**EtherCAT Master System Requirements**

Author: Davide Antonucci, Alessio Margan.

Manager: Nikos Tsagarakis.

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Name** | **Data** | **Comments** |
| **0.0.0** | Davide Antonucci | 09/06/2020 | Initial Work and requirements collection. |

Index

[1 Scope 4](#__RefHeading___Toc706_2097848850)

[2 EtherCAT Master Design 5](#__RefHeading___Toc708_2097848850)

[2.1 EtherCAT Protocol Concept 5](#__RefHeading___Toc715_2097848850)

[2.1.1 EtherCAT Frame and Slave Communication 7](#__RefHeading___Toc825_2097848850)

[2.1.2 Service Data Objects (SDO) 8](#__RefHeading___Toc827_2097848850)

[2.1.3 Process Data Objects(PDO) 8](#__RefHeading___Toc829_2097848850)

[2.1.4 Distributed Clocks for High-Precision Synchronization 9](#__RefHeading___Toc849_2097848850)

[2.1.5 EtherCAT Master State Machine 10](#__RefHeading___Toc851_2097848850)

[2.1.6 Libraries 11](#__RefHeading___Toc853_2097848850)

[2.2 ZMQ or Zero MQ Concept 11](#__RefHeading___Toc857_2097848850)

[2.3 Protobuf (Protocol Buffers) Concept 11](#__RefHeading___Toc855_2097848850)

[3 Non Real Time APIs using ZMQ and Protobuf 12](#__RefHeading___Toc1165_2097848850)

[3.1 Repl\_cmd 13](#__RefHeading___Toc1167_2097848850)

[3.1.1 Ecat\_Master\_cmd 13](#__RefHeading___Toc1169_2097848850)

[3.1.2 FOE\_Master (File Access over EtherCAT) 14](#__RefHeading___Toc1171_2097848850)

[3.1.2.1 Firmware Update Procedure 15](#__RefHeading___Toc1173_2097848850)

[3.1.3 Slave\_SDO\_info 15](#__RefHeading___Toc1175_2097848850)

[3.1.4 Slave\_SDO\_cmd 15](#__RefHeading___Toc1177_2097848850)

[3.1.4.1 Slave SDO cmd Procedure 16](#__RefHeading___Toc1179_2097848850)

[3.1.5 Flash\_cmd 17](#__RefHeading___Toc1181_2097848850)

[3.1.5.1 Flash cmd Procedure 17](#__RefHeading___Toc1183_2097848850)

[3.1.6 Ctrl\_cmd 18](#__RefHeading___Toc1185_2097848850)

[3.1.6.1 Ctrl\_cmd Procedure 19](#__RefHeading___Toc1187_2097848850)

[3.1.6.2 Gains 20](#__RefHeading___Toc1189_2097848850)

[3.1.6.3 Position Control 20](#__RefHeading___Toc1193_2097848850)

[3.1.6.4 Impedance Control 21](#__RefHeading___Toc1195_2097848850)

[3.1.7 Trajectory Cmd 21](#__RefHeading___Toc1197_2097848850)

[3.1.7.1 Homing 22](#__RefHeading___Toc1199_2097848850)

[3.1.7.2 Sine 22](#__RefHeading___Toc1201_2097848850)

[3.1.7.3 Smoother 23](#__RefHeading___Toc1203_2097848850)

[3.1.8 Trj\_queue\_cmd 23](#__RefHeading___Toc1205_2097848850)

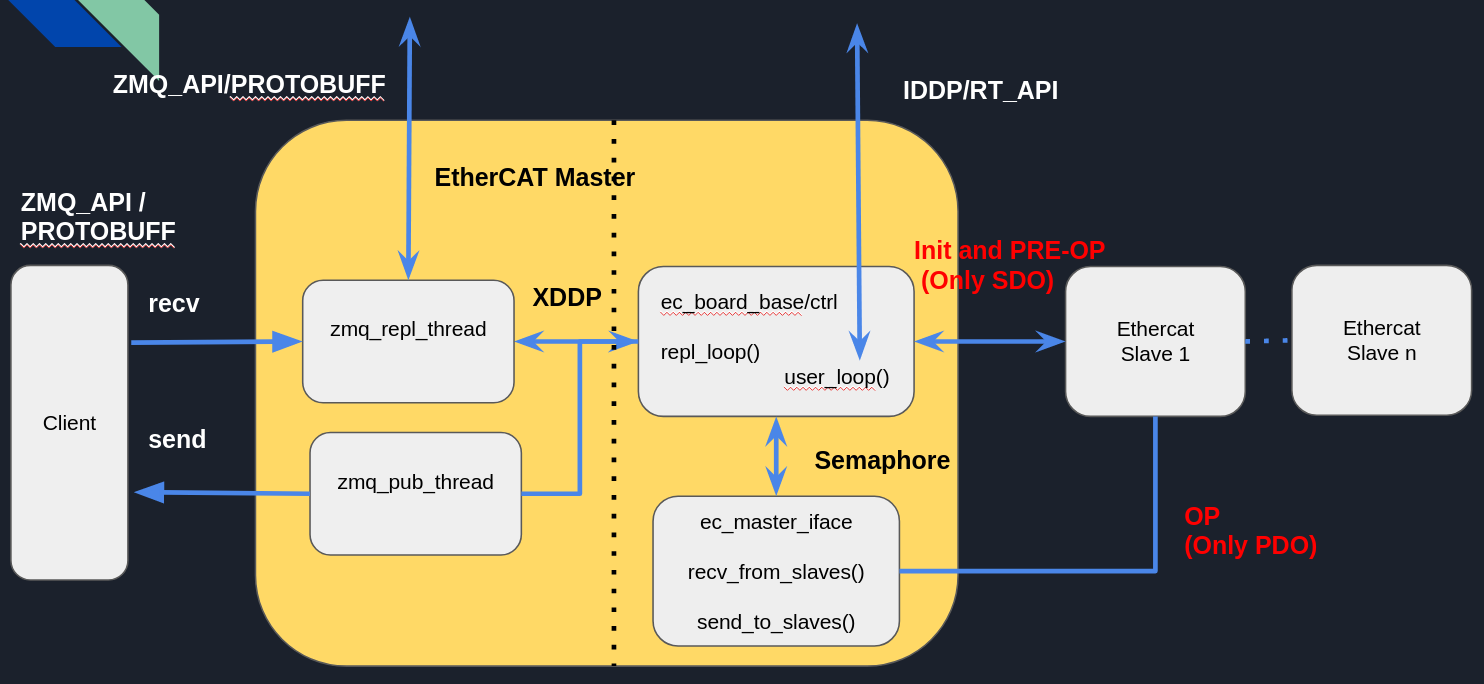
[3.2 Cmd\_reply 24](#__RefHeading___Toc1511_2097848850)

[3.3 Header 24](#__RefHeading___Toc1209_2097848850)

[4 Real Time APIs using IDDP Xenomai OS mechanism 25](#__RefHeading___Toc1191_2097848850)

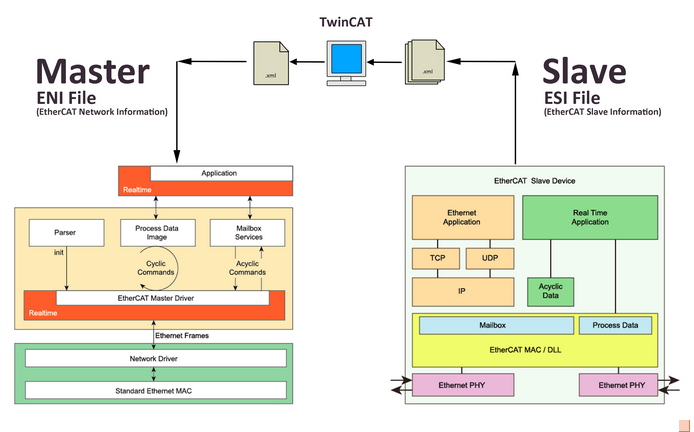
# 1 Scope

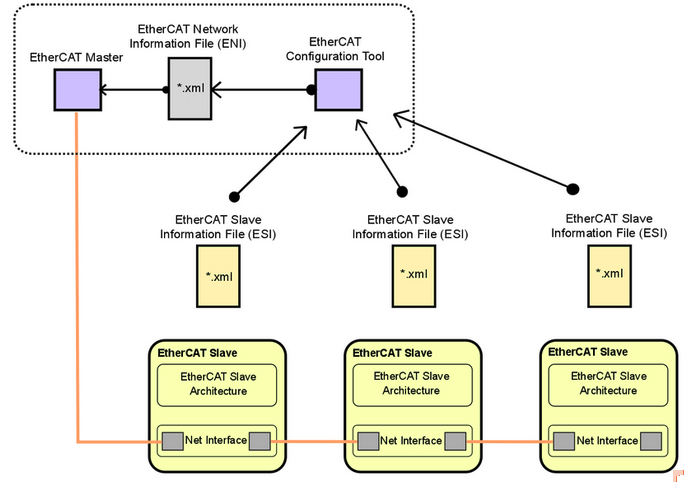
# **2 EtherCAT Master Design**



# 2.1 EtherCAT Protocol Concept

<https://www.ethercat.org/en/technology.html>

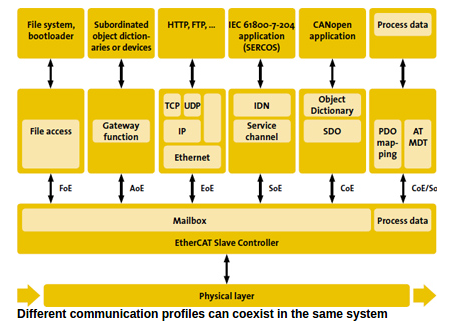




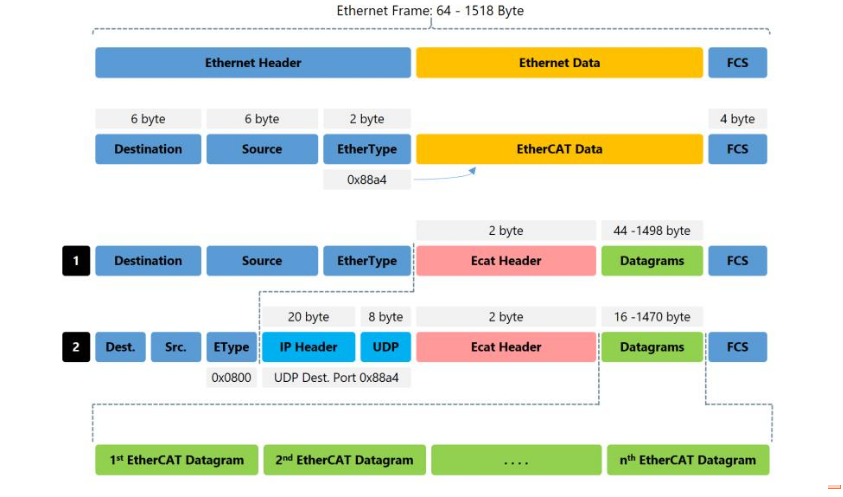
In order to configure and diagnose slave devices, it is possible to access the variables provided for the network with the help of acyclic communication. This is based on a reliable mailbox protocol with an auto-recover function for erroneous messages.

In order to support a wide variety of devices and application layers, the following EtherCAT communication profiles have been established:

* CAN application protocol over EtherCAT (CoE)
* Servo drive profile, according to IEC 61800-7-204 (SoE)
* Ethernet over EtherCAT (EoE)
* File Access over EtherCAT (FoE)
* Automation Device Protocol over EtherCAT (ADS over EtherCAT, AoE)

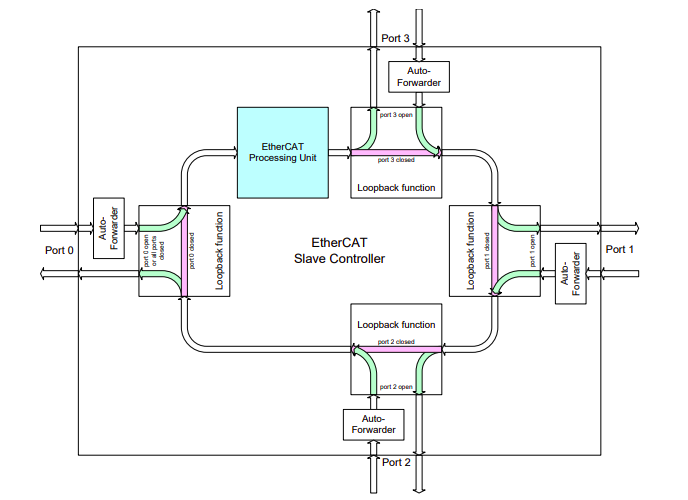


# 2.1.1 EtherCAT Frame and Slave Communication



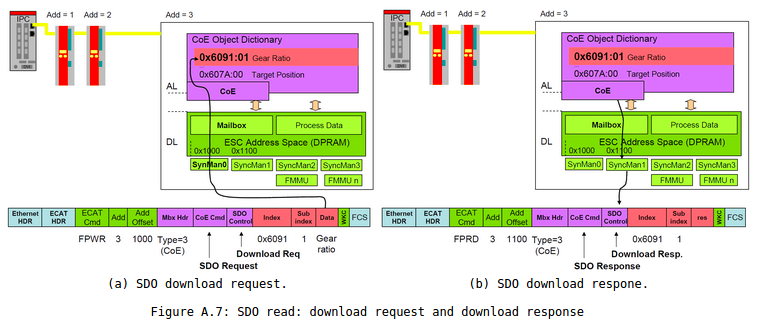
1) EtherCAT communication. 2) EtherCAT communication over TCP/UDP

Beckoff et1100.



# 2.1.2 Service Data Objects (SDO)

The Service Data Object (SDO) are used to access the Object Dictionary of a device. The requester of the OD access is called the Client and the EtherCAT device, whose OD is accessed and services the request, is called the Server. A Client request is always confirmed by a reply from the Server,



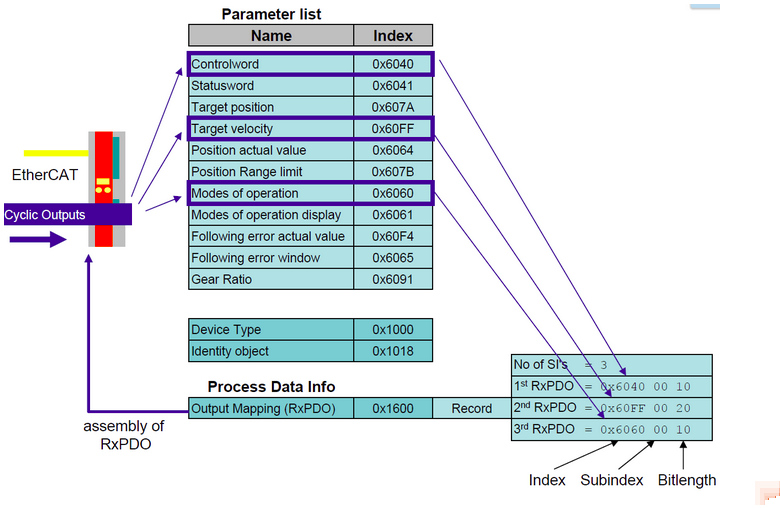
# 2.1.3 Process Data Objects(PDO)

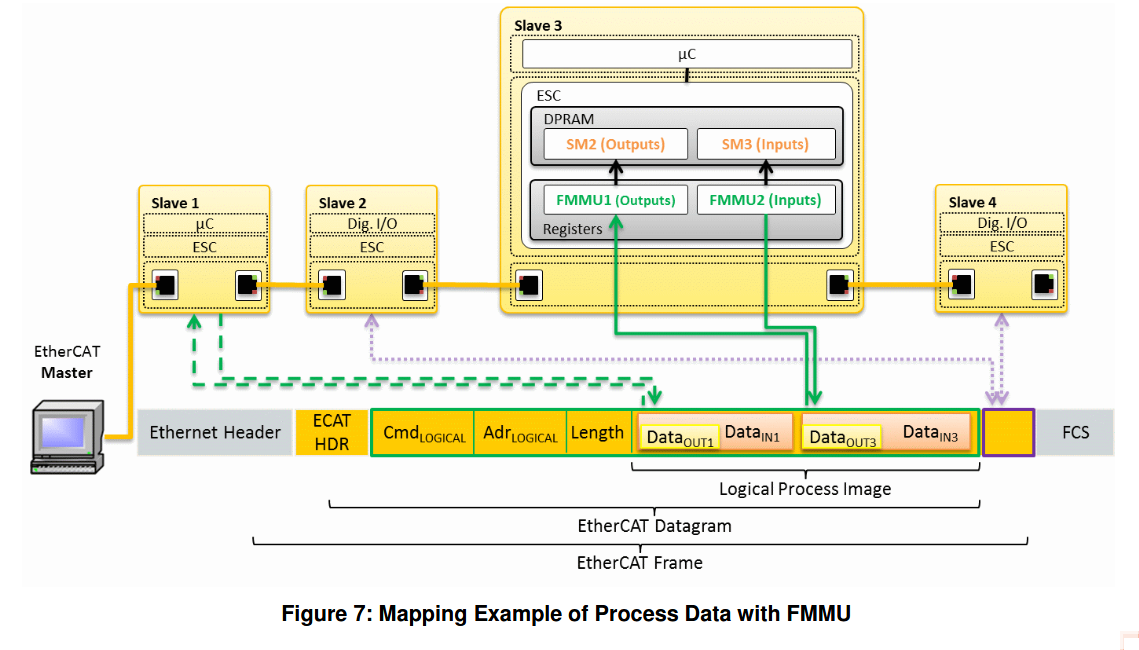
Process Data Objects are used to transfer real-time data among various nodes. Data are transferred from one (and only one) producer to one or more consumers.One PDO can contain multiple object dictionary entries. The objects within a PDO are configurable using the mapping (see section A.3) and the parameter object dictionary entries (fig. A.8). There is a maximum of 64 bits for PDO. There are two kinds of PDO:

* Transmit PDO: reads data from device
* Receive PDO: sends data to device

**PDO mapping**

To configure how many and which object dictionary entries have to be cyclically updated using PDO, the user has to send a sequence of SDO that "map" all the desired variables in the objects that would be transmitted using PDO. The PDO mapping procedure (fig. ) could be performed only when the slave is in pre-operational state. The objects mapped in the PDO input could be different from the objects of the PDO output.

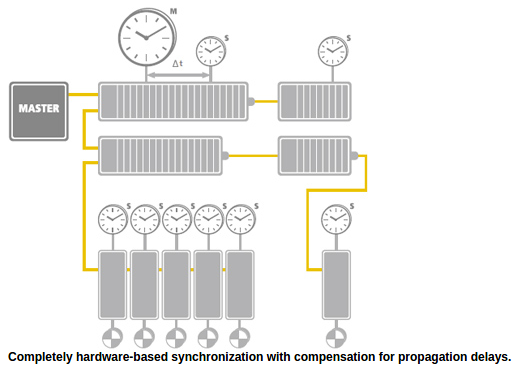




# 2.1.4 Distributed Clocks for High-Precision Synchronization

In applications with spatially distributed processes requiring simultaneous actions, exact synchronization is particularly important. For example, this is the case for applications in which multiple servo axes execute coordinated movements.

In contrast to completely synchronous communication, whose quality suffers immediately from communication errors, distributed synchronized clocks have a high degree of tolerance for jitter in the communication system. Therefore, the EtherCAT solution for synchronizing nodes is based on such distributed clocks (DC).



# 2.1.5 EtherCAT Master State Machine

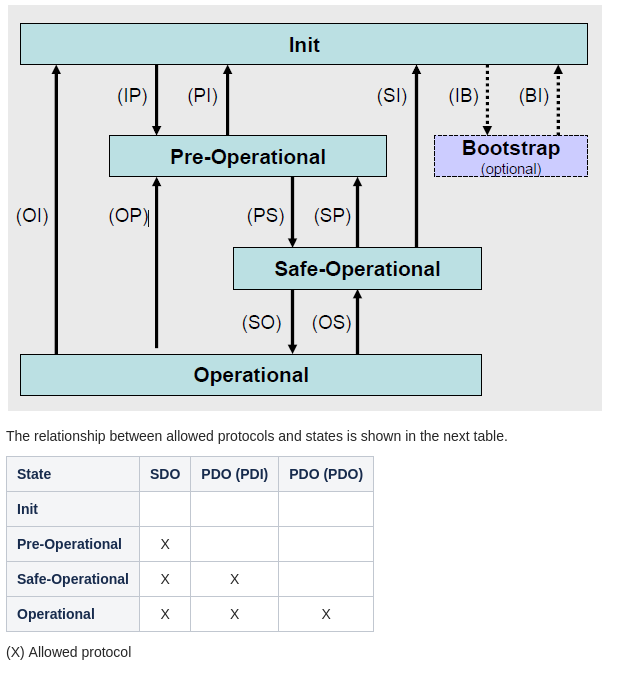
Before to work with EtherCAT is important to understand which communication protocols are used and where can be used.

As the Ingenia devices are based on CANopen over EtherCAT, two different communication protocols are used:

* **SDO - Service Data Object** (Mailbox communication). SDO allows a peer-to-peer communication between master and slave.
* **PDO - Process Data Object** (Process Data communication). PDO allows real time communication. There are two types depending on the direction:
  + *Process Data Input* (PDI). Slave to master.
  + *Process Data Output* (PDO). Master to slave.

Every EtherCAT device has an internal communication state machine that defines which kind of communication are allowed every time.

Following there is a diagram of all defined states:



# 2.1.6 Libraries

Simple Open EtherCAT Master or SOEM:

<https://openethercatsociety.github.io/doc/soem/index.html>

<https://openethercatsociety.github.io/doc/soem/index.html>

Simple Open EtherCAT Slave or SOES:

<https://openethercatsociety.github.io/doc/soes/index.html>

# 2.2 ZMQ or Zero MQ Concept

<https://zeromq.org/>

<https://zeromq.org/>

ZeroMQ (also spelled ØMQ, 0MQ or ZMQ) is a high-performance asynchronous messaging library, aimed at use in distributed or concurrent applications. It provides a message queue, but unlike message-oriented middleware, a ZeroMQ system can run without a dedicated message broker.

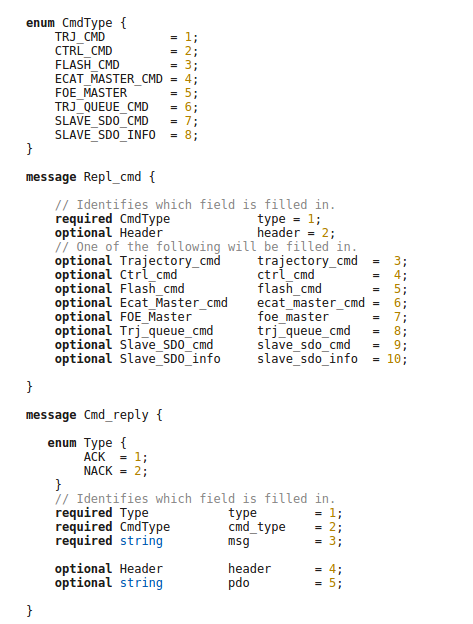
# 2.3 Protobuf (Protocol Buffers) Concept

<https://developers.google.com/protocol-buffers>

<https://developers.google.com/protocol-buffers>

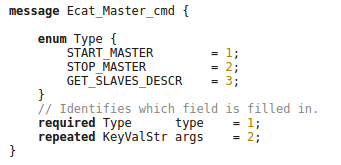
Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages. You can even update your data structure without breaking deployed programs that are compiled against the "old" format.

# **3 Non Real Time APIs using ZMQ and Protobuf**



# 3.1 Repl\_cmd

# 3.1.1 Ecat\_Master\_cmd

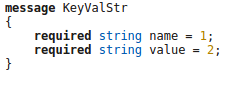


Type:

|  |  |
| --- | --- |
| **Name Command** | **Description** |
| **START\_MASTER** | start the EtherCAT Master (ec\_board base/ctrl ,ec\_master\_iface, zmq\_pub\_thread). |
| **STOP\_MASTER** | stop the EtherCAT Master (ec\_board base/ctrl ,ec\_master\_iface, zmq\_pub\_thread). |
| **GET SLAVES DESCR** | get the slaves informations and ids. |

|  |  |
| --- | --- |
| **GET\_SLAVES\_DESCR Information** | **Description** |
| **Topology** | Number Port actives (see EtherCAT Frame session). (1...4) **i.e 3** |
| **Active Ports** | Which ports (1...4) **i.e 2,3,4** |
| **Position** | Slave position (first, second, etc…). |
| **Esc\_Type** | value saved into ESC EEPROM that identify the slave type. |
| **Robot ID** | Slave ID. |
| **Model Type** |  |
| **Model Size** |  |
| **Model ID** |  |
| **Model Revision** |  |

KeyValstr:

****

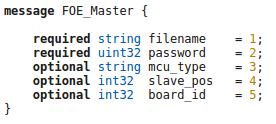
i.e

**For this command type the key value mechanism is used to change the YAML file used by EtherCAT master:**

({**'app\_mode'**:**'run\_mode'**,**'use\_ecat\_pos\_as\_id'**:**'false'**})

({**'app\_mode'**:**'config\_mode'**,**'use\_ecat\_pos\_as\_id'**:**'true'**})

# 3.1.2 FOE\_Master (File Access over EtherCAT)



i.e

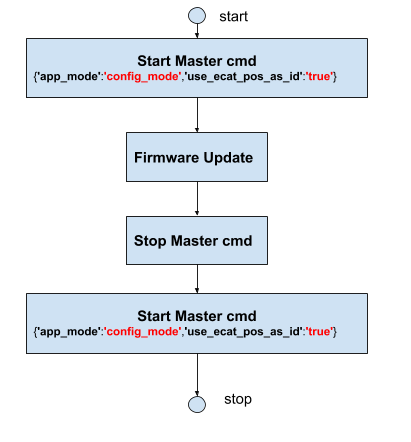
**mcu="c28";**

**filename="cent\_AC\_c28.bin";**

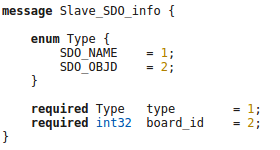
**password=”0xDAD0”**

**PreCondition:** EtherCAT master should be in Pre-operation state.

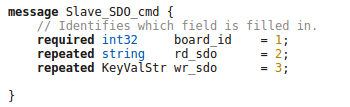
# 3.1.2.1 Firmware Update Procedure



# 3.1.3 Slave\_SDO\_info

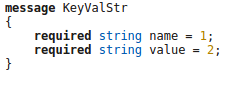


# 3.1.4 Slave\_SDO\_cmd



**i.e** 'slave\_sdo\_cmd': {'board\_id': 12, '**rd\_sdo**': **['Joint\_robot\_id', 'm3\_fw\_ver', 'c28\_fw\_ver', 'Min\_pos', 'Max\_pos']**, **'wr\_sdo'**: []}, 'type': 'SLAVE\_SDO\_CMD'}

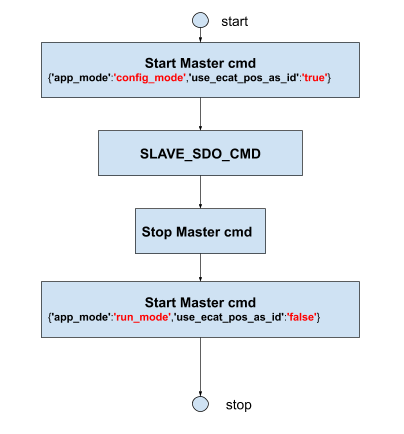
KeyValstr:

****

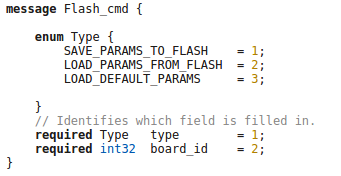
i.e

**For this command type the key value mechanism is used to write the values of SDO fields.**

# 3.1.4.1 Slave SDO cmd Procedure

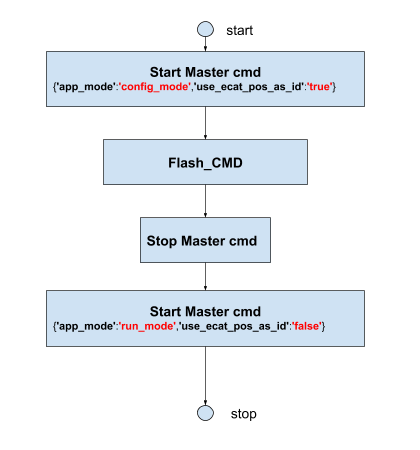
****

# 3.1.5 Flash\_cmd

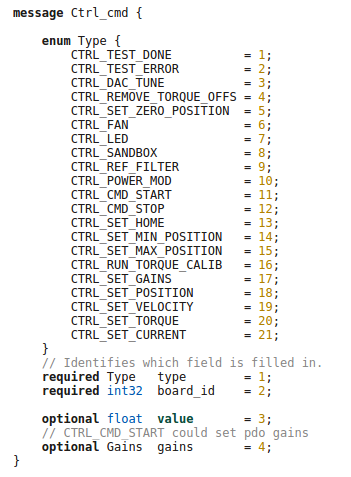


|  |  |
| --- | --- |
| **Name Command** | **Description** |
| **SAVE\_PARAMS\_TO\_FLASH** | Save parameters to the flash. |
| **LOAD\_PARAMS\_FROM\_FLASH** | Load parameters from the flash. |
| **LOAD\_DEFAULT\_PARAMS** | Load default parameters. |

# 3.1.5.1 Flash cmd Procedure

****

# 3.1.6 Ctrl\_cmd

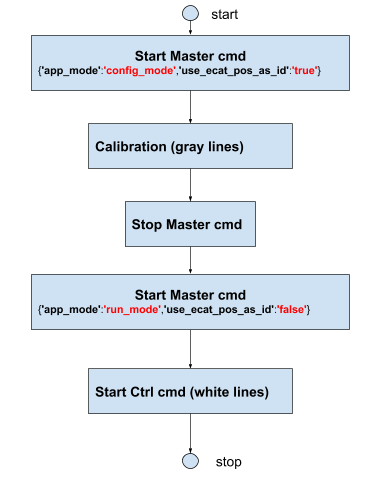


Type:

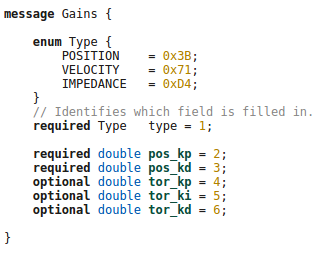
|  |  |
| --- | --- |
| **Name Command** | **Description/Value** |
| **CTRL\_TEST DONE** | NULL (echo OK of the command). |
| **CTRL\_TEST ERROR** | NULL (echo NOK of the command). |
| **CTRL\_DAC\_TUNE** | NULL (only for force torque sensor and set analogical offset).  Note: Reply in process during the calibration. |
| **CTRL\_REMOVE\_TORQUE\_OFFS** | ON/OFF remove torque offset. |
| **CTRL\_SET\_ZERO\_POSITION** | set “zero” position of joints. |
| **CTRL\_FAN** | ON/OFF of FANs. |
| **CTRL\_LED** | ON/OFF of LEDs. |
| **CTRL\_SANDBOX** | ON/OFF of mechanical end stop. |
| **CTRL\_REF\_FILTER** | ON/OFF filter for position reference. |
| **CTRL\_POWER\_MOD** | ON/OFF of power modulator. |
| **CTRL\_SET\_MIN\_POSITION** | set min position limit (rad). |
| **CTRL\_SET\_MAX\_POSITION** | set max position limit (rad). |
| **CTRL\_RUN\_TORQUE\_CALIB** | NULL (torque calibration).  Note: Reply in process during the calibration. |
| **CTRL\_CMD\_START** | start control command. (i.e start trajectory cmd). |
| **CTRL\_CMD\_STOP** | stop control command. |
| **CTRL\_SET\_HOME** | set homing position. |
| **CTRL\_SET\_GAINS** | set gains. |
| **CTRL\_SET\_POSITION** | set position (rad). |
| **CTRL\_SET\_VELOCITY** | set velocity (rad/s). |
| **CTRL\_SET\_TORQUE** | set torque (Nm). |
| **CTRL\_SET\_CURRENT** | set current (A). |

**Note:** The gray lines are commands used for the calibration during the config mode.

# 3.1.6.1 Ctrl\_cmd Procedure

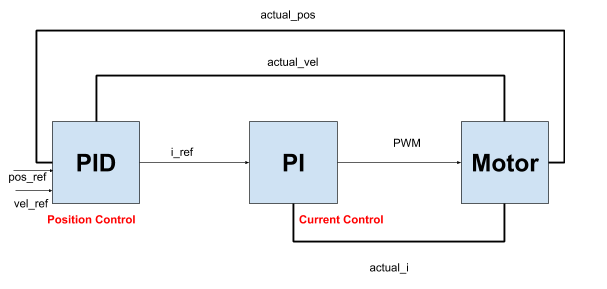
****

# 3.1.6.2 Gains

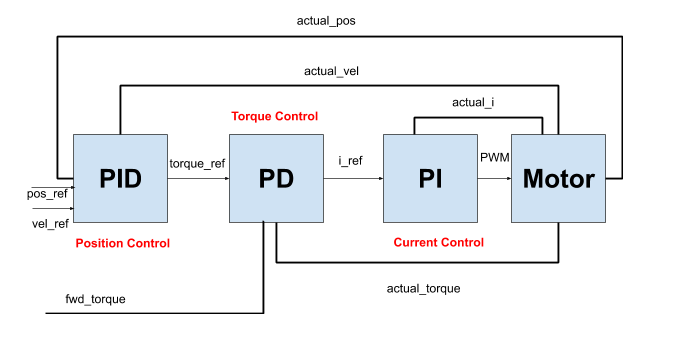


|  |  |
| --- | --- |
| **Name Command** | **Description/Value** |
| **pos\_kp** | Proportional gain of Position Control. |
| **tor\_ki (pos\_ki)** | Integral gain of Position Control. |
| **pos\_kd** | Derivative gain of Position Control. |
| **tor\_kp** | Proportional gain of Torque Control. |
| **tor\_kd** | Derivative gain of Position Control. |

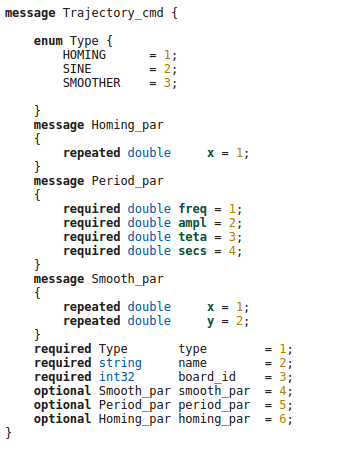
# 3.1.6.3 Position Control



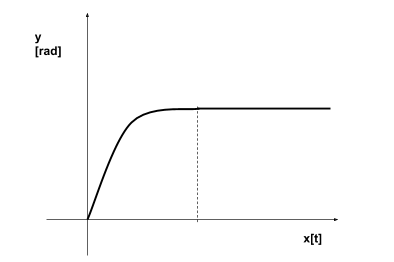
# 3.1.6.4 Impedance Control



# 3.1.7 Trajectory Cmd

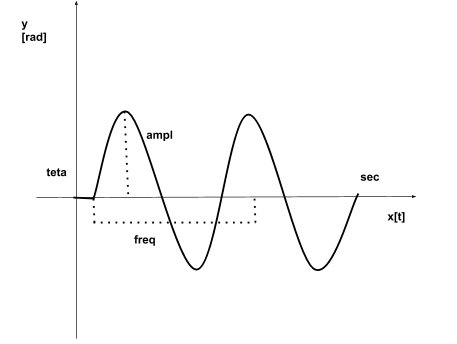


# 3.1.7.1 Homing



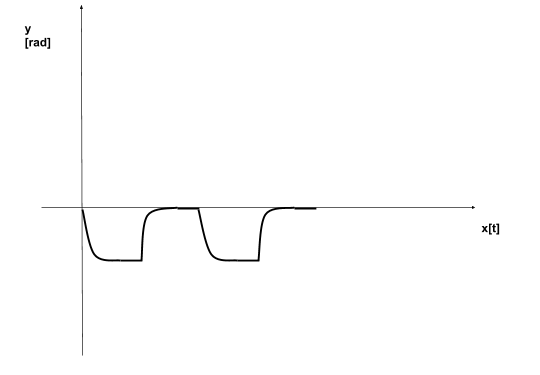
**i.e x=[0,3] y=0.6.**

# 3.1.7.2 Sine



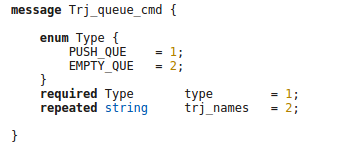
**i.e freq= 0.2, ampl= 1.0, teta= 0, secs=500.**

# 3.1.7.3 Smoother



**i.e x=[0,1,2,3,4]sec ,y=[0,-0.3,0,-0.3,0].**

# 3.1.8 Trj\_queue\_cmd



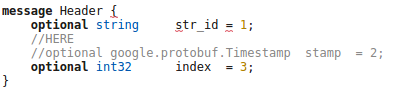
|  |  |
| --- | --- |
| **Name Command** | **Description** |
| **PUSH\_QUE** | Push the trajectory for the master. |
| **EMPTY\_QUE** | Clear trajectory. |

# 3.2 Cmd\_reply

# 

# 

# 3.3 Header



Note: For the Repl\_cmd and Cmd\_reply it’s possible to use the header to verify the communication.

# 4 Real Time APIs using IDDP Xenomai OS mechanism