CANopen
Implementation Guide
for
"SAXOPHONE, CLARINET"
Servo Drives

Rev 01/02

Revision	Description
01/02	Modified Commands: • Correction in XC,XQ commands object 0x2074. Manufacture new objects: • 0x2031 – Binary Upload and Download object • 0x2041 – The latched timer • 0x2051 – Unsynchronized erupt instruction • 0x2052 – Unsynchronized homing done message. • 0x208A – Synchronized Begin on Time object. • 0x2090 – Download Firmware. • 0x2F0A – System Internal 32 bit timer. • 0x2F11 – PVT Head Pointer. • 0x2F12 – PVT Tail Pointer.
05/01	Manufacture new objects: • 0x206A – Error Code value according to EC command. • 0x2F00 – IA array. • 0x2F0B – Status Object • DS402 objects: • 0x6402 - Motor Type • 0x6403 – Motor Catalog Number • 0x6404 – Motor manufactor • 0x6502 – Supported dirver modes • 0x6504 – Driver manufacturer • 0x6505 - Driver manufacturer web site. Modified Commands: • Object 0x1400 is read write.
11/00	Modified Commands: • Correction in SYNC and TIME STAMP section • Correction in NMT emcy code. • Correction in PDO mapping section • Last byte in UploadProgram (0x2070) is 0x08 • Emergency code 0x3b to 0x34 • RPDO2 was corrected for default value and mapped object Manufacture new objects:

	• 0x2004 – Fast ECAM entry. • 0x20A1 – Main Position Error. • 0x2B01 – Velocity Factor. • 0x2B02 – Current Factor. • 0x6071 – Torque Command. • 0x6078 - Active Current.

Table of Contents:

List of Tables:

1 Purpose

This document details the implementation of CAN communication with the ELMO line of digital servo drives, more specifically for the Clarinet and Saxophone models.
This implementation guide is designed to provide a familiarity with the products. It is not a replacement for a thorough reading and understanding of the product documentation.

2 Relevant Documents

Num	Name	Author	Source
1	CAN Implementation Guidelines.	Gruhler G. & Dreier B.	STA Reutlingen
2	CAL Based Communication Profile for Industrial Systems, CiA Standard DS-301, Ver 4.01, 2000.		CiA
3	PVT motion – an application brief.		Elmo Motion Control
4	PT motion – an application brief.		Elmo Motion Control
5	Big and little Endians.		Http://www.carmel.com/pmon/pmon5/html/endian.htm
6	Metronom Command reference.		Elmo Motion Control
7	CMS Protocol Specification, CiA standard DS-202-2.		CiA
8	Metronome Software manual.		Elmo Motion Control

3 Glossary

Item	Description
CAL	CAN Application Layer.
CAN Client or CAN Master	The node in the CAN network that commands the other nodes.
CAN Server or CAN Slave	A commanded node in the CAN network. The Elmo servo drives are CAN servers.
CMS	CAN Message Specification.
COB	Communication object – A CAN message.
ID	Identifier. The ID is the name by which a CAN device is addressed.
COB-ID	A binary bit field that includes the ID of the server with which the master talks, and the type of the COB.

4 Introduction

The Elmo line of digital servo drives support two types of serial communication.
1. RS232
2. RS485 (Option)
3. CANopen – Full CiA DS-301 compliance.

The servo drive can simultaneously communicate using the CAN and the RS232 communication lines. Both the RS232 line and the CAN line are always open for communication.
The parameters of the communication need to be set, as explained later in this document.
The following table compares the two supported communication methods.

Property	CANopen	RS-232/RS485
Baud rate	10000 – 1000000.	38400 (RS232), 19200(RS485)
Interpreter method	Binary.	ASCII.
Fast referencing	Yes for PVT ant PT motions.	No.
Multiple servo drives	Yes.	RS485 only (Future).
Multiple servo drive synchronization	Yes.	No.
Special equipment required	CAN communication interface (available as an Add-In ISA or PCMCIA card for PC computers) with appropriate software.	Direct connection to the serial port of a PC computer.
Standardized	Compliance with the CiA DS-301 standard.	No standard.
Ease of use	Basic capabilities are included in the Composer program from ELMO.	Immediate – just type the command using HyperTerminal or an equivalent terminal software.

Table 4-1: Communication Types

RS232 operation is fast and simple, requiring no detailed understanding of communication processes.
CANopen communication achieves higher rates and is able to support the following advanced functions:
- High speed on line reference generation, required to support complex motions.
- Binary interpretation, which maximizes servo drive command throughput, by eliminating servo drive software overhead.
- Multiple servo drive applications.

To benefit from the advantages of CAN communication and the CiA DS-301 CANopen standard, the user must have a good understanding of the basic programming and timing issues of a CANopen network.

Note:
In the rest of this document, the terms “transmit” and “receive” refer to the servo drive. “Received” data is sent from the commanding equipment to the servo drive, and “Transmitted” data is sent from the servo drive.

5 What is CANopen?

This section explains some of the CANopen communication issues that are most relevant to Elmo servo drives. This is an overview and more detail is available in [1] and [2].

5.1 Physical Layer

CAN is a serial communication standard. It requires that the data to be transferred be coded as electrical pulses on two wire communication lines. The device that handles the CAN physical layer is called the CAN controller. The device that transmits data over the CAN lines is called the CAN transceiver.
Elmo servo drives use the Intel 82527 CAN controller. The Intel 82527 CAN controller has 16 message buffers, of which Elmo controllers use several. Each message buffer is actually an independent communication device which deals with its unique type of communication object (message). The communication objects used will be explained later in this document.
Each message buffer stores up to 8 bytes of message body, in addition to the overhead data. The overhead data defines the message type, attributes and address.

5.2 Standard Vs. Extended Addressing

The overhead attached to each CAN message includes an arbitration field. The arbitration field (COB-ID) defines the type of data sent and the address.
CAN ver. 2.0A supports 11 arbitration bits. Of these 11 bits, the 7 least significant define the address and the 4 most significant bits define the type of message sent. **Note:** that only 16 message types are supported.
CAN ver. 2 .0B supports 29 arbitration bits. Of these, the seven least significant define the addressee and 21 bits define the message type.
CAN communication is prioritized – messages with higher priority are transmitted first. The arbitration field determines the priority of a message. The lower the number in the arbitration field, the higher the message priority . The ID 0 grants the highest available priority.
Elmo products only support the CAN ver. 2.0A addressing method.This means that bit 29 in the arbitration field MUST BE zero, with the 11 ID bits transmitted immediately after that.
A setup parameter (PP[15]) selects the CAN object identification version to use. This parameter is presently reserved , and must be set to zero. Future software versions may support 2.0B object identification mode.

5.3 Client and Servers Relation

A CAN Client, or Master, is a node that asks other nodes to respond to its command.
A CAN Server, or Slave, responds to the commands issued by the CAN master.
The CAN protocol permits multiple master networks as well as single master networks.
Elmo servo drives assume a single-master network arrangement. The servo drives are the servers and the machine controller or PLC is the master.
Every servo drive has a unique ID in the range 1-127. The network master does not need an ID.
A servo drive never sends un-asked-for messages, other then Emergencies.
A servo drive responds only to messages addressed to its ID, or to broadcast messages, which have the ID of 0. All the messages sent by a servo drive are marked with its own ID.

Please **Note:** that if two servo drives have the same ID, then the CAN network may freeze until reset.

5.4 Inhibit Times

The inhibit time for a given message type, is the minimal time that must elapse from the transmission of a message of that type, until the next time transmission of this message is allowed again.
Inhibit times are meant to prevent high priority messages from flooding the bus, eliminating the ability to service messages of lower priority.
The inhibit times of Elmo servo drives are not programmable. The inhibit times for the client to eliminate flooding of the servo drive, are 0.5msec for PDO[1], 1 msec for SDO. Inhibit-times for NMT messages are not defined as NMT messages are considered infrequent.

5.5 Object Dictionary

An object dictionary (*OD*) is a naming system that gives a unique identifier to every data item that needs to be communicated over the CAN bus.
A data item that is referenced in the dictionary is called an *object*.
An object is identified by an index and if it is a complex object, also by a sub-index.
A CANopen client can manipulate an object of a CANopen server by referring to its identifier, according to the access permission of the object (The access permission to an object may be read only, write only, or read-write).
CiA DS-301 requires that a set of mandatory data items be supported by any CANopen devices. Other OD items are predefined by CiA DS-301 to have fixed identifiers, if supported. There is also room in the OD for manufacturer specific items.

[1] See definitions of COB types in ***Communication Objects***.

5.6 Communication Objects

The combination of useful data bytes with their accompanied overhead is called a *Communication object (COB)*. Elmo servo drives use the following COB types:

COB type	Explanation
SDO (Service Data Object)	SDO messages are used to manipulate OD objects using their identifier. The server receive SDO specifies in its message body which object is to deal with. SDO messages can be chained to form a "domain transfer". Domain transfers are useful for large data items such as long strings. Domain transfers pay with speed for safety. In the process of message download, for every downloaded data segment a full sized data segment is uploaded for verification. Similarly, in the process of message upload, for every uploaded data segment a full sized data segment is downloaded for verification. This takes its time, since the CAN bus is half-duplex.
PDO (Process Data Object)	PDO messages are used to manipulate OD objects without explicit reference to their identifier, eliminating the need to communicate the object identifier. This is possible if there is a-priory (?) convention about the OD item referred. A convention that relates a PDO to an OD object is called PDO mapping. PDO mappings by themselves are OD objects, so they may be defined and manipulated using an SDO.
EMCY (Emergency)	Emergency messages are used by the servo drives to warn of an exception. The EMCY is the only COB type that a servo drive may transmit without being explicitly asked. EMCY objects are like "interrupts" from the servo drive – they eliminate the need to poll the servo drive continuously for its status.
NMT (Network management)	NMT objects are used by a client to initialize a servo drive as a server.

Table 5-1: Communicatiob Objects

The CANopen CiA DS-301 standard also defines other objects that Elmo servo drives do not implement, such as DBT (Distribution). All these COB types are not mandatory for compliance with the CiA DS-301 standard.
The type of a COB is resolved by the arbitration field of the message, and therefore determines its priority. The relation between bits 8-11 of the arbitration field (COB ID) and the type of the COB is presented in the table below.

COB type	Bits 8-11 of COB ID	ID range
NMT	0000	0-127 (0-7fh)
SYNC	0001	128 (80h)
EMERGENCY	0001	129-255 (81h-ffh)
Unused (Reserved for PDO1 – Transmit)	0011	385-511 (181h-1ffh)
PDO1 – Receive	0100	513-639 (201h-27fh)
PDO2 Transmit	0101	641-767 (281h-2ffh)
PDO2-Receive	0110	769-895 (301h-37fh)
PDO3 – Transmit	0111	897-1023 (381h – 3ffh)
PDO3 – Receive	1000	1025-1151 (401h – 47fh)
PDO 4 – Transmit	1001	1153-1279 (481h – 4ffh)
PDO 4 -Receive	1010	1281-1407 (501h – 57fh)
SDO-Transmit	1011	1409-1535 (581h-5ffh)
SDO-Receive	1100	1537-1663 (601h-67fh)
Error Control (node guarding)	1110	1793-1919 (701h-77fh)

Table 5-2: COB Types

5.7 Example

The COB ID of PDO1, when received by node #2, will be binary 0100 0000010 which is decimal 514 or 202 hexadecimal.

The IDs of the servo drives are set in the range 1-127.

6 Data Type

Elmo CAN controller's support the following data types as shown in /2/ Table 10-4:

Index	Object	Name
0001	DEFTYPE	Boolean
0002	DEFTYPE	Integer8
0003	DEFTYPE	Integer16
0004	DEFTYPE	Integer32
0005	DEFTYPE	Unsigned8
0006	DEFTYPE	Unsigned16
0007	DEFTYPE	Unsigned32
0008	DEFTYPE	Floating Point (Float)
0009	DEFTYPE	Visible String
0020	DEFSTRUCT	PDO CommPar
0021	DEFSTRUCT	PDO Mapping
0022	DEFSTRUCT	SDO Parameter
0060	DEFTYPE	PVT DataPar
0061	DEFTYPE	PT DataPar
0062	DEFTYPE	Binary Interpreter Inquiry.
0063	DEFTYPE	Binary interpreter Command

Table 6-1: Data Types

6.1 Device Specific Data Types

Device Profile Specific Types:

6.1.1 PVT DataPar Object 0x60

MSB LSB

Time(Unsigned8)	Velocity (Singed24)	Position (Signed32 bits)

6.1.2 PT DataPar Object 0x61

MSB LSB

Position 2 (Signed32 bit)	Position 1 (Signed 32 bit)

6.1.3 Binary Interpreter Inquiry Object 0x62

MSB 7	6	5	4	3	2	1	LSB 0
				Attribute high	Attribute low	Letter high	Letter low

For more details about getting refer to ***Binary Interpreter Commands***.

6.1.4 Binary Interpreter Command Object 0x63

MSB 7	6	5	4	3	2	1	LSB 0
Data high	data	data	data low	Attribute high	Attribute low	Letter high	Letter low

For more details about setting values please refer to ***Binary Interpreter Commands***.

7 Representation of Numbers

CAN communication may also be used to send numerical data. The numerical data is stored in binary form. Integers are stored by their binary representation and floating-point numbers are stored by their IEEE representation. Elmo servo drives support 3 types of data – short integer (2 bytes), long integer (4 bytes) or floating point numbers (4 bytes). These multiple-byte numbers are stored in the CAN messages, according to CAN standards, using the "Little Endian" (Intel type) convention. In this convention, the number is "inverted" before storage – the most significant byte of the number receives the least address, while the least significant byte receives the highest address. A good explanation of big and little endians is given in [5].
Example, suppose we want to construct the following eight-bytes CAN message.

Bytes 0-1	0x1234
Bytes 2-3	0x5678
Bytes 4-7	0x9abcdef0

The data field of the CAN message will be

Byte	Contents
0	0x34
1	0x12
2	0x78
3	0x56
4	0xf0
5	0xde
6	0xbc
7	0x9a

Note:
The Elmo servo drives support remote frames only for Node Guarding.

8 The Object Dictionary

The object dictionary supports:
- Objects that are mandatory by the CiA DS-301 standard
- Elmo specific objects

The object dictionary for Elmo servo drives is defined below. All the OD items may be accessed via SDO.

Name	Index, Subindex	Comment	Access	Mappable
CAN Controller Type.	0x1000	Constant value = 0x20191, stands for Intel 82527.	R	N
Error reg.	0x1001	Contains error information. Any write to object 0x1001 resets the communication error register.	R	N
Manufacturer Status register.	0x1002	Returns the status similar to the SR command in [6].	R	N
Pre defined error field.	0x1003	Returns previous emergency History.	R	N
Number of supported PDO.	0x1004	Supported PDO are returned for synchronous and asynchronous PDO with Tx and Rx difference.	R	N
COB ID for SYNC message.	0x1005	32 bit DWORD, Pre defined.	R	N
Communication cycle period.	0x1006	The spacing, in µsec, between consecutive SYNC signals. **This parameter is included for compatibility with the standard OD, but it is ignored.**	R/W	N
Synchronize window length.	0x1007	Contains the length of time for synchronize messages in µsec. **This parameter is included for compatibility with the standard OD, but it is ignored.**	R/W	N
Manufacturer device name.	0x1008	String, returns Saxophone, Clarinet, Melody, etc.	R	N
Hardware version.	0x1009	A string that conveys the information in WS[30]. Please refer [6].	R	N
Software version.	0x100a	String, returns the value of the VR command, please refer [6].	R	N
Node ID.	0x100b		R	N
Guard Time.	0x100c		R/W	N
Life time factor.	0x100d	Life time factor.	R/W	N
Node guarding identifier.	0x100e	Always 700H + Node ID.	R	N
Number of SDO supported.	0x100f	Always 0x00000001 for single server SDO.	R	N

Name	Index, Subindex	Comment	Access	Mappable
Store parameters.	0x1010	Stores parameters in FLASH. Please Refer *Object 1010H – Save Parameters*.	R/W	N
Restore parameters.	0x1011	Restore parameters from FLASH. Please Refer to *Object 1011H – Restore Parameters*.	R/W	N
COB-ID for time stamp message.	0x1012	Specified the COB ID of time stamp message.	R	N
High resolution time stamp.	0x1013	Set the value of client time stamp according to the time stamp protocol specified in /2/.7.1.	W	Yes Transmit Sync
SDO 1 Server.	0x1200	SDO 1 server parameter.	R	N
PDO 1 Rx Comm.	0x1400	PDO 1 receive communication parameter.	R/W	N
PDO 2 Rx Comm.	0x1401	PDO 2 receive communication parameter.	R	N
PDO 3 Rx Comm.	0x1402	PDO 3 receive communication parameter.	R	N
PDO 3 Rx Comm.	0x1403	PDO 4 receive communication parameter.	R	N
PDO 1 Rx Map.	0x1600	PDO 1 receive mapping parameter.	R/W	N
PDO 2 Rx Map.	0x1601	PDO 2 receive mapping parameter.	R	N
PDO 1 Tx Comm.	0x1800	PDO 1 transmit communication parameter.	R/W	N
PDO 2 Tx Comm.	0x1801	PDO 2 transmit communication parameter.	R	N
PDO 3 Tx Comm.	0x1802	PDO 3 transmit communication parameter.	R/W	N
PDO 4 Tx Comm.	0x1803	PDO 4 transmit communication parameter.	R/W	N
PDO 1 Tx Map.	0x1A00	PDO 1 transmit mapping parameter.	R/W	N
PDO 2 Tx Map.	0x1A01	PDO 2 transmit mapping parameter.	R	N
PDO 3 Tx Map.	0x1A02	PDO 3 transmit mapping parameter.	R/W	N
PDO 4 Tx Map.	0x1A03	PDO 4 transmit mapping parameter.	R/W	N
PVT data.	0x2001	Bytes describing the PVT command as explained in [3].	W	Yes Recived Unsync
PT data.	0x2002	Bytes describing the PT command as explained in [4].	W	Yes Received Unsync
Reserved.	0x2003	Reserved for future real time positioning modes.		
ECAM data	0x2004	Fast, auto increment entry to the ECAM table.	W	Yes Received Unsync

Name	Index, Subindex	Comment	Access	Mappable
Binary Interpreter input.	0x2012	PDO 2 receive that is mapped to this object and is the only one for this purpose.	W	Yes Static mapping
Binary interpreter output.	0x2013	PDO 2 transmit that is mapped to this object and is the only one for this purpose.	R	Yes Static mapping
Recorded data output.	0x2030	Contains the recorded data according to request in RC binary command.	R	N
FLASH Upload	0x2031	Used for upload and Download binary process	R	N
Coordinate system ID.	0x2040	Identifier used to address a synchronized command.	R/W	N
Latched clock.	0x2041	The microsecond clock, as latched upon the last sync. This clock is modulo 65536.	R	Yes Transmit Sync
MS Event trigger	0x2050	Motion done event object. When being mapped this object is transmitted upon motion done.	R	Yes Transmit Unsync
EI Event trigger	0x2051	Erupt Instruction event object. Can be used to transmit PDO at any event or location in User Program.		Yes Transmit Unsync
HM Event trigger	0x2052	Homing done event trigger. Used to mark that a main homing process was completed.		Yes Transmit Unsync
Reference vector.	0x2060	Used only by the Wizard in the process of system tuning.	R/W	N
Program download & list.	0x2070	Download user program. Upload user program list (refer to [8]).	R/W	N
Code status/Single Code.	0x2071	Upload similar to CS command. Download similar to SC command. Uploads the current line from the user program. Download performs the current command in user program and advance to the next command. (refer to [8]).	R/W	N
Program exception message.	0x2072	Similar to MZ command. This object contains messages from user program. Messages can be run time errors or result of MG command (refer to [8])..	R	N
Program compiler .	0x2073	Download, performs user program compilation. Upload, reads a listing of compiler	R/W	N

Name	Index, Subindex	Comment	Access	Mappable
		errors after compilation.		
Program execution.	0x2074	Execute user program. Similar to XQ command. Please refer to .	W	N
CPU Dump.	0x2080	Similar to CD command. Object contains the status of the CPU.	R	N
CAN controller Dump	0x2081	Dumping the CAN controller (Intel 82527)	R	Yes Transmit Sync
Begin On Time	0x208A	Used to start a synchronized motion according to the internal free running timer.		
Firmware downloading.	0x2090	Similar to the DF command. Applicable only if working firmware is installed. Will not work after a previous DF failure – in that case DF must be applied by the RS232 communication. Please read section on firmware downloading using CAN.	W	N
Firmware downloading status.	0x2091	Please read section on firmware downloading using CAN.	R	N
Position actual auxiliary value.	0x20A0	Actual position as taken from the auxiliary sensor input. (PY).	R	Yes Transmit Sync
Position Error	0x20A1	The position error as calculated from command and actual position value. (PE)	R	Yes Transmit Sync
Velocity Factor	0x2B01	The factor between the internal representative of speed to cnt/sec.	R	N
Current Factor	0x2B02	The tactor btween the internal representative of current unit to mA	R	N
Integer Array	0x2F00	General purpose array	R	Yes Transmit Sync
Free running Timer	0x2F0A	The system 32 bits free running timer.	R	Yes Transmit Sync
Status Object	0x2F0B	General status information	R	Yes Transmit Sync
Abort connection option code	0x6007	Function to perform on node guarding event.	R/W	N
Position demand value.	0x6062	Output of the profiler. Position command.	R	Yes Transmit Sync

Name	Index, Subindex	Comment	Access	Mappable
Position actual value.	0x6064	Actual position as taken from the position sensor (PX).	R	Yes Transmit Sync
Velociy sensor actual value.	0x6069	Actual velocity as calulated from the position sensor (VX).	R	Yes Transmit Sync
Velocity actual value.	0x606C	Velocity command as an input to the velocity controller.	R	Yes Transmit Sync
Torque Command	0x6071		R	Yes Transmit Sync
Motor actual current	0x6078	Motor current (IM,IQ)	R	Yes Transmit Sync
Motor Type	0x6402		Rw	N
Motor Catalog Number	0x6403	32 characters	Rw	N
Motor Manufacture	0x6404	32 characters	Rw	N
Supported Driver Mode	0x6502		R	N
Driver Manufacture	0x6504		R	N
Driver Manufatcure Web Site	0x6505		R	N

Table 8-1: Object Dictionary

9 Service Data Objects (SDO)

A single transmit server SDO and a single receive server SDO are used.
According to CiA definitions and priority allocations, no more than single transmit and receive SDOs are available if extended addressing (29 bit) is not supported.
The SDO used by the system is COB 581h-6ffh for transmit and 600h-67fh for receive.

Care should be exercised when using SDOs:
- An SDO has lower priority then a PDO
- An SDO session is not complete until confirmed.

For example, if an SDO is used to change a PDO mapping, then:
- Issue the SDO only after the last session using this PDO is completed
- Don't use the newly mapped PDO until the SDO mapping change is confirmed.

SDOs implement the CMS multiplexed domain protocols.
It is strongly recommended that Section 6 of /7/ be thoroughly read and understood before using SDOs.

Notes:
1. An SDO data exchange requires that any client message is backed by one and only one server message.
2. An SDO caries a toggle bit. In a domain transfer, every consecutive message varies the toggle bit, so that the loss of a single message can be tracked.
3. An SDO transfer can be broken using the special "Abort Domain Transfer" message.
4. An SDO message carries at most 7 bytes of data. One byte (the header byte) is always dedicated for overhead data.
5. The length of an SDO message is always 8, even if some of the bytes are unused. Unused data bytes are marked in the message header.
6. An expedited SDO has no place for data length indication. As a result, the length of the data field must be exactly 4 bytes.

9.1 Initiate SDO Download Protocol

This protocol is used to implement the Initiate SDO Download service for SDOs.
Client to Server:

0			1	4		8
7..5	4	3..2	1	0		
ccs = 1	x	n	e	s	m	D

Server to Client:

0	1	4	8
7..5	4…0		
scs=3	x	m	Reserved

- **ccs:** client command specifier

1: initiate download request

- **scs:** server command specifier

3: initiate download response

- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not

contain data. Bytes [8-n, 7] do not contain data.

- e: transfer type

0: normal transfer
1: expedited transfer

- **s:** size indicator

0: data set size is not indicated
1: data set size is indicated

- m: multiplexor. It represents the index/sub-index of the data to be transfer by the SDO.
- **d:** data

e = 0, s = 0: d is reserved for further use.
e = 0, s = 1: d contains the number of bytes to be downloaded.
Byte 4 contains the lsb and byte 7 contains the msb.
e = 1, s = 1: d contains the data of length 4-n to be downloaded,
the encoding depends on the type of the data referenced
by index and sub-index
e = 1, s = 0: d contains unspecified number of bytes to be downloaded

- **X:** not used, always 0
- **reserved:** reserved for future use, always 0.

9.2 Download SDO Segment Protocol

This protocol is used to implement the Download SDO Segment service.

Client to Server:

0		1		8
7..5	4	3..1	0	
ccs = 0	t	n	c	seg-data

Server to Client:

0		1	8
7..5	4	3..0	
scs=1	t	x	reserved

- **ccs:** client command specifier

0: download segment request

- **scs**: server command specifier

1: download segment response

- **seg-data**: at most 7 bytes of segment data to be downloaded. The encoding depends on the type of the data referenced by index and sub-index
- **n**: indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
- **c**: indicates whether there are still more segments to be downloaded.

0 more segments to be downloaded

1: no more segments to be downloaded

- **t**: toggle bit. This bit must alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.
- **X**: not used, always 0
- **reserved**: reserved for future use, always 0

9.3 Initiate SDO Upload Protocol

This protocol is used to implement the Initiate SDO Upload service.
Client to Server:

0	1	4	8
7..5	4…0		
ccs=2	X	m	reserved

Server to Client:

0			1	4		8
7..5	4	3..2	1	0		
scs = 2	x	n	e	s	m	d

- **ccs:** client command specifier
2: initiate upload request
- **scs:** server command specifier
2: initiate upload response
- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not
contain data. Bytes [8-n, 7] do not contain segment data.
- e: transfer type
0: normal transfer
1: expedited transfer
- **s:** size indicator
0: data set size is not indicated
1: data set size is indicated
- m: multiplexor. It represents the index/sub-index of the data to be transfer by the SDO.
- **d:** data
e = 0, s = 0: d is reserved for further use.
e = 0, s = 1: d contains the number of bytes to be uploaded.
Byte 4 contains the lsb and byte 7 contains the msb.
e = 1, s = 1: d contains the data of length 4-n to be uploaded,
the encoding depends on the type of the data referenced
by index and sub-index
e = 1, s = 0: d contains unspecified number of bytes to be uploaded.
- **X:** not used, always 0
- **reserved:** reserved for future use , always 0

9.4 Upload SDO Segment Protocol

This protocol is used to implement the Upload SDO Segment service.
Client to Server:

0		1	8
7..5	4	3..0	
ccs=1	t	x	Reserved

Server to Client:

0		1		8
7..5	4	3..1	0	
scs = 0	t	n	c	Seg-data

- **ccs:** client command specifier

3: upload segment request

- **scs:** server command specifier

0: upload segment response

- **t:** toggle bit. This bit must alternate for each subsequent segment that is uploaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.
- **c:** indicates whether there are still more segments to be uploaded.

0: more segments to be uploaded

1: no more segments to be uploaded

- **seg-data:** at most 7 bytes of segment data to be uploaded. The encoding depends on the type of the data referenced by index and sub-index
- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0.

9.5 Abort SDO Transfer Protocol

This protocol is used to implement the Abort SDO Transfer Service.
Client to Server or Server to Client:

0	1	4	8
7..5	4…0		
cs=4	x	m	d

- **cs:** command specifier
4: abort transfer request
- **X:** not used, always 0
- m: multiplexor. It represents index and sub-index of the SDO.
- **d:** contains a 4 byte abort code about the reason for the abort.
The abort code is encoded as UNSIGNED32 value.

9.6 Uploading Data Using an SDO

Uploading data has two basic formats. Loading a short (up to 4 bytes) data item is handled by one message conversation called an expedited SDO.
Longer data items are called segmented transfers. Segmented transfers require a longer conversation.

9.7 Example: Expedited SDO

In this example, we use an SDO to read the number of SDO s supported by the servo drive. The result is in object 0x100f in the OD. The result (One transmit SDO and one receive SDO) is formatted as the 32 bit word 0x00010001.
The client message body is described in the following table (% means a binary number):

Byte	Value	Description	Comment
0	%01000000	Header	Leading %010 is the client command specifier (ccs) for **initiate domain upload**.
1	0x0f	Index(LO)	
2	0x10	Index(HI)	Use Little Endian.
3	0	Sub-index	No sub index, so set to 0.
4-8	0	Reserved	

Table 9-1: Expedited SDO – Client Message

The server responds as follows:

Byte	Value	Description	Comment
0	%01000011	Header	Bits 7..5 - %010 is the client command specifier (ccs) for **initiate domain upload**. Bits 3..2 are the **n** bits that means all of data bytes are relevant. Bits 1..0 %11 is for expedite transfer and **d** contains information.
1	0x0f	Index(LO)	
2	0x10	Index(HI)	Use little endian.
3	0	Sub-index	No sub index, so set to 0.
4	1	Data: 0x00010001 in little endian format	If bit 0.3 is 1, bytes 4 to 7 carry an error code.
5	0		
6	1		
7	0		

Table 9-2: Expedited SDO – Server Response

9.8 Downloading Data Using an SDO

Data downloading with SDO is very similar to data uploading. Data downloading can be handled in a single message conversation (expedited transfer), or in a segmented conversation. A detailed example for an expedited download transfer may be found below in the PDO mapping.

9.9 Error Correction

An SDO transaction has features that enable error detection and correction.
Error detection is possible both with software and hardware.
Hardware error conditions relate to overrun, excessive line noise, broken line or other malfunctions of the physical layer.
Software detection is possible by monitoring the toggle bit, which correlates each domain segment with its previous and its next. An unexpected toggle bit signals an error.
An SDO transaction may be theoretically completed successfully in spite of errors. The slave that received a corrupted message may confirm it with an active “fault” bit (Byte 1, bit 3). The response to the fault bit should be a re-transmission of the faulty message.
Elmo servo drives do not try to save a corrupted transaction.
If the servo drive detects an error, by hardware or by software, the servo drive aborts the transaction immediately using the “Abort domain transfer” protocol.
If the servo drive uploads data and the server issues a confirmation message with the “fault” bit on, then the servo drive responds with an immediate “Abort domain transfer” message.
The “Abort domain transfer” message is structured as follows:

Byte	Description
0	0x82
1-2	Index
3	Sub-index
4	Additional code
5	Error code
6-7	Error class

Table 9-3: Abort Domain Transfer Message Structure

Fields 4-7 are described precisely in Section 8 of [1].

9.10 SDO Abort Codes:

Please refer to ***Abort SDO Transfer Protocol***:

Abort code	Description
0503 0000h	Toggle bit not alternated.
0504 0001h	Client/server command specifier not valid or unknown.
0504 0005h	Out of memory.
0601 0000h	Unsupported access to an object.
0601 0001h	Attempt to read a write only object.
0601 0002h	Attempt to write a read only object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility in the device.
0606 0000h	Access failed due to an hardware error.
0607 0010h	Data type does not match, length of service parameter does not match.
0607 0012h	Data type does not match, length of service parameter too high.
0607 0013h	Data type does not match, length of service parameter too low.
0609 0011h	Sub-index does not exist.
0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0609 0036h	Maximum value is less than minimum value.
0800 0000h	General error. – Refer to object 0x206A for more details
0800 0000h	General error.
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of a file error).

Table 9-4: SDO Abort Codes

10 Process Data Objects (PDO)

10.1 Receive PDO

Two Receive PDOs are used.
The first PDO is a high priority PDO.
This PDO is used for high-speed motion mode, to avoid the overhead of the use of the standard interpreter.
Its mapping is dynamic, according to the setting of its PDO mapping at index 0x1601 by an appropriate SDO.
The objects that may be mapped to the first PDO are:

Index	Meaning	Access
2001h	PVT motion command	Write
2002h	PT motion command	Write
2003h	Fast position command (Reserved)	Write
2004h	ECAM entry	Write

Table 10-1: Process Data Objects

By default, the first PDO is mapped to object 0x2001, which is PVT.

The second receive PDO is used for accessing the command interpreter via binary commands.
The mapping to the second PDO is fixed - its PDO mapping is read only.

Example
In order to switch PDO1 to program PT type motion command (Object 0x2002), set the object at index 0x1600 (PDO mapping, type defined by object 0x21, please refer to [2] page 9-86 and 9-87 for more understanding of this procedure)

Sub-index 0	Unsigned 8	1
Sub-index 1	Unsigned32	(0x2002<<16)[2] +0+2 = (index<<16) + (sub-index<<8) + data length

Note: that the number of objects mapped to this PDO is always 1.
For this reason it is not required to write to sub-index 0 of the PDO mapping and it is only required to program sub-index 1 of object 0x1600.

[2] The notation << denotes the bit shift operator. For example 4 is 100 binary, 4<<2 is 10000 binary which is 16

The PDO mapping may be set using a single expedited SDO.
The SDO will be

Byte	Value	Description
0	0x27	Initiate download, expedited, index valid, data valid.
1	0	Index to store at.
2	0x16	Index to store at.
3	1	Sub-index to store at.
4	0	The value to be inserted at sub index 1, according to the little endian convention.
5	2	
6	02	
7	0x20	

Table 10-2: Service Data Objects

The SDO must be answered by

Byte	Value	Description
0	0x67	Initiate download, expedited, index valid, data valid, no failure.
1	0	Index to store at.
2	0x16	Index to store at.
3	1	Sub-index to store at.
4	0	Reserved. Bytes 4-7 are error code if bit 0.3 is 1 (failure indication).
5	0	
6	0	
7	0	

Table 10-3: Answered SDO

10.2 Transmit PDO

Only Transmit PDO2 is used.
This transmit PDO is used to return binary answers from the standard string command interpreter. This PDO cannot be mapped, as its content is indirectly mapped by the mapping of the second received PDO.

10.3 PDO Mapping

PDO mapping is a procedure that enables you to transmit and receive information from and to the node in efficient way.
Nodes parameters are available through CAN objects (COB). All the CAN objects are defined in the object dictionary (Please refer to The Object Dictionary in this manual). You can require or set any COB (according to the object definition and limitation) by the mean of SDO (Please refer to Service Data Objects (SDO) in this manual). SDO provides a confirmed message process that usually increases busload. It can serve you very well for setup and background operations. However in order to be able to communicate with the amplifier in a more intensive way without increasing the busload, COB can be mapped into a PDO. The mapped PDO is used as a carrier to these objects and it will be

transmitted or received according to a predefined **trigger**. PDO has several advantages. It is an unconfirmed message and it can carry more then one object. In some cases a PDO can carry up to 64 different objects, it depends upon the **granularity** definition. Elmo's amplifier **granularity** is 8 thus up to 8 objects can be mapped to a single PDO each in the size of 1 byte.

10.3.1 The Mapping Trigger

Trigger is the event in which a PDO is received into the amplifier or transmitted from the amplifier. The trigger must be predefined. Please refer to pages 9-83 and 9-84 in CiA DS301 for more comprehensive understanding about transmission types. Here is a short summarized of the subject. The exact mechanism of mapping is described in CiA DS301 version 4.01 under objects 0x1400 to 0x1800.

There are several triggers available:

1. **Synchronized Trigger** – Received or transmitted when number of predefined SYNC messages was received
2. **Unsynchronized Trigger –** Also known as **event trigger**. Received or transmitted according to a predefined event. Event can be interpreter reception, motion done and more.
3. **Event Timer Trigger –** Received or transmitted when predefined time was elapsed.

10.3.1.1 The Synchronized Trigger

These types of trigger depend upon the reception of the SYNC message. You must define how many SYNC messages should arrive before the specified PDO will be transmitted or received.

Note:
For synchronized trigger, Elmo's amplifiers support only the transmitted PDO (TPDO).

Values of SYNC messages can be defined from 1-240. When this value is set to 0 the trigger is considered as an **acyclic** and will be transmitted only once upon the receiving of the first SYNC. Other values are **cyclic** and will transmitted each time the define amount of SYNC messages arrived.

Note:
Acyclic trigger must be redefined (sub index 0 of the relevant object 0x1400+node ID or 0x1600+ndoe ID set to 1).

In order to map a synchronized TPDO please follow these steps:
The **example** in this case maps the system status (object 0x2F0A) and current position (object 0x6064) to TPDO1 that will be transmitted each 3rd SYNC message.

Object	Sub-Object	Value	Meaning
0x1802	2	3	Set transmission type to 3
0x1A02	0	0	Reset all entries before new mapping
0x1A02	1	0x2F0A0020	1st entry object 0x2F0A is 32 bits
0x1A02	2	0x60640020	2nd entry object 0x6064 is 32 bits
0x1A02	0	2	Activate mapping of TDPO3 with 2 entries

After the last object was sent each 3 SYNC messages TPDO3 will be transmitted with the status word and the current position (PX).
If the position was PX=10000 the result should be:

Byte1 LSB							MSB Byte 8
0x10	0x27	00	00	00	00	0x20	0x50

10.3.1.2 The Unsynchronized Trigger

Unsynchronized triggers are triggers that can be set at any time. It also called event trigger. The event triggers in Elmo's amplifiers are predefined to specific objects that actually describe the event.

10.3.1.2.1 Transmit Event Trigger

For **example:** object 0x2050 is an object that points out to end-of-motion event in point-to-point motion modes. When mapping this event to a PDO, the PDO will be transmitted whenever the motion was done. This saves from the host the need to poll the motion status.

- The amplifier supports up to 8 Events Trigger in one TPDO.
- Second object can be mapped into the same PDO – one map able object to one event trigger.
- If PDO was already mapped with event trigger and another non-event object no more objects can be mapped into this PDO.
- The data length (DLC) of a transmitted PDO in case of event trigger is always 8.
- Long (32 nits) and short (16 bits) objects are supported.

Note:
Only one PDO can be used to each event object - the last PDO that was mapped will be transmitted when event occurred.

Note:
The transmission type of an event trigger is always 254.

The following steps should be taken in order to use event trigger. In the **example** 3 events are mapped:
Erupt instruction (object 0x2051) with integer array (IA[4]) is mapped to TPDO1.
MS command (object 0x2050) to TPDO4 and HM (object 0x2052) event is mapped to the same TPDO4.

Object	Sub-Object	Value	Meaning
Mapping TPDO1 to event trigger and none event trigger			
0x1800	2	254	Transmission type to manufacture
0x1A00	0	0	Reset all entries before new mapping
0x1A00	1	0x20510020	1st entry object 0x2051, 32 bits
0x1A00	2	0x2F000420	2nd entry object IA[10] is 32 bits
0x1A00	0	2	Start event trigger on TPDO1
Mapping TPDO4 to 2 events trigger			
0x1803	2	254	Transmission type to manufacture
0x1A03	0	0	Reset all entries
0x1A03	1	0x20510020	1st entry object for EI trigger event
0x1A03	2	0x20520020	2nd entry object for homing event
0x1A03	0	2	Start event trigger on TPDO4

After the events were mapped TPDO1 will be transmitted whenever EI=<VALUE>. Please refer to Object 0x2051 EI Event Trigger for more details. Each message includes the VALUE according to EI and the value of IA[4]. In the case that EI=4 and IA[4]=6 the message should look:

Byte1 LSB							MSB Byte 8
0x6	00	00	00	04	00	0x51	0x20

When ever motion status will be steady (Please refer to Object 0x2050 MS Event Trigger in this manual) or HM[1] value will be reset from 1 to 0, TPDO4 will be transmitted with information according to the mapped event.

For motion status event:

Byte1 LSB							MSB Byte 8
00	00	00	00	00	00	0x50	0x20

For main homing event:

Byte1 LSB							MSB Byte 8
00	00	00	00	00	00	0x52	0x20

Note:
TPDO2 and RPDO2 are statically mapped to the interpreter transmits and receive object. These objects 0x2012 and 0x20123 are virtual objects and can be accessed through the object dictionary. PDO2 is reserved for these objects and cannot be used for any other purpose. Please refer to Binary Interpreter Commands for more details.

10.3.1.2.2 Receive Event Trigger

We distinguish between two types of unsynchronized received event trigger.

1. The VIP messages are messages that mapped in to RPDO1 (receive PDO) that is reserved for this purpose. These messages are for tabulated motion modes (PVT, PT and ECAM) fast updating. They are activated directly from the real time in order to perform fast motion modes. Please refer to Tabulated Motion Modes in the Software Manual.
2. The binary interpreter message. RPDO2 is dedicated to this purpose and is statically mapped with this object.

In order to change the VIP messages object please follow the following step. The **example** in this case maps object 0x2002 (PT) to RPDO1.

Object	Sub-Object	Value	Meaning
0x1400	2	255	Reception trigger upon message receive.
0x1600	0	0	Reset all entries before new mapping
0x1600	1	0x20020040	1st entry object 0x2002 is 64 bits
0x1600	0	1	Activate mapping of RPDO1 with 1 entry.

When this sequence is done, any RPDO1 will be treated as reception of object 0x2002.

10.3.1.3 The Timer Trigger

Time trigger are events that are time depended. They are not applicable with Elmo's amplifier at this stage.

11 Emergency

Emergency is COB 80 to ff.
Emegencies object usage is defined in [2] page 9-38.
The structure of the manufacturer specific emergency message is:

0	Error code.
1	
2	Error register.
3	Elmo Error code. Please refer to EC command in the Cmmand Reference.
4	Error code data field 1.
5	
6	Error code data field 2.
7	

The following table lists the supported CAN emergencies:

11.1 Emergency Codes Related to Failure

The servo drive issues an emergency code when:
- Motion is stopped due to a motor failure.
- The CPU encounters a fatal exception.
- Communication error .

The following table details the emergency object attached to failures:

Error code/ Error register/ Elmo error code	Symbolic name	Reason	Data field
0x8100/0x10/9	CAN_MESSAGE_LOST	CAN message overrun.	Data field 1: 1: Lost NMT 2: Lost SYNC 3: Lost PDO1/PDO2/SDO1 - Overrun 4: Lost PDO2-Queue full 5: Lost SDO1-Queue full 6: Missed sync[3].
0x8100/0x10/14	CANT_DO_NMT	NMT command cannot be done.	Data field 1: 1: Communication reset command with motor on.
0x8100/0x10/0xa	PDO_NOT_CONFIGURED	Attempt to use a PDO that does not have a valid mapping.	None.
0x8100/0x10/0xb	BAD_COB_ID	Cannot identify COB ID although the device ID matched.	None.
0x6100/01/0x32	STACK_OVERFLOW	CPU encountered fatal exception.	Data field 1: 1: Stack overflow. 2: CPU exception.
0x8000/0x20/0x33	USER_PROGRAM_EMCY	User program encountered an error.	Run time error.
0x2300/0x2/		Short circuit.	
0x3300/0x4/		Over voltage.	
0x3300/0x4/		Under voltage.	
0x4000/0x8/		Over temperature.	
0xff00/0x80/		Motor fault-other. (Inhibit is not a fault).	Data field 1: (Please refer the MF command in [6]). 1 – main encoder. 2 – aux encoder. 0x40-Hall sensor. 0x80-Speed error. 0x100-Pos error. 0x1000-Bridge failure. 0x10000-Commutation seek failure.

[3] A lost Sync is not a lost message, but a message that its exact timing could not be determined. See section on SYNC below.

			0x20000-Over speed. 0x400000 – Pos limits. 0x800000 – Ref error. 0x10000000 – Cant tune current offsets.

Table 11-1: Emergency Code

Important Note:
A CAN_MESSAGE_LOST emergency may indicate an overrun – a condition in which a CAN message has not been retrieved from the receiver on time. The next message to the same buffer crashed with the as yet unread message. Both the messages may have been lost in the crash.
There may also be more lost messages that were not detected, since they may have been sent while the message loss indication was already on.
The CAN_MESSAGE_LOST emergency tells where a crash occurred – it does not tell how many messages have actually been lost.

11.2 Emergency Codes for Interpreter

The interpreter issues an emergency object when a program command cannot be executed, because:
- It is not understood (e.g. the non existing command XX)
- It is not properly formatted (e.g. data length = 5)
- It cannot be executed in the present context (e.g. BG command while the motor is OFF)
- It has an index or a parameter out of range.

The emergency error code for all the interpreter-generated emergencies is Error code = 0x6200 and error register = 0x20

Error code	Symbolic name	Reason	Data field
5	CAN_STRING_MESSAGE	An MG program command prepared a message – please read.	Field 1: Number of messages in queue.

Table 11-2: Emergency Code For Interpreter

11.3 Emergency Codes for Motor Fault

The following table describes emergencies that are related to motor disable.
Please refer to MF command in [6].

Error code (Hex)	MF value (Hex)	Meaning	Detail
2311	8000	Current Limit	ASIC fault – Current limit exceeded.
2341	B000	Short	ASIC fault – A short was sensed on motor power outputs.
3100	7000	SVP	ASIC fault – Internal power supply problem.

3120	3000	Under Voltage	ASIC fault – Input voltage is under the minimum.
3311	5000	Over Voltage	ASIC fault – Input voltage exceed maximum permitted.
4310	D000	Temperature	ASIC fault – Amplifier temrerature high.
5280	20	FPGA	Digital hardware failure.
5281	100000	Timing Error	Programmable logic exception.
5282	F000	Commutation	ASIC fault – Illegal combination for DHALL.
5283	800	Node Guarding Event	This error occurs if the Metronome is set to operate under life guarding in a CANopen network.
5441	10	External Inhibit	An abort from a condition of the input logic. (IL command)
5480	1000	Power Up Reset	
6180	40000	Stack Overflow	This may happen if the CPU had been subject to a load it can't handle.
6181	80000	Null Interrupt	A fatal error. (like divided by 0).
6320	200	Bad Data Base	Cannot start because of inconsistent database.
7121	200000	Motor Blocked	Motor is blocked according to CL[2],CL[3] restrictions.
7305	1	Main Encoder Error	A and B main encoder channel s fault.
7306	2	Auxiliary Encoder Error	A and B aux encoder channels fault.
7380	4	Feedback Loss	No match between encoder and hall location.
7381	40	Digital Hall Sensor	Two digital Hall sensors had been changed at the same time.
8311	8	Peak Over Current	The peak current has been exceeded.
8312	10000000	Can Not Tune Current Offsets	During motor enable process, A/D for current gets unreasonable values.
8380	10000	Can not find zero Dhall	Failed to find the electrical zero of the motor in an attempt to start a motor with an incremental encoder and no digital Hall sensors.
8480	80	Speed Tracking Error	The speed tracking error DV[2]-VX exceeded the speed error limit ER[2].
8481	20000	Over Speed	Speed limit exceeded: VX<LL[2] or VX>HL[2].
8611	100	Position Tracking Error	The position tracking error DV[3]-PX (UM=5) or DV[3]-PY (UM=4) exceeded the position error limit ER[3].
8680	400000	Out Of Position Limit	Position limit exceeded: PX<LL[3] or PX>HL[3] (UM=5), or PY<LL[3] or PY>HL[3] (UM=4).

Table 11-3: Emergency Code For Motor Fault.

11.4 Emergency Codes Related to PVT/PT Motion

In the course of PT/PVT motion, the servo drive may issue emergency objects, to signal an error, or to signal the immediate need of additional data to prevent data queue underflow (please refer [3] and [4]).
The emergency error code for all the messages below is 0xff00, and the error register is 0x80.

Error code	Symbolic name	Reason	Data field
86 0x56	PVT_QUEUE_LOW	The rows for the entire left valid PVT program has dropped below the value stated in MP[4].	Field 1: Write pointer. Field 2: Read pointer.
91 0x5b	BAD_HEAD_POINTER	Write pointer out of the physical [1,64] range of the PVT table.The reason may be a bad setting of MP[6].	The value of MP[6].
92 0x5c	PDO_NOT_CONFIGURED	The PDO 0x3xx is not configured for a type of motion.	
59 0x34	PVT_QUEUE_FULL	An attempt has been made to program more PVT points then supported by the queue.	The index of the PVT table entry that could not be programmed.
7 0x7	BAD_MODE_INIT_DATA	Cannot initialize motion due to bad setup data. Reasons: - The write pointer is outside the range specified by the start pointer and the end pointer.	
8 0x8	MOTION_TERMINATED	Mode terminated, and the motor has been automatically stopped (In MO=1).	Data field 1: Write pointer. Data field 2: 1 for End of trajectory in non cyclic mode. 2 for A zero or negative time specified for a motion interval. 3 for Read pointer reached write pointer.
9 0x9	CAN_MESSAGE_LOST		A CAN message has been lost. The servo drive did not fetch a message before it was overwritten by a new message.

Table 11-3: Emergency Code (PVT/PT Motion)

12 Network Management (NMT), and Synchronized Motion Initiation

Only a minimal set of network management (NMT) services is supported, as required by the CANopen minimum capability network management services.
NMT commands are used to control the communication-state of the servo drive, and to broadcast manufacturer messages that are targeted to all the listening servo drives.
The following network communication states are supported.

State	Description
Un powered / Initialization	Servo drive is not ready, or performing the boot sequence. It will not respond to communication and it will not transmit anything.
Pre-operational	Servo drive boot sequence completed, but no command has been received to enter operational mode. The servo drive will respond to SDO and NMT messages, but not to PDO.
Operational	The servo drive is fully operational, responding to PDO SDO and NMT messages.
Prepared	The servo drive received a stop-node command. It responds only to NMT services.

Table 12-1: Network Management (NMT)

The Initialization-state is entered when the servo drive is powered. After completing the boot sequence the pre-operational state is entered automatically.
The transition between the Operational, Pre-operational, and Prepared states is according to NMT messages. The COB-ID of an NMT command is always 0.
An NMT message is always 2 bytes long.
The first byte is the command specifier.
The second byte is the ID of the units that are to respond to the message. If the ID is 0, the NMT message will be executed by all the listening servo drives.

The following NMT services are supported:

Command specifier	Service
1	Start remote node (go to operational).
2	Stop remote node (go to prepared).
128	Enter pre operational state.
129	Reset node (Perform the full software reset).
130	Reset communication (Reload the communication parameters from the flash, reset the CAN software, and enter pre-operational). **Note:** NMT service 130 is available only in MOTOR OFF.
192	Synchronized BG (Begin) command.
193	Synchronized ST (Stop) command.
194	Reset clock on next sync.

Table 12-2: Supported NMT Services

The BG and the ST commands (command specifier 192) are applicable only when the network is operational, and when the motor is ON.
The Reset clock command is applicable only when the network is operational.
The reset clock (command specifier 194) zeroes the microsecond counter that serves to synchronize PVT and other complex motions. The reset clock command will become active upon the next SYNC. The value of the microsecond clock can is latched every SYNC, and is readable via object 0x2041. Reading object 0x2041 helps to synchronize PT motions, which are specified relative to the sampling time of the controller.

For the BG, the ST, and the reset clock commands, The ID specifies to whom the command is applicable.
A servo drive will respond to a BG/ST/Reset clock command if the identifier is:
- 0.
- The servo drive ID.
- The coordinate systems ID (object 0x2040 in the OD).

13 Sync and Time Stamp

The sync message has two uses.
The first use is to synchronize the operation of synchronous PDOs

- Synchronous receive PDOs are not supported by Elmo servo drives.
- Synchronous transmit PDOs can be used to transmit data from the servo drive upon receiving a sync signal.

The second use is to synchronize the motion clock of the servo drive with a clock in the network master. The synchronization is made in conjunction with the time stamp message.
The motion clock of the servo drive counts microseconds (regardless of the sampling time of the servo drive). The motion clock is cyclic, and has 32 bits. This means that the motion clock completes a full cycle in 4295 seconds (approx. 72 minutes). When the motion clocks of all the servo drives are synchronized to the motion clock of the master, then multiple servo drives can do complex synchronized motion with exact timing set by the network master.

Amplifier synchronization is made by transmission of a SYNC message, whose arrival time is captured by the amplifier. Upon SYNC reception the amplifier latches its internal timer.
A Time Stamp is a 32 bit message which contains the master internal clock as it was generate upon reception of the client own SYNC. A Time Stamp causes a clock synchronization cycle to be executed . The amplifier uses the Time Stamp as an absolute timer and adjust its internal timer[4] with relation to the time that was latched in the last SYNC. A synchronization cycle does not imply that after that the clocks of the master and the amplifier are synchronized. To ensure that the timing jitter of the time stamping process will not adversely affect motion smoothness, the synchronization process is filtered. It takes about 200 SYNC-Time stamp pairs to ensure that the clocks of the master and amplifier are fully synchronized. This process can be considerably shortened by referring to NMT service 194 (SYNC on next). The next SYNC-Time stamp pair after an NMT 194 service, will lock the time of the stamp to the amplifier as accurately as possible in a single step, without any filtering.

The Time Stamp is object 0x1013 of the object dictionary. COB-ID 256 (0x100) is (a constant) dedicated ID for this purpose. The master can send Time Stamps at any moment.
A time stamp always refers to the previous SYNC message. Interval between Time Stamp to previous SYNC must not exceed 5 second.
The initial synchronization of a master to its amplifiers is recommended at the pre-operational state.

13.1 Important Note

The 82527 CAN controller used by Elmo servo drives cannot mark the time of the sync. Consequently, the exact timing of a sync signal can be measured accurately only if at that time, no other communication object has been serviced[5]. For optimum synchronization, be sure that a time-stamped Sync is transmitted after at least 1msec no other client COB has been issued.

[4] The higher value to adjust in single time stamp message is 250mSec.
[5] The Elmo servo drives actually measure the time at which the 82527 issued an interrupt request. If the Sync is accepted when the 85257 were requesting service or being serviced, its acceptance time cannot be resolved. This condition is easily detected and the unresolved sync is ignored. An emergency will be set if 5 consecutive sync signals are rejected.

! The priority of the SYNC message is higher then all COB-ID (beside the NMT). Events which are SYNC depended like TPDO mapping , Time Stamp or synchronized RPDO should be carefully handled. The needed triggerd event must be handled by the amplifier before the next SYNC is.

14 Binary Interpreter Commands

The interpreter commands are sent in binary form (CAN only). The commands used by the binary interpreter are very similar to commands of the ASCII interpreter that is used for RS232 communication.
The binary interpreter may be used for setting and retrieving all the numerical data of the servo drive setup.
The binary interpreter does not support string operations.
Getting and setting strings may be done by accessing the appropriate messages via SDO.
Expressions (such as AC=2*DC+1000) are not supported by the binary interpreter.

The following table summarizes the main differences between the binary interpreter used for CAN communication and the ASCII interpreter used for RS-232.

Feature	ASCII interpreter	Binary interpreter
Command length.	Depends on data.	Fixed: 8 bytes for Set commands, 4 bytes for Get commands.
Delimiter.	Commands and servo drive responses are delimited by ';' or <CR>.	No delimiters.
Servo drive response to set commands.	Always.	Servo drive does not respond to Set commands. An emergency object is sent if the command execution fails.
Long response strings.	Some commands like LS and BH return long strings.	No interpreter support for returned long strings. Long strings can be read via SDO.

Table 14-1: Binary Interpreter Commands

The binary interpreter supports 3 types of commands:
- Set value. These commands have a data length of 8. They are normally answered by an identical reflection of the set command.
- Get value. These commands have the data length of 4. An 8-byte response includes an echo of the command and the resulting numerical value.
- Execute command. This command has a data length of 4. An 8 byte response includes an echo of the command.

If an interpreter command could not be serviced for any reason, bit 3.6 is set on, and byte 7 of the response contains the Elmo error code – please refer the EC command.

The interpreter binary command has the following fixed structure:
Set value command:
Data length = 8.

Byte	Description
0	First command character. For example, 'C' for the CA command. ***Must be uppercase***.
1	Second command character. For example, 'A' for the CA command. ***Must be uppercase***.
2-[3.0:3.5]	Index for vectored commands, 0 for scalar commands. ***Use little endian (Intel) word format***.
3.6:3.7	Flags. Bit 3.7 is 1 if the number in bytes 4-7 is written in floating point format. Bit 3.7 is 0 if the number in bytes 4-7 is written in long integer format. Bit 3.6 has 2 cases: When TPDO it is denotes a fault indications in response messages.When '1' then bytes 4-7 of the response are to be interpreted as an error code, NOT as a valid value. When RPDO it is denotes an inquiry request. When value is '1' the received message is an inquiry regardless to the amount of bytes (DLC) in the received PDO. Response will inclued 8 bytes.
4-7	Parameter for command, long or float. ***Use little endian (Intel) word format***.

Table 14-2: Binary Interpreter Commands – Set Value Command

Get value command: (inquiry)
Data length = 4.

Byte	Description
0	First command character. For example, 'C' for the CA command. ***Must be uppercase***.
1	Second command character. For example, 'A' for the CA command. ***Must be uppercase***.
2-[3.0:3.5]	Index for vectored commands, 0 for scalar commands. ***Use little endian (Intel) word format***.
3.6:3.7	Flags. Bit 3.9 is 1 if the number in bytes 4-7 is written in floating point format. Is 0 if the number in bytes 4-7 is written in long integer format.

Table 14-3: Binary Interpreter Commands – Get Value Command

Exec command:

Byte	Description
0	First command character. For example, 'B' for the BG command. ***Must be uppercase***.
1	Second command character. For example, 'B' for the BG command. ***Must be uppercase***.
2-3	0

Table 14-4: Binary Interpreter Commands – Execute Command

Note: that the client determines a Get command from its corresponding Set command by the data length.
The length of a data request is 4 bytes, whereas the length of data assignment message is 8.

For example, the command QP[1000]=0x12345678 is (1000 decimal is 0x3e8).

Byte	Value
0	'Q'
1	'P'
2	0xe8
3	0x3
4	0x78
5	0x56
6	0x34
7	0x12

Table 14-5: Binary Interpreter Commands – Example QP Command

The command AC (data request) is:

Byte	Value
0	'A'
1	'C'
2	0
3	0

Table 14-6: Binary Interpreter Commands – Example AC Command

The command MC=1.0 is (given that the IEEE representation of 1.0 is 0x3f800000).

Byte	Value
0	'M'
1	'C'
2	0x0
3	0x80
4	0x0
5	0x0
6	0x80
7	0x3f

Table 14-7: Binary Interpreter Commands – Example MC Command

Set value commands are not answered, unless an exception occurred.

Get value commands are answered in a format similar to the set command: The first four bytes are a reflection of the data request and the last four bytes contain the information, long or float. The flag bit in byte 2 signals whether the numerical field is to be interpreted as a long integer, or as an IEEE floating point number.

Important Note:
Always use the data type bit (3.7) in the returned interpreter message, even though the numerical data type is known in advance and given in the reference manual. This is because Elmo cannot

guarantee that the type of numerical data returned for any given interpreter command will remain unchanged in future versions.

14.1 ASCII Interpreter Commands That Are Not Supported By The Binary Interpreter

Some commands deal with strings. These commands are not accessible by the binary interpreter. In most cases, these strings may be accessed as objects from the OD using the SDO.

Command	Description	Alternative
VR	Detailed software version string.	Use SDO to read object 0x100a.
CD	CPU dump for the case of a fatal exception.	Use SDO to read object 0x2080.
CS	Code status and Single code for program.	Use SDO to read object 0x2071.
LS/DL	List/Download a program.	Use SDO to read/write object 0x2070.
MZ	Program message.	Use SDO to read object 0x2072.
CC	Program compilation. Program compilation is made automatically upon program loading.	Use SDO to read object 0x2073.
IN	Manual input to program.	Available for RS-232 only.
DF	Down load firmware version.	Use SDO to write object 0x2090. If a previous DF failed, DF is available for RS-232 only.
BH	Bring recorded value.	Use SDO to read object 0x2030.
SC	Single code for user program.	Use SDO 0x2074 to advance the program counter by a single step while uploading the same line.

Table 14-8: Not Binary Interpreter Commands

The binary interpreter cannot handle expressions. In order to deal with expressions, use the programming feature.

15 Communication Object Details

Elmo CAN products support communication objects as follows:

15.1 Object 0x1200 SDO Server Parameter

Object Description:

INDEX	1200H
Name	Server SDO 1 Parameter
Object Code	RECORD
Number Of Elements	3
Data Type	SDOPar (Object 0x22)

Values Description:

Sub-Index	0
Description	Number of entries
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	2
Default Value	No

Sub-Index	1
Description	COB ID Client->Server (rx)
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	600H + Node ID

Sub-Index	2
Description	COB_ID Server -> Client (tx)
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	No
Default Value	580 + Node ID

15.2 Object 0x1400 – 0x1403 Receive PDO Communication Parameter

Object Description:

INDEX	1400H – 1403H
Name	Receive PDO parameter
Object Code	RECORD
Number Of Elements	2
Data Type	PDOCommPar (Object 0x20)

Values Description:

Sub-Index	0
Description	Number of entries
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	2-4
Default Value	No

Sub-Index	1
Description	COB ID used by PDO
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	Index 1400H: 200H + Node ID Index 1401H: 300H + Node ID Index 1402H: 400H + Node ID Index 1403H: 500H + Node ID

Sub-Index	2
Description	Transmission type
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8 ([2] Table 55 page9-84)
Mandatory Range	No
Default Value	255

Note:
PDO 1 receive is used for fast position mode please refer [3],[4].
PDO 2 receive is used for binary interpreter please refer ***Binary Interpreter Commands***.
PDO 3 receive for user specific application usage.
PDO 4 receive for user specific application usage.

15.3 Object 0x1600 – 0x1601 Receive PDO Mapping

Object Description:

INDEX	1600H – 1601H
Name	Receive PDO mapping
Object Code	RECORD
Number Of Elements	1 (granularly 64)
Data Type	PDOMapping (Object 0x21)

Values Description:

Sub-Index	0
Description	Number of mapped application object in PDO.
Object Class	Optional
Access	Rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	1
Default Value	1

Sub-Index	1
Description	PDO mapping for the 1st and last application object to be mapped
Object Class	Optional
Access	Rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	Object 1600H – 0x20010040 (PVT) Object 1601H – 0x20120040

Note:
PDO 1 receive mapping is possible only to objects 2001H, 2002H and 2003H.
PDO 2 receive mapping is controller usage for binary interpreter input.

15.4 Object 0x1800 – 0x1803 Transmit PDO Communication Parameter

Object Description:

INDEX	1800H – 1803H
Name	Transmit PDO parameter
Object Code	RECORD
Number Of Elements	2
Data Type	PDOCommPar (Object 0x20)

Values Description:

Sub-Index	0
Description	Number of entries
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	2-4
Default Value	No

Sub-Index	1
Description	COB ID used by PDO
Object Class	Optional
Access	Object 1800H – RW Object 1801H – R Object 1802H – RW Object 1803H – RW
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	Index 1800H: 180H + Node ID Index 1801H: 280H + Node ID Index 1802H: 380H + Node ID Index 1803H: 480H + Node ID

Sub-Index	2
Description	Transmission type
Object Class	Optional
Access	Object 1800H – RW Object 1801H – R Object 1802H – RW Object 1803H – RW
PDO Mapping	No
Value Range	0-240 ([2] Table 55 page9-84)
Mandatory Range	No
Default Value	0

Note:
PDO 1-transmit is for user general purpose.
PDO 2-transmit is used for binary interpreter please refer ***Binary Interpreter Commands***.
PDO 3-transmit is for user general purpose.
PDO 4-transmit is for user general purpose.

15.5 Object 0x1A00 – 0x1A03 Transmit PDO Mapping

Object Description:

INDEX	1A00H – 1A03H
Name	Transmit PDO mapping
Object Code	RECORD
Number Of Elements	8 (granularity 8)
Data Type	PDOMapping (Object 0x21)

Values Description:

Sub-Index	0
Description	Number of mapped application object in PDO
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	8
Default Value	0

Sub-Index	1 – 8
Description	PDO mapping for the nth application object to be mapped
Object Class	Optional
Access	Object 1A00H – Rw Object 1A01H – Ro Object 1A02H – Rw Object 1A03H – Rw
PDO Mapping	No
Value Range	Unsigned32.
Mandatory Range	No
Default Value	

Note:
PDO 1-transmit mapping is for user general purpose.
PDO 2-transmit mapping is used for binary interpreter please refer ***Binary Interpreter Commands***.
PDO 3-transmit mapping is for user general purpose.
PDO 4-transmit mapping is for user general purpose.

16 Special Treated Object Details

This section provides more information about objects that are CiA DS-301 standard and has device specific details.

16.1 Object 1010H – Save Parameters

Purpose:
This object supports the saving of parameters in non volatile memory. By read access the amplifer provides information about its saving capabilities. The groups that are distinguished are:
Sub-index 0: The largest supported sub index.
Sub-index 1: Save all parameters.
Sub-index 2,3: not supported.
Sub-index 4: Save user program.
In order to avoid storage by mistake, storage is only executed when a specific signature is written to he appropiate sub-index. The signature is “save”.

MSB			LSB
‘e’	‘v’	‘a’	‘s’
65H	76H	61H	73H

Object Description:

INDEX	1010H
Name	Store parameters
Object Code	RECORD
Number Of Elements	4 (2 supported)
Data Type	Unsigned32

Values Description:

Sub-Index	0
Description	Largest supported Sub Index
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	4
Default Value	4

Sub-Index	1
Description	Save all parameters
Object Class	Optional
Access	Rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

Sub-Index	2
Description	(N/A)Save communication parameters
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	0
Default Value	0

Sub-Index	3
Description	(N/A)Save application parameters
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	0
Default Value	0

Sub-Index	4
Description	Save User program
Object Class	Optional
Access	Rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	1

Note:

16.2 Object 1011H – Restore Parameters

This object supports the restoring of parameters from non volatile memory. By read access the amplifer provides information about its restoring capabilities. The groups that are distinguished are:
Sub-index 0: The largest supported sub index.
Sub-index 1: Restore all parameters.
Sub-index 2,3: not supported.
Sub-index 4: Restore user program.
In order to avoid storage by mistake, restoring is only executed when a specific signature is written to he appropiate sub-index. The signature is “load”.

MSB			LSB
‘d’	‘a’	‘o’	‘l’
64H	61H	6FH	6CH

Object Description:

INDEX	1011H
Name	Store parameters
Object Code	RECORD
Number Of Elements	4 (2 supported)
Data Type	Unsigned32

Sub-Index	0
Description	Largest supported Sub Index
Object Class	Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	4
Default Value	4

Sub-Index	1
Description	Restore all parameters
Object Class	Optional
Access	Rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

Sub-Index	2
Description	(N/A)Restore communication parameters
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	0
Default Value	0

Sub-Index	3
Description	(N/A)Restore application parameters
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	0
Default Value	0

Sub-Index	4
Description	Restore User program
Object Class	Optional
Access	Rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	1

17 Manufacture Object Details

17.1 Object 0x2001 PVT Data

Purpose:
Set PVT data for PVT motion mode.

Object Description:

INDEX	2001H
Name	PVT data
Object Code	VAR
Number Of Elements	1
Data Type	PVT DataPar (Object 0x60)

Values Description:

Sub-Index	-
Description	-
Object Class	Optional
Access	Wo
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Byte Stream:
According to [3].

Note:
Transmission type (0bject 0x1400[2]) for mapping this object must be 255. An abort message will be transmitted otherwise (0604 0043h)

17.2 Object 0x2002 PT Data

Purpose:
Set PT data for PT motion mode.

Object Description:

INDEX	2002H
Name	PT data
Object Code	VAR
Number Of Elements	1
Data Type	PT DataPar (Object 0x61)

Values Description:

Sub-Index	-
Description	-
Object Class	Optional
Access	Wo
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Byte Stream:
According to [4].

Note:
Transmission type (0bject 0x1400[2]) for mapping this object must be 255. An abort message will be transmitted otherwise (0604 0043h)

17.3 Object 0x2003 Fast Position Data

Purpose:
Set Position Absolute data for fast position motion mode.

Object Description:

INDEX	2002H
Name	PA data
Object Code	VAR
Number Of Elements	-
Data Type	PT DataStruct (Object 0x61)

Values Description:

Sub-Index	-
Description	-
Object Class	Optional
Access	Wo
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Byte Stream:
According to [4].

Note:
This object is reserved for future use.

17.4 Object 0x2012 Binary Interpreter Inquiry

Purpose:
Binary interpreter object

Object Description:

INDEX	2012H
Name	Binary interpreter
Object Code	VAR
Number Of Elements	-
Data Type	Binary Interpreter inquiry (Object 0x62)

Values Description:

Sub-Index	-
Description	-
Object Class	M
Access	Ro
PDO Mapping	Yes – static maping
Value Range	No
Mandatory Range	No
Default Value	No

Byte Stream:
Please refer to ***Binary Interpreter Commands*** for details regarding this object.

Note:
This object is for computability with commands according to [2]. Therefore mapping this object to PDO 2 is a mandatory need and cannot be accessed by the user.

17.5 Object 0x2013 Binary Interpreter Command

Purpose:
Binary interpreter object

Object Description:

INDEX	2013H
Name	Binary interpreter
Object Code	RECORD
Number Of Elements	1
Data Type	Binary Interpreter Command(Object 0x63)

Values Description:

Sub-Index	-
Description	-
Object Class	M
Access	Wo
PDO Mapping	NO
Value Range	NO
Mandatory Range	NO
Default Value	NO

Byte Stream:
Please refer to ***Binary Interpreter Commands*** for details regarding this object.

Note::
This object is for compatibility with commands according to [2]. Therefore mapping to PDO 2 is mandatory and cannot be accessed by the user.

17.6 Object 0x2030 Recorder Data

Purpose:
Getting recorded parameters according to RC and the Sub-Index field.
The 0x1 Sub-Index will draw the parameter, which was recorded in RC=(1<<Sub-Index).

Object Description:

INDEX	2030H
Name	Bring Recorded Data
Object Code	RECORD
Number Of Elements	16
Data Type	UNSIGNED32

Values Description:

Sub-Index	0H
Description	Number of supported elements
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	UNSIGNED8
Mandatory Range	16
Default Value	16

Sub-Index	1H
Description	Main Speed
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	2H
Description	Main Position
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	3H
Description	Position Command
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	04H
Description	Digital Input
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	05H
Description	Position Error for UM=4,UM=5
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	06H
Description	Torque Command
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	07H
Description	Reserved to Bus Voltage
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	08H
Description	Auxiliary Position
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	09H
Description	Auxiliary Speed
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-
Sub-Index	0AH
Description	Active Current (IQ or IM)
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	0BH
Description	Reactive Current (ID value - Sax Only)
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	0CH
Description	Analog input 1
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	0DH
Description	Analog input 2
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	0EH
Description	Current phase A (IA value)
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	0FH
Description	Current phase B (IB value)
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Sub-Index	10H
Description	Speed Command
Object Class	M
Access	Ro
PDO Mapping	No
Value Range	please refer to ‘Header byte sequence’
Mandatory Range	No
Default Value	-

Bytes stream:

17.6.1 Bring Data Upload Process

Initiate SDO Upload protocol according to [2]. After confirmation a character stream is transmitted. Each octet has to be answered according to Upload SDO segmented Protocol, except sub index 0, which returns object supported entries in expedite SDO format.

Segmented response is built from the header and data stream.
Note: that Sub-Index in this object is a value of the RC bit field.

Header byte sequence:

Byte [bit]	Stream	Example
1[0-3]	Sample Time multiplier	1 = recorded each Ts * 1 * RG 4 = recorded each Ts * 4 * RG
1[4-5]	Variable type	0 = integer type 1 = float type
1[6-7]	Internal representation of characters	0 = 2 bytes are 1 item (short value) 1 = 4 bytes are 1 item (long value) 2 = 4 bytes are 1 float (future)
2 – 3	Data length	
4 – 7	Floating factor	

Table 17-1: Upload SDO

Rest of byte sequence is the data stream.
All bytes are transfered according to ‘ Representation of numbers’ sector.

Sub index 0 will upload the supported object entries.

Note:
In the event of a change in the recorder variables while uploading, the process will be aborted.

17.7 Object 0x2031 Binary Up / Down sequence (HS)

Purpose:
Uploading and downloading parameters to the RAM.

This object is used as a carrier to the data the is downloaded to the amplifier RAM memmory and then to the FLASH.

The Binary Upload and Download sequence is described in the Software Manual Appedndix B in details.

Object Description:

INDEX	2031H
Name	
Object Code	DOMAIN
Number Of Elements	
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	Optional
Access	WR
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

17.8 Object 0x2040 Coordinate System ID

Purpose:
A 7 bit identifier used to address a synchronized command like BG or ST to a pre-defined set of servo drives.

Object Description:

INDEX	2040H
Name	ID for synchronized commands
Object Code	VAR
Number Of Elements	-
Data Type	Unsigned8

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Rw
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

Note:

17.9 Object 0x2041 – Latched Free running Timer .

Purpose:
This object transmits the accurate 32 bits timer of the amplifier.
The 32 bits timer is a timer with 1 usec resolution that is updated once in a real time cycle.
The accuracy of the report is 1 microsecond and the resolution is real time resolution.
Real time resolution can be detected using WS[29] command.

Typical usage of this object is when the host wants to use a node as the system SYNC master. The amplifier can transmit this object upon a SYNC message that came from the host (or SYNC manager). The timer value can be transmitted to the whole bus nodes as the Time Stamp of the SYNC message. The nodes in the bus can adjust the internal timer to this value according to the previous SYNC message received time.

Please refer to Sync and Time Stamp in this manual.

Object Description:

INDEX	2041H
Name	Amplifier free running timer
Object Code	VAR
Number Of Elements	-
Data Type	UNSIGNED32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

17.10 Object 0x2050 MS Event Trigger

Purpose:
This object is used as trigger event (transmission type 254) when motion is done. Note that this object will be sent when motion is enabled.

Object Description:

INDEX	2050H
Name	Motion Done Event
Object Code	VAR
Number Of Elements	-
Data Type	Unsigned32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

Note:
When object is transmitted the expected message should be:

Byte 1							Byte 8
				00	00	0x50	0x20

17.11 Object 0x2051 EI Event Trigger

Purpose:
This object is used as trigger event (transmission type 254) when EI command is sent to the interpreter though the RS232 or User Program.

When the object is mapped to a TPDO, each time the interpreter receives EI=<VALUE> the TPDO is sent in the following format:

Byte 1							Byte 8
				VALUE	00	0x51	0x20

VALUE can be any number from 0 to 255. Different VALUES may help the user to transmit the object from a known locations or events with in the User Program.

Please refer to PDO Mapping in this manual for more information about unsynchronized event mapping.

Object Description:

INDEX	2051H
Name	Event instruction trigger
Object Code	VAR
Number Of Elements	-
Data Type	Unsigned32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	RW
PDO Mapping	Yes
Value Range	[0,255]
Mandatory Range	No
Default Value	No

Note:

- EI is a volatile with default value of 0.
- When requiring EI value either by RS232 or CAN the mapped PDO will not be transmitted. The user will get the last value that was set to EI.
- When this object is downloaded with a value, EI parameter is updated but the mapped PDO will not be transmitted.

17.12 Object 0x2052 Main Homing Event Trigger

Purpose:
This object is used as trigger event (transmission type 254) when homing sequence is finished.

Please refer to the Command Reference for more information about the HM command.

When this object is mapped to TPDO, the TPDO will be transmitted each time that HM[1] decreases from 1 to 0.

The format of the message:

Byte 1							Byte 8
				00	00	0x52	0x20

Please refer to PDO Mapping for more information about unsynchronized event mapping.

Object Description:

INDEX	2052H
Name	Main Homing Done event trigger
Object Code	VAR
Number Of Elements	-
Data Type	Unsigned32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Note:
This object is valid when used in mapping procedure. The object returns no value when upload SDO is required.

17.13 Object 0x2060 Reference table

Purpose:
Used only by Elmo Composer Wizard in the process of system tuning.

Object Description:

INDEX	2060H
Name	Reference table
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	W
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

Note:

17.14 Object 0x206A Error Code Value (EC command)

Purpose:
Read the error code.

Object Description:

INDEX	206AH
Name	Error Code
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

Note:
Please refer to the Command reference – EC Command for more information about the available error codes.
When receiving a general abort message (080000h) an extra information regarding this error will be given with 0x206A object.

17.15 Object 0x2070 List/Download a program

Purpose:
Uploading this object will list the current user program.
Downloading this object will place a user program into RAM memory.

Object Description:

INDEX	2070H
Name	Read / Write user program
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	RW
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

17.15.1 User Program Upload Process

Initiate SDO Upload protocol according to [2]. After confirmation, a character stream is transmitted. Each octet has to be answered according to Upload SDO segmented Protocol.
The last byte in the last octet will be 0x08.

17.15.2 User Program Downloading Process

Initiate SDO Download protocol according to [2]. After confirmation, a character stream lead by 'D','L' according to the user program, is received. Each octet must be answered according to Download SDO segmented Protocol.
Download of a new program must start with the CP command (binary interpreter) that clears porgram counter.
In case that a stop durring downloading is required '\' (ASCII 0x5C) is used to the terminate transmission.

Note:
Before executing program a compilation must be issued. (object 0x2073).
Upload and download are from and to the RAM memory of the controller.
Use binary interpreter SG command to save the program to FLASH memory and LG command to load from FLASH to RAM.

17.16 Object 0x2071 Upload/Download Code Status

Purpose:
Uploading this object will return user program current line number and code.
Downloading this object will perform the current command and advance to the next one.

Object Description:

INDEX	2071H
Name	Code status/Step single
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Rw
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

17.16.1 Code Status Upload Process

Initiate SDO Upload protocol according to [2]. After confirmation a character stream is transmitted. Each octet must be answered according to Upload SDO segmented Protocol.
Segmented upload stream will include line number and code.

17.16.2 Single Code Download Process

Initiate SDO Download protocol using the expedite format according to [2]. One command of a user program will be performed and a confirmation will be sent.

Note:
One program line may include several commands separated by ';'. Upload transmits a whole line and download performs 1 command.

17.17 Object 0x2072 Program Message (MZ command)

Purpose:
Upload this object to fetch on-the-fly program messages.
The program message buffer can be used on 2 occasions:
1. Run time errors.
2. Run time messages (after MG)

Object Description:

INDEX	2072H
Name	Get program message.
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:
Initiate SDO Upload protocol according to [2]. After confirmation a characterstream is being transmitted. Each octet must be answered according to Upload SDO segmented Protocol.
If no message or errors are found in the buffer, an expedite confirmation will be transmitted with the data 0x1A (^Z) , 'O', 'K'.

In case of a message the following format will be use:

Line number	Space (ASCII 0x20)	Message String	';' (ASCII 0x3B)

All characters are visible string.

In case of an error the following format will be use:

Line number	Space (ASCII 0x20)	Command string	Space (ASCII 0x20)	'$' (ASCII0x24)	Elmo error code.	';' (ASCII 0x3B)

All characters are visible string.
Note:
Object might report up to 50 errors.
Last character in a message or error line is 0x3B (;).
Last character in a message and/or error list is 0x1A (^Z).

17.18 Object 0x2073 Program Compilation

Purpose:
Compile and upload error list.

Object Description:

INDEX	2073H
Name	Compile user's program.
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	Mandatory-
Access	Ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:
Initiate SDO Upload protocol according to [2]. After confirmation a characterstream is transmitted. Each octet must be answered according to Upload SDO segmented Protocol.
If no error is detected in the last compilation, only an expedite confirmation will be transmitted with the data 0x1A (^Z) , 'O', 'K'.
In case of an error the following format will be use:

Line number	Space (ASCII 0x20)	Command string	Space (ASCII 0x20)	'$' (ASCII 0x24)	Elmo error code.	';' (ASCII 0x3B)

All characters are visible string.

Note:
No program execution is allowed without prior compilation.
This object may report up to 50 errors.
The last character in an error line is 0x3B (;).
The last character in an error list is 0x1A (^Z).

17.19 Object 0x2074 Execute User Program.

Purpose:
Execute user program.

Object Description:

INDEX	2074H
Name	Executes user's program.
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	Mandatory-
Access	Wo
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:
Initiate SDO Download protocol according to [2].
Execution format is according to Software Manual.

The usage with **expedite** format execution will start the program from the first command.
The format for an **expedite** data should be:

Byte 1				Byte 5			Byte 8
0x22	0x74	0x20	0x00	0x00	0x00	0x00	0x00

Byte 5 – byte 8: Reserved for future usage – considered as 0.

The bit of the size indicator is not significant for this object. In that manner 0x22 or 0x23 give the same results.

In order to start from a referenced label use the **segmented** format.

! Up to 68 characters can be used for total labeling of a segmented data format. This includes 2 messages of 32 characters and the '##' signs.

Please refer to Program Development and Execution in the Software Manual.

Example:
We want to execute XQ##FIRST_LAB

The format for **segmented** data should be:

First the host initiates SDO sequence download:

Byte 1				Byte 5			Byte 8
0x20	0x74	0x20	0	0	0	0	0

After the reply from the amplifier the host transmits:

Byte 1				Byte 5			Byte 8
00	'#'	'#'	'F'	'I'	'R'	'S'	'T'

After the replay from the amplifier the final host transmission is:

Byte 1				Byte 5			Byte 8
0x17	'_'	'L'	'A'	'B'	0	0	0

The execution of a program can abort due to run time error. Please refer to XQ command in the command reference for more details.

Note:

- Execution is available after uploading object 0x2073 that is actually the compilation command..
- Run time error can be drawn using object 0x2073.

17.20 Object 0x2080 CPU Dump

Purpose:
This object holds the CPU status.

Object Description:

INDEX	2080H
Name	Dump CPU status
Object Code	VAR
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:

17.20.1 CPU Dump Upload Process

Initiate SDO Upload protocol according to [2]. After confirmation character stream is transmitted.
Each octet must be answered according to Upload SDO segmented Protocol.
Segmented upload stream will include:
Null Address – Address that caused a NULL interrupt
Failure Address – Address that caused a failure for stack overflow
Called handler – Exception handler after a failure.
Database Fail – Cause of failure in database tests.
Digital in – Status of digital input.

Note:
Recroder must be invalidate before using this object.

17.21 Object 0x2081 CAN Controller Status

Purpose:
This object informs the CAN controller status register.

Object Description:

INDEX	2081H
Name	CAN Controller Status
Object Code	VAR
Number Of Elements	-
Data Type	UNSIGNED8

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

The CAN controller status register bits:

7	6	5	4	3	2	1	0
Bus Off	Warning	Wake Up	Reserved	Reserved	L. Error 2	L. Error 1	L. Error 0

Bus Off:
When this bit is set (“1”):
There is an abnormal rate of occurences of errors on the CAN bus. The error counter of the controller has reached 256 error frames. During bus off the controller can nither receive nor transmits messages. The CPU starts a recovery sequence automatically, however the bus off state will be recovered after the controller has sensed 128 packages of 11 consecutive recessive bits (“1”). After recovary the controller resume to a normal operation.

Warning Level:
When this bit is set (“1”):
There is an abnormal rate of occurences of errors on the CAN bus**.** The error counter of the controller has count 96 error frames. The controller stays in normal operation.

Wake:
For sleeping mode – not implemented.

Last Error0-2:
This field contains a code which includes the first error to occur in a frame on the CAN bus.
0 – No errors
1 – Stuff Error:
More then 5 bits in a sequence have occurred in a part of a received message.
2 – Form Error:
Result from a violation of the fixed form in the following bit field:
- end of frame.
- interframe space.
- acknowladge field
- CRC

3 – ACK:
Message that was sent by this controller was not acknowladged by any other node.
4- Bit 1 Error:
Dominate bit was sensed while recesive one was sent.
5 – Bit 0 Error:
Recessive bit was sensed while dominate one was sent.
6 – CRC:
The CRC checksum was incorrect in the received message.
7 – Initiation value set by the CPU.

17.22 Object 0x208A - Begin Time

Purpose:
This object receives an absolute time for synchronized Motion Begin (BG).
Please refer to BT command in the Command Reference.

Please refer to Sync and Time Stamp in this manual for more information about the synchronization process.

Object Description:

INDEX	208AH
Name	Begin Time
Object Code	VAR
Number Of Elements	-
Data Type	UNSIGNED32

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	RW
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

17.23 Object 0x2090 Firmware Download

Purpose:
Download controller firmware using S-records format.

Please refer to Firmware Downloading in this manual.

Object Description:

INDEX	2090H
Name	Download firmware
Object Code	DOMAIN
Number Of Elements	-
Data Type	Visible String

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Wo
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:
The characters should be sent according to the S Record format. Status of downloading process can be read by object 0x2091 when the process failed or finished.

Format of the Messages:
After the final character of the each S Record line, the host must be send the character '0x0A' for end of line indication. Next S Record line can be sent immediately after '0xA'.

Example:

First the host initiates SDO segmented sequence:

Byte 1				Byte 5			Byte 8
20	0x90	0x20	0x00	0x00	0x00	0x00	0x00

After the reply from the amplifier, the host transmits:

Byte 1				Byte 5			Byte 8
00	'S'	'3'	'6'	'A'	'3'	'9'	'1'

The end-of-line format and the beginning of new S Record line:

Byte 1				Byte 5			Byte 8
10	'8'	'5'	0x0A	'S'	'6'	'0'	'B'

Note:
Applicable only if working firmware is installed.
In case of a failure, download can only be performed with RS232.

17.24 Object 0x2091 Firmware Downloading Status

Purpose:
Gives the status of firmware while and after downloading.

Object Description:

INDEX	2091H
Name	Status of firmware download
Object Code	VAR
Number Of Elements	-
Data Type	Unsigned8

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Byte stream:
This object returns a single ASCII character, which refers to the status of firmware download.
Please refer to Firmware Downloading in this manual.

17.25 Object 0x20A0 Auxiliary Position Actual Value.

Purpose:
Return the actual position of the auxiliary axis (PY).

Object Description:

INDEX	20A0H
Name	Auxiliary position.
Object Code	VAR
Number Of Elements	-
Data Type	Signed32

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	Yes
Value Range	±2^31
Mandatory Range	No
Default Value	No

Note:

17.26 Object 0x20A1 MainPosition Error.

Purpose:
Returns the error between the postion command and the actual position (PE).

Object Description:

INDEX	20A1H
Name	Position Error.
Object Code	VAR
Number Of Elements	-
Data Type	Signed32

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	Yes
Value Range	±2^31
Mandatory Range	No
Default Value	No

Note:

17.27 Object 0x20B1 Velocity Factor.

Purpose:
The value for translating the velocity internal units to cnt/sec.

Object Description:

INDEX	20B1H
Name	Velocity factor
Object Code	VAR
Number Of Elements	-
Data Type	float

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Note:
Velocity factor depends upon the TS values of the controller.
The result of mapped position objects like 0x6064 (PX) is in internal units, user should multiplie this value with the factor to get a counts / sec values.

17.28 Object 0x20B1 Current Factor.

Purpose:
The value for translating the current internal units to Ampere.

Object Description:

INDEX	20B2H
Name	Current factor
Object Code	VAR
Number Of Elements	-
Data Type	Float

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Note:
Current factor depends on the maximum current of the amplifier (MC).
The result of mapped current objects like 0x6078 (IM) is in internal units, user should multiplie this value with the factor to get values in ampere.

17.29 Object 0x2F00 Integer Array (IA).

Purpose:
Gets the value of integer array. The IA[] is a general purpose array. Values can be set to this array using the binary interpreter or with user program.

Object Description:

INDEX	2F00H
Name	Integer Array
Object Code	ARRAY
Number Of Elements	-
Data Type	Signed32

Values Description:

Sub-Index	-
Description	-
Object Class	-Optional
Access	Rw
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Example:
For IA[1]= 12345678 ; (decemal value)

Note:
This object may be useful when mapping TPDO to MS trigger (0x2050). The object 0x2050 will be transmit each time the motion ends in position mode. The problem is that the user can not tell the reason to the end-of-motion. This may be after a homing was done, or a switch was detected or a real target was reached. Mapping IA[] to the same object, can give indication for the reason that 0x2050 was launched.

17.30 Object 0x2F0A - Amplifier Free running Timer .

Purpose:
This object transmits the accurate 32 bits timer of the amplifier.
This data can be used in order to synchronized a time begin motion.

Please refer to Object 0x208A - Begin Time in this maual.

Please refer to Sync and Time Stamp in this manual.

Object Description:

INDEX	2F0AH
Name	Amplifier free running timer
Object Code	VAR
Number Of Elements	-
Data Type	UNSIGNED32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

17.31 Object 0x2F0B Status Object.

Purpose:
Status information of the following:
Bit 0 : Motor On . (MO=1)
Bit 1: Motor fault (MF > 0)
Bit 2: Program is running (PS > 0)
Bit 3: PTP in place.
Bit 4: Position profiler done.

Object Description:

INDEX	2F0BH
Name	Status Object
Object Code	VAR
Number Of Elements	-
Data Type	Signed32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

Note:
Status information is available in object 0x1002 (SR command).

17.32 Object 0x2F11 – PVT Head Pointer .

Purpose:
This object informs the host the index location of the last updated PVT message in the PVT table. According to that and with object 0x2F12the host can know the rate and the location in which the PVT table should be updated.
This object can be only used when mapped into a synchronized TPDO. It cannot be read or write with SDO protocol.

Please refer to PVT and PT – Tabulated Motion Modes in the Software Manual.

Object Description:

INDEX	2F11H
Name	PVT head pointer
Object Code	VAR
Number Of Elements	-
Data Type	UNSIGNED32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

17.33 Object 0x2F12 – PVT Tail Pointer .

Purpose:
This object informs the host the index of the next read PVT message in the PVT table. According to that and with object 0x2F11 the host can know the rate and the location in which the PVT table should be updated.
This object can be only used when mapped into a synchronized TPDO. It cannot be read or write with SDO protocol.

Please refer to PVT and PT – Tabulated Motion Modes in the Software Manual.

Object Description:

INDEX	2F12H
Name	PVT Tail pointer
Object Code	VAR
Number Of Elements	-
Data Type	UNSIGNED32

Values Description:

Sub-Index	-
Description	-
Object Class	-
Access	Ro
PDO Mapping	Yes
Value Range	No
Mandatory Range	No
Default Value	No

18 Error Control Protocol

18.1 Node Guarding and Life Guarding

(CiA DS-203-2 5)
Elmo amplifiers support node guarding and life factor.
The network master must issue an RTR (Remote transmission request) for the COB ID (0x700+device ID). That means that the first amplifier with the unit ID of 1 will have the node guarding COB ID of 1793.
The amplifier will respond with the single byte message of which bit 7 is a toggle (varies with each transmission) and the other 6 bits reflect the communication state of the amplifier.
Toggle bit reset value is 0.

The states read:

1	Disconnected
4	Prepared
5	Operational.

The amplifier can be programmed to shut itself off when the master fails to ask for the node guard object in time. If the amplifier waits (Guard time * Life time factor) milliseconds without detecting node guarding activity, it shuts itself off. This means that an automatic ST command is issued the communication resets and goes to the prepared state. The guard time is object 0x100c in the OD, and the life time factor is object 0x100d in the OD.
If either the guard time or the life time factor are zero, the auto shut off feature of the amplifier (the life guarding) is disabled.

18.2 Node Guarding and Life Guarding

(CiA DS-203-2 5)
Elmo servo drives support node guarding and life factor.
The network master must issue an RTR (Remote transmission request) for the COB ID (0x700+device ID). That means that the first servo drive with the unit ID of 1 will have the node guarding COB ID of 1793.
The servo drive will respond with the single byte message of which bit 7 is a toggle (varies with each transmission) and the other 6 bits reflect the communication state of the servo drive.
Toggle bit reset value is 0.

The states read:

1	Disconnected
4	Prepared
5	Operational.

The servo drive can be programmed to shut itself off when the master fails to ask for the node guard object in time. If the servo drive waits (Guard time * Life time factor) milliseconds without detecting node guarding activity, it shuts itself off. This means that an automatic ST command is issued the communication resets and goes to the prepared state. The guard time is object 0x100c in the OD, and the life time factor is object 0x100d in the OD.
If either the guard time or the life time factor are zero, the auto shut off feature of the servo drive (the life guarding) is disabled.

19 Firmware Downloading

New firmware versions can be loaded via the CAN communications by writing the new firmware as S-Records to object 0x2090. **Note:** that unlike, RS232, firmware downloading by CAN communication may only be used to download firmware if the servo drive firmware is not corrupted[6]. Since RS232 communications are supported by the BIOS, CAN downloading require the existing firmware to run.
Firmware loading is protected – the parameter TP[6] must be set to 1 before firmware downloading can be done. The S records are written as string SDO s to object 0x2090. After each completed S Record, the status may be read via object 0x2091.

Object 0x2091 reads:

O.k	'A'	
Not completed	'B'	S record transfer in process. This state is set after download initiation.
General download failure	'C'	Download failure due to unknown failure. 'Abort SDO Transfer' according to [2] will be transmitted.
Flash error	'D'	Download failure due to error in the flash. 'Abort SDO Transfer' according to [2] will be transmitted.
Programming complete	'F'	Complete download successfully. This state is set after a complete message was arrived. This state will remain until NMT application or controller reset.

After firmware downloading, the servo drive continues to communicate using the old firmware. If the downloading fails, it is possible to retry the downloading. In order to flag that the firmware loading should start from the beginning, set TP[6]=1.
After a successful firmware loading, the NMT command reset application (129) must be issued to reboot the servo drive. Power reset will provide the same result.

[6] This means that if a firmware download is interrupted prematurely, we must reload the firmware to the servo drive using RS232 communication

20 Initial Setup for CAN Communication

All the communication parameters, such as the CAN baud rate for the targets, are programmed via the PP[N] command – please see [6]. In order to program the communication parameters, we need to first communicate with the servo drive. The CAN communication parameters may be set using the RS232 communication channel, which is always active. In the case where the servo drive is programmed to RS232 parameters that are different from the default, there is a way to force the servo drive to its default RS232 communication parameters – please see the user manual [6].It is also possible to try all the supported CAN baud rates until the unit responds.

The following parameters affect CAN communication:

	Description	Range
PP[13]	Servo drive CAN ID.	1 – 127
PP[14]	CAN baud rate.	1 – 7 7 for 10000 6 for 20000 5 for 50000 4 for 100000 3 for 125000 2 for 250000 1 for 500000
PP[15]	Standard/Extended arbitration field.	Must be zero. (Only standard arbitration is supported by the software at present).

Table 20-1: CAN Communication

Setting the PP[13] and the PP[14] parameters does not change anything immediately. In order to activate the new communication parameters, an NMT reset communication command is required, or the servo drive must be re- booted.
If the servo drive is to be re-booted, either by the NMT reset application command or by a power-on sequence, the SV (Save parameters) command must be used so that the communication parameters will become permanent.

In order to start CAN PDO communication, an NMT network start command must be issued – please refer the above section on NMT.

After the net start command, the servo drive is in the CAN operational state.