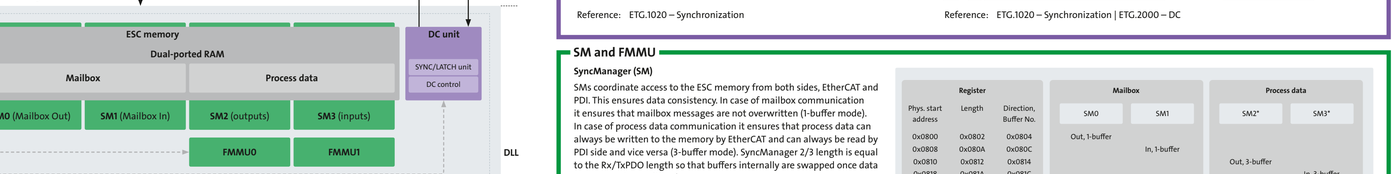

**SM-Synchronous:** Application is synchronized with the SM2 (SM3) event, which is generated when process data is written to SM2 (read from SM3). Events are mapped to global IRQ or polled from $_{reg}$0x0220. ESI element *Device:Dc* is not available if only SM-Synchronous is supported. If both, SM-Synchronous and DC-Synchronous are supported, then it is indicated in the ESI by *Dc:AssignActivate* = "#x0000"




**DC-Synchronous:** Application is synchronized using DC-based interrupt signals (SYNC0, SYNC1; ns-accuracy), generated by the SYNC/LATCH unit. Among many other DC modes, the following two show basic concepts:

- SYNC0: triggers the complete processing of the local cycle (see Minimum Cycle Time)
- SM2 and SYNC0 (even higher synchronization accuracy of the output event): SM2 event triggers reading of output data from SM2, processing, writing values to hardware drivers; then the SYNC0 event is used to activate output drivers. ESI element *Dc:AssignActivate* = "#x0300" for SYNC0 event generation




Reference: ETG.1020 – Synchronization | ETG.2000 – DC

## SM and FMMU

### SyncManager (SM)

SMs coordinate access to the ESC memory from both sides, EtherCAT and PDI. This ensures data consistency. In case of mailbox communication it ensures that mailbox messages are not overwritten (1-buffer mode). In case of process data communication it ensures that process data can always be written to the memory by EtherCAT and can always be read by PDI side and vice versa (3-buffer mode). SyncManager 2/3 length is equal to the Rx/TxPDO length so that buffers internally are swapped once data was completely written/read.

Reference: ETG.1000.4 – SyncManager

| Register | | | Mailbox | | Process data | |
|---|---|---|---|---|---|---|
| Phys. start address | Length | Direction, Buffer No. | SM0 | SM1 | SM2* | SM3* |
| 0x0800 | 0x0802 | 0x0804 | Out, 1-buffer | | Out, 3-buffer | |
| 0x0808 | 0x080A | 0x080C | | In, 1-buffer | | In, 3-buffer |
| 0x0810 | 0x0812 | 0x0814 | | | | |
| 0x0818 | 0x081A | 0x081C | | | | |

*physical memory = 3 times Rx/TxPDO length

### Fieldbus Memory Managment Unit (FMMU)

Typically, logical commands (LRD, LWR, LRW) are used for process data exchange: A single Lxx command addresses one or multiple SubDevices. The FMMUs of the individual SubDevices are configured during start-up to map the data from the EtherCAT command (logical address space) to the physical memory and vice versa. FMMUs are configured via registers starting at $_{reg}$0x0600, see also example 3.

Reference: ETG.1000.4 – Fieldbus memory management unit

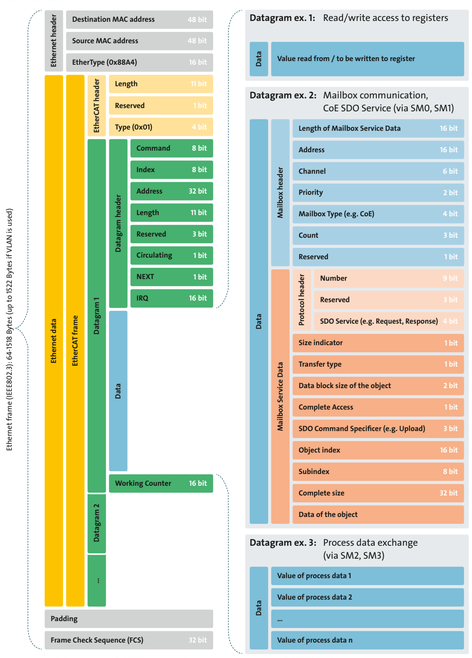
## Frame processing and examples

### Communication principle

EtherCAT communication is always initiated by the MainDevice by sending frames via its Ethernet interface. Those are processed on the fly by the ESC. Processing within the ESC works in a "roundabout" fashion: Behind the EtherCAT Processing Unit (EPU) the frame is forwarded to the next port (and, if open, sent out to be processed by other SubDevices), while the returning frame is sent back to the MainDevice via port 0. Port 0 shall always be the IN port of the SubDevice. The topology always forms a logical ring, and neither frame collision nor congestion can occur by design. Throughput time can be calculated precisely, and errors can be detected easily via status and error counter registers ($_{reg}$0x0100, $_{reg}$0x0300 – 0x0313).

Reference: ETG.1000.4 – Frame processing principles

### Frame structure




**Datagram ex. 1:** Read/write access to registers




**Datagram ex. 2:** Mailbox communication, CoE SDO Service (via SM0, SM1)

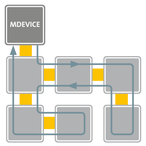



**Datagram ex. 3:** Process data exchange (via SM2, SM3)

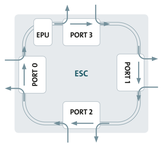



References: ETG.1000.4 – EtherCAT frame structure | ETG.1000.6 – CoE coding

### Logical ring




### Frame processing and forw. order

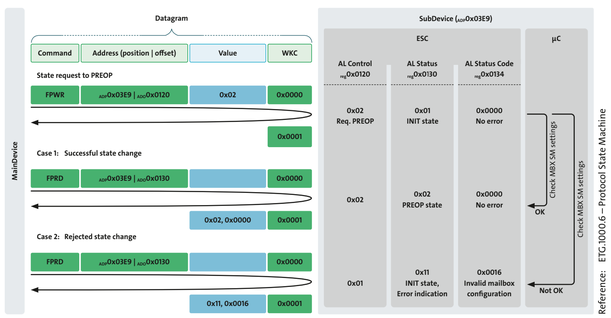



### Specific port and error registers

| Register | Port 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| DL Control | $_{reg}$0x0100 — 0x0103 | | | |
| DL Status | $_{reg}$0x0110 — 0x0111 | | | |
| Frame Error Counter | $_{reg}$0x0300 | $_{reg}$0x0302 | $_{reg}$0x0304 | $_{reg}$0x0306 |
| Phys. Error Counter | $_{reg}$0x0301 | $_{reg}$0x0303 | $_{reg}$0x0305 | $_{reg}$0x0307 |
| Link Lost Counter | $_{reg}$0x0310 | $_{reg}$0x0311 | $_{reg}$0x0312 | $_{reg}$0x0313 |

### Datagram example 1: read/write access to register

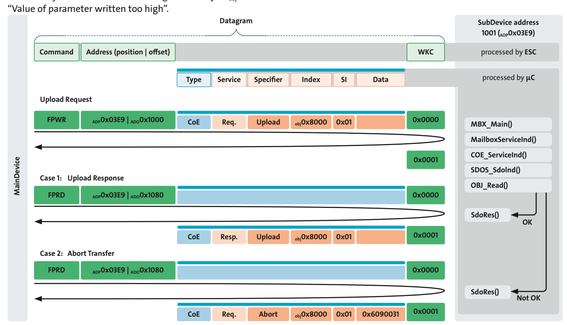
In this example ESM register 0x0120, 0x0130, 0x0134 are written and read for state machine interactions.




Reference: ETG.1000.6 – Protocol State Machine

### Datagram example 2: Mailbox communication, CoE SDO Service (via SM0, SM1)

In this example an SDO Download Request is written to SM0 Mailbox Out and the response/abort is read from SM1 Mailbox In: A value is successfully downloaded to the configuration object $_{obj}$0x8000:01 in case 1 and an abort is returned in case 2 with Abort Code 0x06090031 "Value of parameter written too high".




Reference: ETG.1000.4 – EtherCAT frame structure; ETG.1000.6 – SDO

### Datagram example 3: Process data exchange (FMMU)

In this example process data is exchanged cyclically using a Logical Read Write command (LRW) and mapped by FMMUs to and from the SubDevice's DPRAM (SM2/3).




Reference: ETG.1000.4 – Fieldbus memory management unit

### Commands

EtherCAT commands address one or several SubDevices. Position (APxx), Node (FPxx), Logical (Lxx), and Broadcast (Bxx) addressing is possible. With each successfull read/write interaction every SubDevice increments the Working Counter (WKC).

| Specifier | Cmd | Description |
|---|---|---|
| 0x00 | NOP | No operation |
| 0x01 | APRD | Auto increment physical read |
| 0x02 | APWR | Auto increment physical write |
| 0x03 | APRW | Auto increment physical read write |
| 0x04 | FPRD | Configured address physical read |
| 0x05 | FPWR | Configured address physical write |
| 0x06 | FPRW | Configured address physical read write |
| 0x07 | BRD | Broadcast read |
| 0x08 | BWR | Broadcast write |
| 0x09 | BRW | Broadcast read write |
| 0x0A | LRD | Logical memory read |
| 0x0B | LWR | Logical memory write |
| 0x0C | LRW | Logical memory read write |
| 0x0D | ARMW | Auto increment physical read multiple write |
| 0x0E | FRMW | Configured address physical read multiple write |

### Working Counter

| Command | Action | Increment |
|---|---|---|
| xRD | Successful read | +1 |
| xWR | Successful write | +1 |
| xRW | Successful read | +1 |
| | Successful write | +2 |

Reference: ETG.1000.4 – EtherCAT frame structure; ETG.1000.4 – Error detection overview

# EtherCAT Device Protocol Poster

EtherCAT®