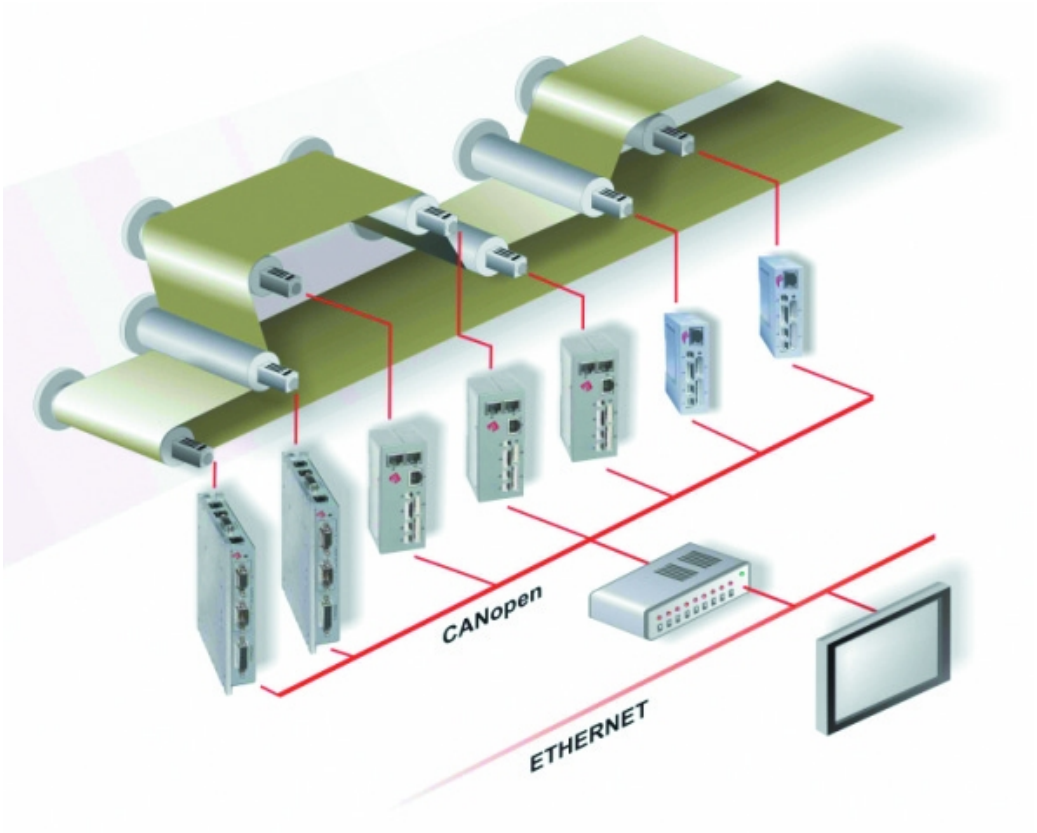
Elmo Motion Control
CANopen DS 301
Implementation Guide



January 2005

Elmo
Motion Control

Important Notice

This guide is delivered subject to the following conditions and restrictions:

- This guide contains proprietary information belonging to Elmo Motion Control Ltd. Such information is supplied solely for the purpose of assisting users of *SimplIQ* servo drives in implementing CANopen networking.
- The text and graphics included in this manual are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.
- Information in this document is subject to change without notice. Corporate and individual names and data used in examples herein are fictitious unless otherwise noted.

Doc. No. MAN-CAN301IG

Revision History

Ver. 1.7 January 2005 0xA6 error coded added, 0x2F21 bit 9 emergency even added, RPDO2 Structure Example 6 modified.

Ver. 1.6 December 2004 References to Harmonica changed to *SimplIQ* (MAN-CAN301IG)

- New objects: 0x2202 (Digital Input), 0x2F00 (user integer), 0x2F01 (user float array), 0x2F02 (ET Array)
- In objects 1800 to 1803 Sub Index 1 is Read Only
- New Protocol for the object 1018 sub index 3

Ver. 1.5 September 2003 Changes:

- Objects added to "Data Type"
- Enhanced PDO mapping in PDO and Object Dictionary sections, and objects 0x1400-4, 0x1600-4, 0x1800-4 and 0x1A00-4
- PDO mapping defined in object 0x1600
- EDS storage objects removed from manual
- New objects: 0x2200 (digital input), 0x2F15 (profile position remaining points), 0x2F40 (configuration object)
- DSP-402 objects transferred to DSP-402 manual
- Event time for TPDO defined in object 0x1800

Ver. 1.0 Jan. 2003 Initial Release (CANIGHA0103)

Elmo Motion Control Inc.
1 Park Drive, Suite 12
Westford, MA 01886
USA
Tel: +1 (978) 399-0034
Fax: +1 (978) 399-0035

Elmo Motion Control GmbH
Steinbeisstrasse 41
D-78056, Villingen-Schwenningen
Germany
Tel: +49 (07720) 8577-60
Fax: +49 (07720) 8577-70

www.elmomc.com

Contents

Chapter 1: Introduction 1
1.1 Relevant Documentation 1
1.1.1 Elmo Documentation 1
1.1.2 CAN Documentation 2
1.2 Abbreviations and Terms 2
1.3 SimplIQ Communication 3

Chapter 2: CANopen Basics 5
2.1 Physical Layer 5
2.2 Standard vs. Extended Addressing 5
2.3 Client - Server Relations 5
2.4 Inhibit Times 6
2.5 RTR – Remote Transmission Request 6
2.6 Object Dictionary 6
2.7 Communication Objects 7
2.8 Object Dictionary - Data Types 8
2.9 Representation of Numbers 11

Chapter 3: The Object Dictionary 12

Chapter 4: Service Data Objects (SDOs) 17
4.1 Initiate SDO Download Protocol 18
4.2 Download SDO Protocol 19
4.3 Initiate SDO Upload Protocol 20
4.4 Upload SDO Segment Protocol 21
4.5 Abort SDO Transfer Protocol 22
4.6 Uploading Data Using an SDO 23
4.7 Downloading Data Using an SDO 24
4.8 Error Correction 24

Chapter 5: Process Data Objects (PDOs) 26
5.1 Receive PDOs 26
5.2 Transmit PDOs 28
5.3 PDO Mapping 28
5.3.1 The Mapping Trigger – Transmission Type 28
5.3.2 The Synchronous Trigger 29
5.3.3 The Asynchronous Trigger 29
5.3.4 RPDO Error Handling 30
5.3.5 Mapping Parameter Objects 30
5.3.6 Default Values 31

Chapter 6: Emergency (EMCY) 34
6.1 Emergency Configuration 34
6.2 Emergency Codes Related to Failure 34
6.3 Emergency Codes for Motor Faults 34
6.4 Emergency Codes Related to PVT/PT Motion 34

Chapter 7: Network Management (NMT) 36

Chapter 8: SYNC and Time Stamp 37

Chapter 9: Binary Interpreter Commands 38
9.1 Binary Interpreter Commands and Results 39
9.1.1 Set and Query Commands 39
9.1.1.1 RPDO2 Structure 39
9.1.1.1 TPDO2 Structure 41
9.1.2 Execute Command 42
9.2 ASCII Interpreter Commands not Supported by Binary Interpreter 42

Chapter 10: The OS Interpreter 43

Chapter 11: The EDS 45

Chapter 12: Communication Profile 46
Object 0x1000: *Device type* 46
Object 0x1001: *Error register* 47
Object 0x1002: *Manufacturer status register* 48
Object 0x1003: *Pre-defined error field* 48
Object 0x1005: *COB-ID SYNC message* 50
Object 0x1008: *Manufacturer device name* 51
Object 0x1009: *Manufacturer hardware version* 51
Object 0x100A: *Manufacturer software version* 52
Object 0x100B: *Node ID* 52
Object 0x1010: *Save parameters* 53
Object 0x1011: *Restore parameters* 54
Object 0x1012: *COB-ID time stamp* 55
Object 0x1013: *High-resolution time stamp* 56
Object 0x1014: *COB-ID emergency object* 56
Object 0x1016: *Consumer heartbeat time* 57
Object 0x1017: *Producer heartbeat time* 58
Object 0x1018: *Identity object* 59
Note: 60
Object 0x1023: *OS command and prompt* 61
Object 0x1024: *OS command mode* 62
Object 0x1029: *Error behavior* 63
Object 0x1200: *SDO server parameter* 64
Objects 0x1400 - 0x1403: *Receive PDO communication parameter* 65
Objects 0x1600 - 0x1603: *Receive PDO mapping* 67
Objects 0x1800 - 0x1803: *Transmit PDO communication parameter* 68
Objects 0x1A00 - 0x1A03: *Transmit PDO mapping* 71

Chapter 13: Manufacturer-specific Objects 72
Object 0x2001: *PVT data* 73
Object 0x2002: *PT data* 74
Object 0x2004: *ECAM data* 75
Object 0x2012: *Binary interpreter input* 75
Object 0x2013: *Binary interpreter output* 76
Object 0x2030: *Recorder data* 76
Object 0x2040: *Coordinate system group ID* 82
Object 0x2041: *Amplifier-free running timer* 83
Object 0x2082: *CAN controller status* 83

Object 0x208A: *Begin time* 85
Object 0x2090: *Firmware download* 86
Object 0x20A0: *Auxiliary position actual value* 87
Object 0x20A1: *Main position error* 87
Object 0x2200: *Digital input* 88
Object 0x2201: *Digital input low byte* 89
Object 0x2F00: *User Integer* 90
Object 0x2F01: *User Float Array* 90
Object 0x2F02: *ET Array* 91
Object 0x2F11: P*VT head pointer* 92
Object 0x2F12: *PVT tail pointer* 92
Object 0x2F15: *Profile position remaining points* 93
Object 0x2F20: *PDO events* 94
Object 0x2F21: *Emergency events* 96
Object 0x2F22: *Bus off time out* 100
Object 0x2F23: *Digital input TPDO event parameters* 101
Object 0x2F30: *Last time stamp correction* 103
Object 0x2F31: *Last SYNC time* 103
Object 0x2F40: *Configuration object* 104

Chapter 14: Error Control Protocol 105

Chapter 15: Downloading Firmware 106

Chapter 16: Initial CAN Communication Setup 107
16.1 Setup Using RS-232 107
16.2 Bootup Protocol 108

Appendix: Little and Big Endians 106

Chapter 1: Introduction

This manual explains how to implement CANopen DS 301 communication with Elmo's *SimplIQ* DSP-based digital servo drives. It provides a description of *SimplIQ* drives and the means of implementing communication based on the CiA CANopen protocols.

Most *SimplIQ* functionality is standard, according to CiA documents DS 301, version 4.01, DSP 402 (proprietary) and the CiA OS interpreter. In this document, emphasis is placed on manufacturer-specific behaviors, although it also repeats certain CiA standard material, to enhance understanding and to complete certain descriptions. The manual does not contain all relevant CiA information, indicating that many objects are implemented, but are not documented herein. The user should therefore complement this manual with the CiA documents outlined in the following section.

1.1 Relevant Documentation

1.1.1 Elmo Documentation

This manual is part of the Elmo *SimplIQ* documentation set, as outlined in the following diagram:

Programming

CANopen Implementation Guide
SimplIQ Software Manual
SimplIQ Command Reference Manual

Setup

Composer User Manual

Installation

Cello Installation Guide
Basssoon Installation Guide
Harmonica Installation Guide

In addition to this document, the SimplIQ documentation set includes:

- The *SimplIQ Installation Guide,* which provides full instructions for installing the *SimplIQ* digital servo drives
- The Composer *User Manual,* which includes explanations of all the software tools that are a part of Elmo's Composer software environment
- The *SimplIQ Command Reference Manual,* which describes, in detail, each software command used to manipulate the SimplIQ motion controller.

  This is the main source of detailed explanations of all *SimplIQ* commands mentioned in this manual.
- The *SimplIQ Software Manual,* which describes the comprehensive software used with *SimplIQ* digital servo drives

1.1.2 CAN Documentation

Document Name	Author	Source
CAN Implementation Guidelines	Gruhler G. and Drier B.	STA Reutlingen
CiA DS 301 V 4.01: *CANopen Communication Profile for Industrial Systems - based on CAL*		CiA
CiA DS 302 V 3.0: *CANopen Framework for Programmable Devices*		CiA
CiA DS 305 V 1.0: *CANopen Layer Setting Services and Protocol (LSS)*		CiA
CiA DSP 402 V 2.0: *CANopen Device Profile*		CiA
CiA DS 202-2 V 1.1: *CAN Application Layer (CAL) - CMS Protocol Specification*		CiA

1.2 Abbreviations and Terms

The following **abbreviations** are used in this manual:

CAL	CAN application layer.
CMS	CAN message specification.
COB	Communication object; a CAN message.
ID	Identifier; 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 COB.

EDS	Electronic data sheet; a standard form of all CAN objects supported by a device. The EDS is used by external CAN configurators.
LSS	Layer setting service: methods for configuring the ID and baud rate of a slave, using the standard DSP 305.
OD	Object dictionary, which is the full set of objects supported by the node. It is the interface between the application and communication (see "Object" below.)
PLC	Programmable controller. A PLC can serve as a CAN master for *SimplIQ* digital servo drives.

The following **terms** are used in this manual:

CAN client or CAN master	A host — typically a PC — or other control equipment that supervises the nodes of a network.
CAN server or CAN slave	A node in the CAN network that can give service under control of the CAN master.
LSB	Least Significant Bit (or Byte)
MSB	Most Significant Bit (or Byte)
Object	A CAN message with a meaningful functionality and/or data. Objects are referenced according to addresses in the object dictionary.
Receive	In this manual, "received" data is sent from the control equipment to the servo drive.
Transmit	In this manual, "transmitted" data is sent from the servo drive to the other equipment.

1.3 *SimplIQ* Communication

SimplIQ digital servo drives support two types of serial communication:

- RS-232
- CANopen

SimplIQ digital servo drives can simultaneously communicate using both CAN and RS-232 communication lines, which are always open for communication. The communication parameters are set using the PP command.

The following table compares the main features of both communication modes, as implemented with Elmo SimplIQ digital servo drives:

Features	**CANopen**	**RS-232**
Baud rate	50,000 - 1,000,000.	4800 - 57,600 (RS-232).
Interpreter method	Binary or ASCII.	ASCII.
Fast referencing	Yes, for PVT, PT and ECAM.	No.
Network of servo drives	Yes.	No.
Multiple servo drive synchronization	Yes.	No.
Standardization	Compliant with CiA standard.	No standard.
Special equipment required	CAN communication interface (available as an add-on ISA or PCMCIA card for PCs) with appropriate software.	No: direct connection to serial port of PC.
Ease of use	Basic capabilities included in Elmo Composer program.	Immediate: Just type command using HyperTerminal or equivalent terminal software.

Table 1-1: ***SimplIQ* Communication Types**

RS-232 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 online reference generation, required for supporting complex motions
- Binary interpretation, which maximizes servo-drive command throughput by eliminating servo drive software overhead
- Servo network applications

To benefit from 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.

Chapter 2: CANopen Basics

This chapter describes — in general — the CANopen communication features most relevant to Elmo *SimplIQ* servo drive. More detailed information is available in the specific CANopen documentation.

2.1 Physical Layer

CAN is a serial communication standard in which the transferred data is coded as electrical pulses on a two-wire communication line. 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. *SimplIQ* digital servo drives use a CAN controller built into the drive DSP.

2.2 Standard vs. Extended Addressing

Each CAN message frame includes an arbitration field that defines the type of data sent and its address. CAN version 2.0A supports 11 arbitration bits for this purpose; the seven least significant define the address and the four most significant define the type of message sent. Only 16 message types are supported. CAN version 2.0B supports 29 arbitration bits, of which the seven least significant define the address and 21 bits define the message type. In CiA DS 301, the arbitration bits indicate the object and the node-ID, together comprising the COB-ID.

CAN communication is prioritized so that messages with higher priority are transmitted first. The arbitration field determines the message priority: The lower the number in the arbitration field, the higher the message priority. ID 0 gives the highest priority. The *SimplIQ* drives support the CAN version 2.0A (11-bit) addressing method only, meaning that it ignores messages of 29 bits. A setup parameter (PP[13] - Node ID) selects the CAN object identification to be used.

2.3 Client - Server Relations

A CAN master (or client) is a controller that makes requests to nodes to respond to its commands. A CAN slave (or server) responds to the commands issued by the CAN master. The CAN protocol permits both single-master and multiple-master networks.

The *SimplIQ* servo drives assume a single-master network arrangement, in which the servo drives are the slaves 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 require an ID. As a slave, the servo drive never sends an unrequested message, other than emergencies. The drive responds only to messages addressed to its ID or to broadcast messages, which have an ID of 0. All messages sent by a servo drive are marked with its own ID.

 If two servo drives have been assigned the same ID, the CAN network may crash.

2.4 Inhibit Times

The inhibit time for a given message type is the minimum time that must elapse from the time the message is first transmitted until the time that it may be transmitted again. The purpose of inhibit times is to ensure that high-priority messages do not flood the bus and thereby prevent service messages of lower priority from being transmitted. The inhibit times of *SimplIQ* drives are defined only for asynchronous TPDOs, by sub-index 3 of the associated transmission type object.

The resolution of the inhibit time is 100 microseconds, with an accuracy of 2 milliseconds.

If several events occur during the inhibit time of a message, only the last message will be transmitted when the inhibit time extinct . For example, if TPDO3 is activated by digital input 1 and digital input 3 with an inhibit time of 10 milliseconds, then if a digital input 1 event emits TPDO3 at time 0, a digital input 3 event occurring 5 milliseconds later will not cause any TPDO3 transmission for the next 5 millisecond. The TPDO3 transmission would be postponed, allowing other messages to be transmitted, until the inhibit time is exhausted.

The DS 301 version 4.0 protocol allows the user to modify the inhibit time only for COBs that are not valid (sub-index 1 of objects 0x1400 and 0x1800), while *SimplIQ* drives allow the user to modify the inhibit time at any time, with pending messages being discarded according to that time definition.

2.5 RTR – Remote Transmission Request

The RTR is not supported by the Elmo drive.

2.6 Object Dictionary

An object dictionary (OD) is a naming system that gives a unique identifier to each data item — or "object" —that is communicated over the CAN bus. 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. (An object's access permission may be read-only, write-only, or read-write.)

CiA DS 301 requires a set of mandatory objects for all CANopen devices. Other OD items are predefined by CiA DS 301 to have fixed identifiers, if supported. The OD also accommodates manufacturer-specific objects.

2.7 Communication Objects

The data-byte units transported through a CAN network are called communication objects (COBs). *SimplIQ* servo drive uses the following COB types:

COB Type	Description
Service data object (SDO)	SDO messages are used to manipulate OD objects according to their IDs. The server receives the SDO, which specifies in its message which object is to be dealt with. SDO messages can be chained to form a "domain transfer," which is useful for sending large data items such as long strings. Domain transfers are time-consuming, because the CAN bus is half-duplex. Each time a data segment is downloaded, a full-sized data segment is uploaded for verification, and vice versa.
Process data object (PDO)	PDO messages are used to manipulate OD objects without explicit reference to the object identifier, which is possible if there is an a-priori convention concerning the OD item referenced. Such conventions are called "PDO mappings"; these are actually OD objects themselves, and may be defined and manipulated using an SDO.
Emergency (EMCY)	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 first being explicitly asked. EMCY objects are similar to servo drive "interrupts": they eliminate the need to poll the servo drive continuously for the emergency status.
Network Management (NMT)	NMT objects are used by CAN clients to initialize a servo drive as a server.
Layer Setting Service (LSS)	This service is used to assign IDs and baud rates to newly-installed products.

Table 2-1: **Communication Objects Used by *SimplIQ* Servo Drives**

The type of COB transmitted is indicated in the arbitration field of the message, and thereby determines its priority. The relation between bits 8 to 11 of the arbitration field (COB-ID) and the COB type is presented in the following table:

COB Type	Bits 8 - 11 of COB-ID	ID Range
NMT	0000	0
SYNC	0001	128 (80h)
Time Stamp	0010	256 (100h)
Emergency	0001	129…255 (81h…ffh)
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)
PDO4 - Transmit	1001	1153…1279 (481h…4ffh)
PDO4 - 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 2-2: **COB Types**

Example:
The COB-ID of PDO1, when received by node #2, is binary 01000000010, which is decimal 514, or 202 hexadecimal. The IDs of the servo drives are set in the range 1…127.

2.8 Object Dictionary - Data Types

The Elmo CAN controller supports the following data types:

Index	Object	Name
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

Index	Object	Name
0040	DEFTYPE	PVT DataPar
0041	DEFTYPE	PT DataPar
0042	DEFTYPE	Binary interpreter query
0043	DEFTYPE	Binary interpreter command
0044	DEFTYPE	DSP 402 PV data record
0081	DEFTYPE	DSP 402 interpolated data configuration record

Table 2-3: **Data Types**

Note:
Data Objects 0002 to 0004 are used as "dummy" entries.

Communication Parameter Record Object 0x20

Index	Sub-index	Field in PDO Communication Parameter Record	Data Type
0020h	0h	Number of supported entries in record	UNSIGNED8
	1h	COB-ID	UNSIGNED32
	2h	Transmission type	UNSIGNED8
	3h	Inhibit time	UNSIGNED16
	4h	Reserved	UNSIGNED8
	5h	Event timer	UNSIGNED16

PDO Mapping Parameter Record Object 0x21

Index	Sub-index	Field in PDO Parameter Mapping Record	Data Type
0021h	0h	Number of mapped objects in PDO	UNSIGNED8
	1h	First object to be mapped	UNSIGNED32
	2h	Second object to be mapped	UNSIGNED32
…	…	…	…
	40h	64th object to be mapped	UNSIGNED32

The four device-specific data types used by *SimplIQ* digital servo drives are as follows:

PVT DataPar Object 0x40

MSB LSB

Time (UNSIGNED8)	Velocity (SIGNED24)	Position (SIGNED32)

PT DataPar Object 0x41

MSB LSB

Position 2 (SIGNED32)	Position 1 (SIGNED32)

Binary Interpreter Query Object 0x42

MSB LSB

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

For more information about the binary interpreter query, refer to Chapter 9.

Binary Interpreter Command Object 0x43

MSB LSB

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

For more information about the binary interpreter command, refer to Chapter 9.

DSP 402 Interpolated Mode PV Object 0x44

MSB LSB

Position 2 (SIGNED32)	Velocity (SIGNED32)

DSP 402 Interpolated Time Period Record Object 0x80

Index	Sub-index	Field in Interpolation Time Period Record	Data Type
0080h	0h	Number of entries	UNSIGNED8
	1h	Interpolation time units	UNSIGNED8
	2h	Interpolation time index	INTEGER8

DSP 402 Interpolated Data Configuration Record Object 0x81

Index	Sub-index	Field in Interpolation Time Period Record	Data Type
0081h	0h	Number of entries	UNSIGNED8
	1h	Maximum buffer size	UNSIGNED32
	2h	Actual buffer size	UNSIGNED32
	3h	Buffer organization	UNSIGNED8
	4h	Buffer position	UNSIGNED16
	5h	Size of data record	UNSIGNED8
	6h	Buffer clear	UNSIGNED8

2.9 Representation of Numbers

CAN communication delivers numerical data stored in binary form. Integers are stored by their binary representation, while floating-point numbers are stored according to the IEEE representation. *SimpliQ* digital servo drives support three types of data: short integers (two bytes), long integers (four bytes) and floating-point numbers (four bytes). These multiple-byte numbers are stored in the CAN messages according to CAN standards, using the "little endian" (Intel-type) convention. With this method, the number is inverted before storage: The most significant byte of the number receives the lowest address and the least significant byte receives the highest address. More information about the little endian method is provided in the Appendix of this manual.

Example:
The following is an 8-byte CAN message:

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

The CAN message data field will be as follows:

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

Chapter 3: The Object Dictionary

The object dictionary is essentially a grouping of objects that are accessible via receive and transmit SDOs. Part of the object can be mapped to transmit and receive PDOs (TPDO and RPDO, respectively) in a predefined manner.

The following layout is used with the objects in the object dictionary:

Index (Hex)	Object
0	Not used
0001 - 001F	Static data type
0020 - 003F	Complex data type
0040 - 005F	Manufacturer-specific data type
0060 - 0FFF	Reserved
1000 - 1FFF	Communication profile area
2000 - 2FFF	Manufacturer-specific profile area
6000 - 6FFF	Standardized device profile area
A000 - FFF	Reserved

The table that follows lists the objects supported by *SimplIQ* digital servo drives. Each object is addressed by a 16-bit index. Some of the objects may include 8-bit sub-indices, which are described in the object description. The object **Name** is that given by CiA or Elmo according to object type. **Access** can be R (read only), W (write only) or R/W (read/write). In the **Can be Mapped** column, N indicates that the object cannot be mapped to a PDO, while Y indicates that the object *can* be mapped to a PDO or that it is permanently mapped to a predefined PDO.

Name	Index	Description	Access	Mappable?
CAN controller type	0x1000	Device type and functionality. Constant value = 0x12D.	R	N
Error register	0x1001	Contains error information.	R	N
Manufacturer status register	0x1002	Returns status similar to SR command.	R	Y
Pre-defined error field	0x1003	Returns previous emergency history. Setting object 0x1003 sub-index 0 to 0 deletes error history.	R	N
COB-ID for SYNC message	0x1005	32-bit Dword, pre-defined.	R	N
Communication cycle period	0x1006	Spacing, in µsec, between consecutive SYNC signals. *This parameter is included for compatibility with the standard OD, but it is ignored.*	R/W	N

Name	Index	Description	Access	Mappable?
Manufacturer's device name	0x1008	String that returns the drive name such as "Harmonica"	R	N
Hardware version	0x1009	A string that conveys the information in WS[30].	R	N
Software version	0x100A	String that returns value of VR command.	R	N
Node ID	0x100B		R	N
Store parameters	0x1010	Stores parameters in flash memory.	R/W	N
Restore parameters	0x1011	Restore parameters from flash memory.	R/W	N
COB-ID for Time Stamp message	0x1012	Specifies COB-ID of Time Stamp message.	R	N
High-resolution Time Stamp	0x1013	Defines a Time Stamp message with a resolution of 1 µsec.	R/W	N
COB-ID of Emergency message	0x1014	Defines COB-ID of the Emergency object (EMCY).	R	N
Consumer heartbeat time	0x1016	Defines minimal acceptance rate for heartbeat messages.	R/W	N
Producer heartbeat time	0x1017	Defines rate of generating heartbeats.	R/W	N
LSS address	0x1018	Address for ID and baud rate configuration, according to DSP 305.	R	N
OS interpreter	0x1023	OS prompt interpreter.	R/W	N
OS command mode	0x1024	Mode command and status readout of OS interpreter.	R/W	N
Error behavior	0x1029	Defines behavior in a case of serious device failure, according to CiA DS 301.	R/W	N
SDO1 server	0x1200	SDO1: server parameter. (link)	R	N
PDO1 RX Comm.	0x1400	PDO1: receive communication parameter. (link)	R/W	N
PDO2 Rx Comm.	0x1401	PDO2: receive communication parameter.	R/W	N
PDO3 Rx Comm.	0x1402	PDO3: receive communication parameter.	R/W	N
PDO4 RX Comm.	0x1403	PDO4: receive communication parameter.	R/W	N

Name	Index	Description	Access	Mappable?
PDO1 Rx Map.	0x1600	PDO1: receive mapping parameter. (link)	R/W	N
PDO2 Rx Map.	0x1601	PDO2: receive mapping parameter.	R/W	N
PDO3 Rx Map.	0x1602	PDO3: receive mapping parameter.	R/W	N
PDO4 Rx Map.	0x1603	PDO4: receive mapping parameter. (link)	R/W	N
PDO1 Rx Comm.	0x1800	PDO1: transmit communication parameter.	R/W	N
PDO2 Tx Comm.	0x1801	PDO2: transmit communication parameter.	R/W	N
PDO3 Tx Comm.	0x1802	PDO3: transmit communication parameter.	R/W	N
PDO4 Tx Comm.	0x1803	PDO4: transmit communication parameter.	R/W	N
PDO1 Tx Map.	0x1A00	PDO1: transmit mapping parameter.	R/W	N
PDO2 Tx Map.	0x1A01	PDO2: transmit mapping parameter.	R/W	N
PDO3 Tx Map.	0x1A02	PDO3: transmit mapping parameter.	R/W	N
PDO4 Tx Map	0x1A03	PDO4: transmit mapping parameter.	R/W	N
PVT data	0x2001	Bytes describing PVT command.	W	Y
PT data	0x2002	Bytes describing PT command.	W	Y
Fast position	0x2003	Reserved for future real-time positioning modes.		
ECAM data	0x2004	Fast, auto-increment entry to ECAM table.	W	Y
Binary interpreter input	0x2012	Set binary interpreter command.	W	Y
Binary interpreter output	0x2013	Get binary interpreter command.	R	Y
Recorded data output	0x2030	Contains recorded data according to request in RC binary command.	R	N
Group ID	0x2040	Identifier used to address a group of drives for coordinated commands.	R/W	N
Amplifier free running timer	0x2041	Transmits accurate 32-bit timer of drive.	R	Y
CAN controller status	0x2082	CAN controller status register.	R	N

Name	Index	Description	Access	Mappable?
Begin on time	0x208A	Used to start a synchronized motion according to internal free running timer.	R/W	N
Firmware download	0x2090	Similar to DF command.	W	N
Auxiliary position actual value	0x20A0	Actual position as taken from auxiliary sensor input (PY).	R	Y
Position error	0x20A1	Position error as calculated from command and actual position value. (PE).	R	Y
Digital input	0x2200	Reflects the digital input (IP)	R	Y
Digital inputs low byte	0x2201	Reflected Negative limit switch, Positive limit switch and Home switch.	R	Y
User Integer	0x2F00	Provides an array of 24 integer numbers for general-purpose use.	R	Y
User Float Array	0x2F01	Provides an array of 24 floating numbers for general-purpose use.	R/W	Y
ET Array	0x2F02	Enables ECAM table variables (ET[1] to ET[255]) to be loaded.	R/W	Y
PVT buffer head pointer	0x2F11	PVT motion advance information.	R	Y
PVT buffer tail pointer	0x2F12	PVT motion advance information.	R	Y
Buffered PTP remained point	0x2F15	Used to detect number of remaining points in DSP 402 Profile Position buffered mode.	R	Y
Asynchronous PDO event	0x2F20	Defines which events emit a PDO.	R/W	N
Emergency event	0x2F21	Defines which events emit an emergency.	R/W	N
Bus-off timeout	0x2F22	Defines bus-off timeout, after which CAN communication will be renewed if *SimplIQ* enters bus-off.	R/W	N
Digital Input TPDO Event Parameters	0x2F23	Defines which digital input transmissions will activate a TPDO driven by a digital input event.	R/W	N

Name	Index	Description	Access	Mappable?
Last Time Stamp correction	0x2F30	Difference between last SYNC time and last Time Stamp. Serves to estimate how accurately the internal amplifier's clock is locked on the master clock.	R	Y
Internal µsec counter at last SYNC	0x2F31	Internal µsec counter of drive, sampled at last SYNC. Useful when one drive is used to synchronize entire network.	R	Y
Configuration object	0x2F40	Configure functionality of the node.	R/W	N

Table 3-1: **Object Dictionary**

Chapter 4: Service Data Objects (SDOs)

SimplIQ digital servo drives use a single transmit server SDO (COB 581h-6ffh) and a single receive server SDO (COB601h-67fh). This is according to CiA definitions and priority allocations for 11-bit addressing.

When using SDOs, it is important to remember that:

- An SDO has a lower priority than a PDO.
- An SDO session is not complete until it is confirmed.

For example, if an SDO is used to change a PDO mapping, the SDO should be issued only after the last session in which the PDO is completed, and the newly-mapped PDO should not be used until the SDO mapping change is confirmed.

SDOs implement the CMS multiplexed domain protocols.

Notes:

- In an SDO data exchange, each client message may be backed by one and only one server message.
- An SDO carries a toggle bit, which varies in every consecutive message of a domain transfer, so that the loss of a single message can be tracked.
- An SDO transfer can be terminated using the special "Abort domain transfer" message.
- An SDO message carries a maximum of seven bytes of data. One byte (the header byte) is always dedicated to overhead data.
- The length of an SDO message is always eight bytes, even if some of them are unused. Unused data bytes are marked as such in the message header.
- The maximum length of payload data in an expedited SDO is four bytes.

4.1 Initiate SDO Download Protocol

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

Client to server:

0				1	4	8
7…5	4	3…2	1	0		
css = 1	**x**	**n**	**e**	**s**	**m**	**d (data)**

Server to client:

0	1	4	8
7…5	4…0		
scs = 3	**x**	**m**	**reserved**

where:

css Client command specifier
1: Initiate download request

scs Server command specifier
3: Initiate download response

n Number of bytes in **d** that do not contain data. Only valid if **e** = 1 and **s** = 1; otherwise it is 0. 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. Represents index/sub-index of data to be transferred by SDO.

d Data
e = 0, **s** = 0: **d** is reserved for future use.
e = 0, **s** = 1: **d** contains number of bytes to be downloaded.
Byte 4 contains LSB and byte 7 contains MSB.
e = 1, **s** = 1: **d** contains data of length 4-n to be downloaded. The encoding depends on the type of data referenced by index and sub-index.
e = 1, **s** = 0: **d** contains an unspecified number of bytes to be downloaded.

x Not used; always 0.

reserved Reserved for future use; always 0.

4.2 Download SDO Protocol

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

Client to server:

0			1	8
7…5	4	3…1	0	
css = 0	**t**	**n**	**c**	**seg-data (segment data)**

Server to client:

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

where:

css Client command specifier
0: Download segment request

scs Server command specifier
1: Download segment response

seg-data Maximum seven bytes of segment data downloaded. Encoding depends on type of data referenced by index and sub-index.

n 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 Whether or not there are still more segments to be downloaded:
0: More segments to be downloaded
1: No more segments to be downloaded

t Toggle bit, which alternates for each subsequent segment to be downloaded. First segment has toggle-bit set to 0. Toggle bit is equal for request and response message.

x Not used; always 0.

reserved Reserved for future use; always 0.

4.3 Initiate SDO Upload Protocol

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

Client to server:

0	1	4	8
7…5	4…0		
css = 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**	**reserved**

where:

css Client command specifier
2: Initiate upload request

scs Server command specifier
2: Initiate upload response

n Number of bytes in **d** that do not contain data. Only valid if **e** = 1 and **s** = 1; otherwise it is 0. 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. Represents index/sub-index of data to be transferred by SDO.

d Data
e = 0, **s** = 0: **d** is reserved for future use.
e = 0, **s** = 1: **d** contains the number of bytes to be uploaded.
Byte 4 contains LSB and byte 7 contains MSB.
e = 1, **s** = 1: **d** contains data of length 4-n to be downloaded. The encoding depends on the type of data referenced by index and sub-index.
e = 1, **s** = 0: **d** contains an unspecified number of bytes to be uploaded.

x Not used; always 0.

reserved Reserved for future use; always 0.

4.4 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	
css = 3	**t**	**x**	**reserved**

Server to client:

0		1		8
7…5	4	3…1	0	
scs = 0	**t**	**n**	**c**	**seg-data**

where:

css Client command specifier
3: Upload segment request

scs Server command specifier
0: Upload segment response

t Toggle bit, which alternates for each subsequent segment to be uploaded. First segment has toggle-bit set to 0. Toggle bit is equal for request and response message.

c Whether or not there are still more segments to be uploaded:
0: More segments to be uploaded
1: No more segments to be uploaded

seg-data Maximum seven bytes of segment data uploaded. Encoding depends on type of data referenced by index and sub-index.

n 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 future use; always 0.

4.5 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 (data)**

where:

cs Command specifier
4: Abort transfer request

x Not used; always 0.

m Multiplexor. Represents index/sub-index of SDO.

d Four-byte abort code giving reason for abort, encoded as Unsigned32 value.

The SDO Abort codes are listed in the following table:

Abort Code	Description
0503 0000h	Toggle bit not alternated.
0504 0001h	Invalid or unknown client/server command specifier.
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 000h	Object does not exist in object dictionary.
0604 0041h	Object cannot be mapped to PDO.
0604 0042h	Number and length of objects to be mapped exceeds PDO length.
0604 0043h	General parameter incompatibility.
0604 0047h	General internal incompatibility in device.
0606 0000h	Access failed due to hardware error.
0607 0010h	Data type does not match, length of service parameter does not match.
0607 0012h	Data type does not match, service parameter too long.
0607 0013h	Data type does not match, service parameter too short.
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.
0800 0020h	Data cannot be transferred to or stored in application.

Abort Code	Description
0800 0021h	Data cannot be transferred to or stored in application due to local control.
0800 0022h	Data cannot be transferred to or stored in application due to present device state.
0800 0023h	Object dictionary dynamic generation failed or no object dictionary is present (for example, object dictionary is generated from file and generation has failed due to a file error).

Table 4-1: **SDO Abort Codes**

When the abort code is 0800 0000h, the actual error can be retrieved using the EC command.

4.6 Uploading Data Using an SDO

Data is uploaded in two basic formats:

- A short data item (up to four bytes) is uploaded by a single message conversation, called an expedited SDO.
- Longer data items require a longer conversation and are called segmented transfers.

Example of an expedited SDO:
An SDO is used to read the number of SDOs supported by the servo drive. The result (one transmit SDO and one receive SDO) is in object 0x1000 in the OD and is formatted as the 32-bit word 0x00020192. The client message body is outlined in the following table (% indicates a binary number):

Byte	Value	Description	Comment
0	%01000000	Header	Leading %010 is client command specifier (ccs) for Initiate Domain Upload.
1	0x00	Index (LO)	
2	0x10	Index (HI)	Use little endian.
3	0	Sub-index	No sub-index; therefore set to 0.
4 - 8	0	Reserved	

Table 4-2: **Expedited SDO - Client Message**

The server response is outlined in the following table:

Byte	Value	Description	Comment
0	%01000011	Header	Bits 7…5: %010 is client command specifier (**css**) for Initiate Domain Upload. Bits 3, 2: **n** bits that indicate that all data bytes are relevant. Bits 1, 0: %11, to expedite transfer and **d** contains information.
1	0x00	Index (LO)	
2	0x10	Index (HI)	Use little endian.
3	0	Sub-index	No sub-index, so it is set to 0.
4	0x92	Data: 0x00020192 in little endian format	
5	0x01		
6	0x02		
7	0		

Table 4-3: **Expedited SDO - Server Response**

4.7 Downloading Data Using an SDO

Data downloading with SDOs is very similar to data uploading. It can be handled in a single message conversation (expedited transfer) or in a segmented conversation. A detailed example of an expedited download transfer is given in section 5.3.6.

4.8 Error Correction

Included in an SDO transaction are features that enable error detection and correction. Errors can be detected with both software and hardware. Hardware error conditions relate to overrun, excessive line noise, broken lines or other malfunctions of the physical layer. The toggle bit is used for software detection; it correlates each domain segment with its previous and next segments. An unexpected toggle bit signals an error.

An Elmo slave that receives a corrupted message always terminates the SDO sequence with an "Abort domain transfer" message, structured as follows:

Byte	Description
0	0x80
1 – 2	Index
3	Sub-index
4	Additional code
5	Error code
6 – 7	Error class

Table 4-4: **Abort Domain Transfer Message Structure**

Fields 4 to 7 are described fully under "Abort SDO Transfer Protocol" in this manual.

Chapter 5: Process Data Objects (PDOs)

5.1 Receive PDOs

RPDOs are used to receive predefined and unconfirmed messages. An RPDO is received through use of an event, which may be asynchronous (such as "Message Received") or synchronous with the reception of a SYNC. Four receive PDOs are used with the Elmo drive. The following must be indicated:

- Objects that are mappable and have write access can be mapped to each RPDO .
- Execution of the mapped objects begins in the lower index of the relevant mapping object (0x1600 - 0x1603).
- High-priority objects, used for high-speed motion modes, can be mapped to an RPDO to avoid the overhead in using the standard interpreter. This mapping is dynamic, according to the setting of the PDO mapping at index 0x1600 - 0x1603 by the appropriate SDO. In a drive, only one fast reference object can be mapped to a single RPDO. The fast reference objects that can be mapped to an RPDO are:

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

- All type of objects can be mapped. Unsigned8, Signed8, Unsigned16, signed16, Unsigned32, and SignedT32 can be mapped. Manufacture specific object such as Binary interpreter and PVT\PT can also be mapped.
- RPDO1-4 can be freely mapped. The only constraints are that only one of the objects 0x2001 – 0x2004 may be mapped to an RPDO at a time, and also none of these objects may be mapped to more than one RPDO's simultaneously.
- Data of RPDOs mapped to objects 0x2001-0x2004 is processed immediately upon reception.
- Data of RPDOs is queued and it is passed for interpretation and execution at the next available background loop (Idle Loop) and according to the transmission type.
- The transmission type associated to the Rx mapping can be either synchronous where the data of the RPDO is passed for executing after the occurrence of the following SYNC (value =1), or asynchronous (value=255 or 254) in which the data of the RPDO is passed to the application immediately after reception and processed on the next Idle loop.
- At most one copy of each mapped RPDO can be stored for synchronous execution. If same RPDO arrives before next SYNC, it overwrites the previous with no notification.

- A change in RPDO mapping wipes any pending synchronous or asynchronous queued RPDO s of that type. The user must be aware and responsible.
- Change of transmission type from synchronous to asynchronous does not wipe pending instances.
- Changing transmission type to synchronous does not wipe any queued asynchronous instances.
- Objects can receive data from SDO and RPDO on the same time. User must be aware that the result of such case cannot be predicted. The final value of the object maybe either the SDO or the RPDO data.
- RPDOs cannot be retrieved by a remote transmitting (RTR).

Error Cases

When an RPDO fails to interpret, an emergency message 0x6300 0x01 is transmitted.

There might be several objects mapped into the same RPDO. The EMCY message identifies the objects that failed. A failure is consider when the received data could not be interpreted or executed. Refer to the "RPDO error handling" section in this manual.

In some cases Elmo's error code is produced. The emergency may contain this error code.

Example of RPDO mapping:
In order to set PDO1 to program PT-type motion commands (object 0x2002), set the object at index 0x1600 (PDO mapping, type defined by object 0x21).

Sub-index 0	Unsigned8	1
Sub-index 1	Unsigned32	0x20020020

The PDO mapping may be set using a single expedited SDO, which will be:

Byte	Value	Description
0	0x23	Initiate download, expedited, index valid, data valid.
1	0	Index (LO) to store at.
2	0x16	Index (HI) to store at.
3	1	Sub-index to store at.
4	0x20	Data length
5	0	Sub-index
6	0x02	High data
7	0x20	Low data

Table 5-1: **Service Data Objects**

The SDO is answered by the following:

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.
5	0	
6	0	
7	0	

Table 5-2: **Answered SDO**

5.2 Transmit PDOs

Four transmit PDOs can be used in Elmo drives. TPDOs are used to retrieve an object (data) from the drive. Objects that have read access and are mappable can be mapped to each one of the TPDOs. The transmitted data inside the TPDO is ordered according to the mapping order. The data starting from the LSB data is mapped first - in the lower index of the relevant mapping object.

5.3 PDO Mapping

PDO mapping is a convention that assigns (maps) an object from the object dictionary (data payload) to a PDO. Once mapped, the PDO can carry the assigned data items without explicit reference to the object dictionary, thereby saving on communication and CPU overhead.

Only a subset of the objects in the object dictionary can be mapped to a PDO, which can either receive (RPDO) or transmit (TPDO). The mapping of an RPDO enables reception of commands and variables — for example, efficient transmission of high-speed online motion commands to the drive — whereas the mapping of a TPDO enables the drive to send a predefined message in response to an event.

A TPDO is considered synchronous if triggered by a SYNC signal, and asynchronous if triggered by another event.

An RPDO is buffered upon reception; it is sent for interpretation immediately (when defined as asynchronous) or upon receipt of the next SYNC signal (when defined as synchronous).

5.3.1 The Mapping Trigger – Transmission Type

The transmission of a TPDO and RPDO is triggered by an event, which is defined by the PDO communication parameters: sub-index 2 of objects 0x1800 to 0x1803 (TPDO) and sub-index 2 of objects 0x1400 to 1403 (RPDO). These object dictionary entries are transmission types. The data type of the PDO parameter object is described in object 0x20.

PDO transmission types can be one of the following:

Transmission Type	Description
0	Synchronous transmission performed once, at next SYNC.
N=1…240	Synchronous transmission performed once per 0 < N ≤ 240 accepted SYNC signals.
254	Asynchronous transmission in response to a manufacturer-specific event.
255	Asynchronous transmission in response to a device profile (such as DSP 402).

The Elmo drive treats type 254 and 255 alike.

5.3.2 The Synchronous Trigger

Synchronous triggers are always related to the previous SYNC reception.

The RPDO transmission type is 1. The received message is buffered but actually transmitted for execution at the next SYNC message. Only one RPDO can be buffered for synchronous trigger. If another RPDO arrives before the SYNC, it overrides the previous RPDO without any notification. This method enables the simultaneous synchronization of executing commands in several drives. When a SYNC arrives, the buffered message is performed in the next available background cycle (Idle loop). Elmo fast reference objects (0x2001-0x2004) are executed immediately after reception regardless of transmission type.

Objects can receive data from SDOs and RPDOs simultaneously. Be aware that when this occurs, the results are unpredictable. The final value of the object may be either the SDO or the RPDO data.

With TPDOs, the message is transmitted according to transmission type value 1 to 240, where 1 indicates on each single SYNC, 2 means every second SYNC message, and so on.

5.3.3 The Asynchronous Trigger

Asynchronous triggers are defined in the device-specific protocol (such as DSP-402) or by Elmo manufacture-specific object 0x2F20.

When the device-specific protocol is used, the transmission type is 255 and the asynchronous behavior is defined in the object description.
A transmission type of 0 means that the message shall be transmitted after occurrence of The SYNC but acyclic (not periodically), only if an event occurred before the SYNC.

When the Elmo manufacture-specific object is used, each sub-index of object 0x2F20 defines trigger events for a single TPDO. The settings for the object go into effect only if the TPDO communication parameters are set to transmission type 254, and if the TPDO is correctly mapped. The Elmo drive treats transmission type 254 and transmission type 255 alike.

5.3.4 RPDO Error Handling

When an RPDO fails to be interpreted, an emergency message is transmitted.

Several objects may be mapped into the same RPDO. The EMCY message identifies the objects that failed. A failure occurs when the received data cannot be interpreted or executed.

In some cases, the Elmo error code is produced and may be included in the EMCY message.

The general emergency message structure is as follows:

0	Error code
1	
2	Error register
3	Elmo error code, for PVT/PT and RPDO messages only (refer to the *SimplIQ Software Manual*)
4	Error code data field 1
5	
6	Error code data field 2
7	

In case of an RPDO:

Error code field: 0x6300

Error register: 0x01

Elmo error code: 0 – RPDO failed without a relative error
> 0: RPDO failed with this error code

Error code data fields:
The relevant mapped object that failed according to the 32 bits mapping object.

Byte 4-5: Object size, in bits
Byte 6: Sub index
Byte 7: Index

The failed RPDO emergency can be masked by clearing bit 8 in object 0x2F21.

5.3.5 Mapping Parameter Objects

Objects 0x1A00 to 0x1A03 define the mapping for TPDOs. Objects 0x1600 – 0x1603 define the mapping for RPDOs. The type of PDO mapping parameter is described in object 0x21.

The method of mapping an object is defined in objects 0x1600 - 0x1603.

Example of TPDO mapping:
A drive will transmit the position value (object 0x6064, sub-index 0, INTEGER32), the PVT buffer head pointer (object 0x2F11, sub-index 0, INTEGER16) and PVT buffer tail pointer (object 0x2F12, sub-index 0, INTEGER16) per every accepted SYNC signal, according to the following procedure:

Step	Explanation
Set 0 to 0x1A00(0)	Stop all emissions of TPDO1.
Set 0x60640020 to 0x1A00(1)	Program the position value (object 0x6064) to first 4 bytes of the PDO.
Set 0x2F110010 to 0x1A00(2)	Program object 0x2F11 for next 2 bytes (PVT head pointer).
Set 0x2F120010 to 0x1A00(3)	Program object 0x2F12 for next 2 bytes (PVT tail pointer).
Set 1 to 0x1800(2)	Set transmission type in PDO communication parameters to "Transmit every SYNC." Other PDO communication parameters are not programmable.
Set 3 to 0x1A00(0)	Activate the three mapped objects.

In this example, xxxx(y) indicates sub-index y of object xxxx.

A PDO mapping element is a 32-bit bit field (object 0x21), divided as follows:

16-bit index	8-bit sub-index	Object length: 8 bits

5.3.6 Default Values

Default values of PDO mapping parameters are used at:

- Power up
- NMT communication reset (NMT 81h)
- NMT node reset (NMT 82h)

In these cases, the values of the PDO parameters receive the following:

Receive PDO 1 mapped to the DSP-402 *controlword* in the following manner:

Index	Sub-index	Name	Default Value
1400h	0h	Number of entries	2
	1h	COB-ID used by PDO	4000027Fh
	2h	Transmission type	255

Index	Sub-index	Name	Default Value
1600h	0	Number of mapped entries	1
	1	Controlword	6040 00 10h

Transmit PDO 1 monitors the drive behavior by transmitting the *statusword* whenever it changes (typically after reception of a *controlword)*:

Index	Sub-index	Name	Default Value
1800h	0h	Number of entries	3
	1h	COB-ID used by PDO	4000001FFh
	2h	Transmission type	255
	3h	Inhibit time	0

Index	Sub-index	Name	Default Value
1600h	1h	Number of entries	1
		Statusword	6041 00 10h

Index	Sub-index	Name	Default Value
2F20h	1h	TPDO1 asynchronous events	0


The asynchronous transmission of TPDO1 reflects changes performed 3 milliseconds prior to transmission.

Receive PDO 2 is mapped to the binary interpreter by default. This is done for compatibility reasons and to enable communication with the Elmo Composer.

Index	Sub-index	Name	Default Value
1401h	0h	Number of entries	2
	1h	COB-ID used by PDO	4000037Fh
	2h	Transmission type	254

Index	Sub-index	Name	Default Value
1601h	0	Number of entries	1
	1	Binary interpreter command	2013 00 40h

Transmit PDO 2 is mapped to the binary interpreter result object, transmitted each time the binary interpreter completes its processing. The event behavior is set by object 0x2F20, defined in the *CANopen Implementation Manual*.

Index	Sub-index	Name	Default Value
1801h	0h	Number of entries	3
	1h	COB-ID used by PDO	400002FFh
	2h	Transmission type	254
	3h	Inhibit time	0

Index	Sub-index	Name	Default Value
1A01h	1h	Number of entries	1
	2h	Binary interpreter result	2014 00 40h

Index	Sub-index	Name	Default Value
2F20h	2h	TPDO2 events	0x8000000

Chapter 6: Emergency (EMCY)

The Emergency object COB-ID is 0x81 to 0xFF. EMCY objects are fully defined in CiA DS 301. The structure of the manufacturer-specific emergency message is as follows:

0	Error code
1	
2	Error register
3	Elmo error code (refer to *SimplIQ Software Manual*)
4	Error code data field 1
5	
6	Error code data field 2
7	

Unused bytes must be set to zero.

6.1 Emergency Configuration

The events listed in this chapter can be configured to indicate which will emit an emergency message, using object 0x2F21.

6.2 Emergency Codes Related to Failure

SimplIQ digital servo drives issue an emergency code in response to an abnormal condition. All emergencies can be divided into two groups: those that can be masked and those that cannot. For a description of emergency codes that can be masked, refer to object 0x2F21. Emergency codes that cannot be masked are related to PVT/PT motion and are described in section 6.4 following.

6.3 Emergency Codes for Motor Faults

For a description of emergencies related to motor faults, refer to object 0x2F21 and to the MF command section in the *SimplIQ Command Reference Manual*.

6.4 Emergency Codes Related to PVT/PT Motion

During PT/PVT motion, the servo drive may issue emergency objects in order to indicate an error or to signal that it is in immediate need of additional data to prevent data queue underflow.

The following table lists the supported CAN emergencies. The Emergency error code for all messages in the table is 0xFF00, and the error register is 0x81.

Error Code (Hex)	Symbolic Name	Reason	Data Field
0x56	PVT_QUEUE_LOW	Number of valid PVT data rows has dropped below value stated in MP[5].	Field 1: Write pointer Field 2: Read pointer
0x5B	BAD_HEAD_POINTER	Write pointer out of physical range [1…64] of PVT table. MP[6] setting may be wrong.	Value of MP[6]
0x34	PVT_QUEUE_FULL	Attempt made to program more PVT points than supported by queue.	Field 1: Index of PVT table entry that could not be programmed
0x7	BAD_MODE_INIT_DATA	Cannot initialize motion due to bad setup data. Write pointer is outside range specified by start pointer and end pointer.	
0x8	MOTION_TERMINATED	Mode terminated and motor automatically stopped (in MO=1).	Field 1: Write pointer Field 2: 1: End of trajectory in non-cyclic mode 2: Zero or negative time specified for motional interval 3: Read pointer reached write pointer
0xA6	OUT_OF_MODOLU	The position is more than 2 Modolu.	

Table 6-1: **Emergency Codes (PVT/PT Motion)**

For more details, refer to object 0x2F21.

Chapter 7: Network Management (NMT)

Only the minimum, required, set of network management (NMT) services is supported by *SimplIQ*. NMT commands are used to control the communication state of the servo drive and to broadcast manufacturer messages to all other connected servo drives.

The following network communication states are supported:

State	Description
Unpowered/Initialization	Servo drive is not ready, or it is booting. Drive will not respond to communication and will not transmit anything.
Pre-operational	Servo drive boot sequence is complete, but no command has been received to enter operational mode. The servo drive will respond to SDO and NMT messages, but not to PDOs.
Operational	Servo drive is fully operational, responding to PDO, SDO and NMT messages.
Prepared	Servo drive has received a stop-node command and can respond only to NMT services.
Stopped	Servo drive can respond only to NMT objects (including heartbeats).

Table 7-1: **Network Management (NMT)**

When the servo drive is powered on, it enters the initialization state. After completing the boot sequence, it automatically enters the pre-operational state. The transition between pre-operational, operational and prepared states is carried out according to NMT messages. The COB-ID of an NMT command is always 0.

An NMT message is always two bytes long: the first byte is the command specifier and 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 the entire set of connected 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 (0x80)	Enter pre-operational state.
129 (0x81)	Reset node (perform full software reset).
130 (0x82)	Reset communication (reload communication parameters from flash, reset CAN software and enter pre-operational state).

Table 7-2: **Supported NMT Services**

Note: It is recommended to turn off the motor and kill any user program before executing NMT 130.

Chapter 8: SYNC and Time Stamp

The SYNC message has two uses:

- Synchronize the operation of synchronous PDOs. Only synchronous TPDOs can be used to transmit data from *SimplIQ* digital servo drives upon receiving a SYNC signal; synchronous RPDOs are not supported.
- 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 drive). It is cyclic and has 32 bits, meaning that it completes a full cycle in 4,295 seconds (approximately 72 minutes). When the motion clocks of all connected servo drives are synchronized to the motion clock of the master, multiple servo drives can perform complex synchronized motion with exact timing set by the network master.

The drives are synchronized by the transmission of a SYNC message, whose arrival time is captured by the drive. Upon reception of the SYNC, the drive latches its internal timer.

A Time Stamp is a 32-bit message that contains the master internal clock as generated upon receipt of the client's own SYNC. The Time Stamp causes a clock synchronization cycle to be executed. The drive uses the Time Stamp as an absolute timer and adjusts its internal time in relation to the time latched in the last SYNC[1]. To synchronize the master and drive clocks to full precision, the synchronization process is filtered in order to ensure that the timing jitter of the time stamping process does not adversely affect motion smoothness. It takes about 200 SYNC-Time Stamp pairs to ensure that all clocks are fully synchronized.

COB-ID 256 (0x100) is a constant dedicated ID used for this purpose. The master can send Time Stamps at any time.

A Time Stamp always refers to the previous SYNC message and must come no later than 5 seconds after the relevant SYNC.

[1] The highest value to adjust in a single time message is 250 milliseconds.

Chapter 9: Binary Interpreter Commands

With CAN, the interpreter commands are sent in binary form and are used for setting and retrieving all numerical data of the *SimplIQ* digital servo drives setup. The commands used by the binary interpreter for CAN communication are very similar to commands of the ASCII interpreter used for RS-232 communication.

The binary interpreter does not support string operations. Getting and setting strings may be performed by accessing the appropriate messages via SDOs, using the OS interpreter. 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	; or <CR> for commands and servo drive responses.	None.
Servo drive responses to Set commands	Always.	Drive does not respond to Set commands. An emergency object is sent if command execution fails.
Long response strings	Returned by certain commands, such as LS and BH.	No support for returned long strings, which are read via SDOs.

Table 9-1: **Comparison of ASCII vs. Binary Interpreter Commands**

TPDO2 is mapped by default to the transmit binary interpreter object (0x2012) and RPDO2 is mapped by default to the receive binary interpreter object (0x2013). TPDO2 is transmitted as an unsynchronized "Binary Interpreter complete" event.

The binary interpreter supports three types of commands:

- Set value
  These commands are eight bytes in length. The transmitted message includes either the reflection of the Set command or an error code, if a failure has occurred.
- Get value
  These commands can be four or eight bytes in length. An 8-byte response includes the reflection of the command and the resulting numerical value, and an error if a fault has occurred.
- Execute command
  This command can be four or eight bytes in length. An 8-byte response includes the reflection of the command and the resulting numerical value, and an error if a fault has occurred.

If an interpreter command cannot be serviced for any reason, bit 6 in byte 3 of TPDO2 is set on, and byte 4 of the response contains the Elmo error code (refer to the EC command section of the SimplIQ *Command Reference Manual*).

9.1 Binary Interpreter Commands and Results

The sequences in this section illustrate the binary interpreter options for setting, querying and executing commands.

9.1.1 Set and Query Commands

The host (client) sends commands (RPDO2) for setting variables in eight bytes (DLC=8). The drive (server) transmits the reply (TPDO2) as an asynchronous event of the received object.

9.1.1.1 RPDO2 Structure

RPDO2 is used to set values for the drive and query (get) values from it. The structure of the command is as follows:

- Bytes 0 to 3 are the header, which includes the command, command index (when needed) and data type (float or integer).
- Bytes 4 to 7 are the data, which is always four bytes. The format can be integer or float. The bytes are interpreted in little endian format (see Appendix).

The following table describes the format in which the host sends commands to the drive:

Byte	**0**	**1**	**2**	**3**			**4 - 7**
Bits	0…7	0…7	0…7	0…5	6	7	
Description	First command character	Second command character	Index for array parameter. 0 for scalar command		See note	0: Integer 1: float	Data in little endian format

Bytes 0 and 1, which represent the command character in ASCII, must be uppercase.

Byte 3, bit 6:
When this bit is set to 1, the drive treats the command as a "query" and not as a "setting." In this case, the rest of the data bytes are discarded and the drive replies to the command according to 4 bytes DLC. For compatibility reasons, bytes 4 to 7 should be 0.

Notes:

- In array commands in which the index is used (as in ET[100]), the lowest significant bits are in byte 2 (bits 0 to 7) and the most significant bits are in byte 3.

- Always use the bit 7 of byte 3 to indicate the data type (float or integer) in the transmitted message, even if the numerical data type is known in advance and given in the reference manual. This enables Elmo to guarantee that the type of numerical data returned for any interpreter command will remain unchanged in future versions.

Example 1:
CL[1] is set to 1.0, which is 3F800000h in hex IEEE format.
`CL[1]=1.0`

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	43	4C	0	80	F0	49	02	0

 Bit 7 in byte 3 is set to 1 to indicate that the value is float.

Example 2:
AC is set to 150,000 (0249F0h):
`AC=150000`

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	41	43	0	0	F0	49	02	0

Example 3:
AC is queried and the DLC is 4:
`AC`

Byte	**0**	**1**	**2**	**3**
Hex value	41	43	0	0

Example 4:
This is the same query as in Example 3, but with a **DLC of 8** ("Byte 3, bit 6" note).
`AC`

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	41	43	0	40	0	0	0	0

In both Example 3 and Example 4, the reply from the server is:

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	41	43	0	0	F0	49	02	0

Example 5:
`CA[18] = 4096` (1000h) (18 in decimal - 12h in hex)

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	43	41	12	0	0	0	10	0

Example 6:
In this example, the server replies to the command ET[992] (3E0h), assuming that the value is 32121 (7D79h). This is done to query the DLC 8 format (bit 6 in byte 3 is set):
`ET[992]` (3E0h)

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	45	54	E0	43	0	0	0	0

The server replies as follows:

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	45	54	E0	03	79	7D	0	0

9.1.1.1 TPDO2 Structure

The server (drive) replies (TPDO2) to query and set requests in eight bytes (DLC=8):

- Bytes 0 to 3 are the header, which includes the responding command, command index (when needed) and data type (float or integer). It also indicates whether the response data is true data or an error code.
- Bytes 4 to 7 are data, which is either a reflection of the host Set command or an error code according to the EC command.

Byte	**0**	**1**	**2**	**3**			**4 - 7**
Bits	0…7	0…7	0…7	0…5	6	7	
Description	First character	Second character	Index for array parameter. 0 for scalar command		See note.	0: Integer 1: Float	Valid data or error code. Little endian format.

The server replies as follows:

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	45	54	E0	03	79	7D	0	0


Bytes 0 and 1 represent the command character and must be uppercase.

Byte 3, bit 6:
When this byte is 1 for TPDO, the data in bytes 4 to 7 should be interpreted as an error code. Refer to the EC command section in the SimplIQ *Command Reference Manual* for details.


In array commands in which the index is used (as in ET[100]), the lowest significant bits are in byte 2 (bits 0 to 7) and the most significant bits are in byte 3.

Example:
The server replies to the command `CA[1]=4`, which is out of range: error code 21 (15h).

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	43	42	02	40	21	0	0	0

9.1.2 Execute Command

These commands are used to instruct the drive to perform a sequence. The reply to these commands is only an acknowledgement or an error code; there is no value for executing command. Execute commands are a unique case of RPDO2, which can be used with a DLC of either 4 or 8.

Example:
BG command, to start a motion.

DLC4:

Byte	**0**	**1**	**2**	**3**
Hex value	42	47	0	0

DL8:

Byte	**0**	**1**	**2**	**7**
Hex value	42	47	0	0

The reply is always eight bytes long and indicates either success or failure (error).

Success

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	42	47	0	0	0	0	0	0

Failure: error code 58 (3Ah) for "Motor must be on"

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex value	42	47	0	40	3A	0	0	0

9.2 ASCII Interpreter Commands not Supported by Binary Interpreter

Commands that deal with strings are not accessible using the binary interpreter. In most cases, these strings may be accessed using the OS interpreter prompt.

Command	**Description**	**Alternative**
VR	Detailed software version string.	Use OS prompt instead of SDO to read object 0x100a.
CD	CPU dump in case of fatal exception.	Use OS prompt.
LS/DL	List/download from serial flash.	Use OS prompt.
DF	Download firmware version.	Use SDO to write object 0x2090.
BH	Bring recorded value.	Use SDO to read object 0x2030.
XC##\XQ##	Execute user program.	Use OS prompt.

Figure 9-1: **Some of Commands not Available to Binary Interpreter**

The binary interpreter cannot handle expressions, which must be dealt with using the OS interpreter.

Chapter 10: The OS Interpreter

The OS interpreter is used to process any *SimplIQ* interpreter string command, and to return the string results. The only limitation in its use is that the returned strings cannot exceed 500 characters in length, a limit that must be considered when uploading recorded data. A more efficient — and unlimited — method to upload recorder data is to use object 0x2030.

To issue an OS interpreter command

1. Set OS mode to evaluate the string immediately, by writing 0 to object 0x1024.
2. Write the command string to object 0x1023, sub-index 1.

The command execution can be resolved by an event-driven PDO (object 0x2F20) or by polling object 0x1023, sub-index 2. The polling may return:

- 0xFF: Command still executing (can be aborted by writing 3 to object 0x1024).
- 0x1: Command successfully executed. Result is waiting for read.
- 0x3: Command rejected. Error code waiting for read.

When the response is ready, it can be read from object 0x1023, sub-index 3.

Example:
The following describes the use of the OS interpreter to send the command PX=1234.

Client initiates OS Evaluate Immediately mode:

RSDO	Object 0x1024						
23	24	10	00	00	00	00	00

Server replies:

TSDO	Object 0x1024						
60	24	10	00	00	00	00	00

Client initiates segmented SDO download:

RSDO	Object 0x1023		Sub-index				
21	23	10	01	00	00	00	00

Server replies:

TSDO	Object 0x1023		Sub-index				
60	23	10	01	00	00	00	00

Client sends PX=1234 in one SDO:

RSDO	P	X	=	1	2	3	4
01	50	58	3D	31	32	33	34

Server acknowledges that the RSDO was received OK:

TSDO							
20	00	00	00	00	00	00	00

Client gets PX value from OS interpreter (assuming OS was already defined as "Evaluate Immediately":

RSDO	Object 0x1023		Sub 1	P	X		
23	23	10	01	50	58	00	00

Server acknowledges that the RSD0 was received OK:

TSDO	Object 0x1023						
60	23	10	01	00	00	00	00

Client queries status of command:

RSDO	Object 0x1023		Sub-index				
40	23	10	02	00	00	00	00

Server replies that command was executed and that the result is waiting:

TSDO	Object 0x1023		Sub-index	Execute OK			
42	23	10	02	01	00	00	00

Client queries for reply value:

RSDO	Object 0x1023		Sub-index				
42	23	10	03	00	00	00	00

Server replies with value of valid PX:

TSDO	Object 0x1023		Sub-index	1	2	3	4
43	23	10	03	31	32	33	34

Chapter 11: The EDS

The Electronic Data Sheet (EDS) assists CANopen configuration personnel in determining which objects a CAN slave supports. The EDS has a standard format that is explained in CiA DS 301, version 4. This document defines an optional read-only object used to upload the EDS directly from the CAN slave.

- Object 0x1021 is the EDS, stored as an ASCII string – *future implementation*
- Object 0x1022 defines the EDS compression style, which must be 0, for No compression - *future implementation*

The EDS is loaded to the internal serial flash memory of the *SimplIQ* digital servo drive as part of the firmware download process – *future implementation*

Chapter 12: Communication Profile

1000h: *Device type*
1001h: *Error register*
1002h: *Manufacturer status register*
1003h: *Predefined error field*
1005h: *COB-ID SYNC message*
1008h: *Manufacturer device name*
1009h: *Manufacturer hardware version*
100Ah: *Manufacturer software version*
100Bh: *Node ID*
1012h: *COB-ID time stamp*
1013h: *High-resolution time stamp*
1014h: *COB-ID emergency object*
1017h: *Producer heartbeat time*
1018h: *Identity object*
1021h: *Store EDS (not implemented)*
1022h: *EDS storage format (not implemented)*
1023h: *OS command and prompt*
1024h: *OS command mode*
1029h: *Error behavior*
1200h: *SDO server parameter*
1400h - 1403h: *Receive PDO communication parameter*
1600h - 1603h: *Receive PDO mapping*
1800h - 1803h: *Transmit PDO communication parameter*
1A00h - 1A03: *Transmit PDO mapping*

Object 0x1000: *Device type*
This object contains information about the device type and functionality. It is comprised of a 16-bit field that describes the device profile used, and a second 16-bit field that gives additional information about optional functionality of the device.

MSB LSB

Byte 4	**Byte 3**	**Byte 2**	**Byte 1**
Additional information		Device profile number	

- Object description:

Index	**1000h**
Name	Device type
Object code	VAR
Data type	UNSIGNED32
Category	Mandatory

- Entry description:

Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	0x191

Object 0x1001: ***Error register***
This object is an error register for the device.

- Object description:

Index	**1001h**
Name	Error register
Object code	VAR
Data type	UNSIGNED8
Category	Mandatory

- Entry description:

Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	0

- Data description (M for Mandatory and O for Optional):

Bit	**M/O**	**Meaning**
0	M	Generic error
1	O	Current
2	O	Voltage
3	O	Temperature
4	O	Communication error (overrun, error state)
5	O	Device profile specific
6	O	Reserved (always 0)
7	O	Manufacturer specific

If a bit is set to 1, the specified error has occurred. The only mandatory error that must be signaled is the generic error, which is signaled in any error situation.

Object 0x1002: ***Manufacturer status register***
This object is a common status register for manufacturer-specific purposes. It returns the status similar to the SR command.

- Object description:

Index	**1002h**
Name	Manufacturer status register
Object code	VAR
Data type	UNSIGNED32
Category	Optional

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	UNSIGNED32
Default value	

Object 0x1003: ***Pre-defined error field***
This object holds the errors that have occurred in the device and have been signaled via the Emergency object. In doing so, it provides an error history. The entry at sub-index 0 contains the number of actual errors recorded in the array, starting at sub-index 1. It can read 0 if no error is registered, or 1 if an error is registered. The present version of *SimplIQ* digital servo drives store one error at the most.

Writing a 0 to sub-index 0 empties the array. Values higher than 0 are not allowed, in order to prevent an Abort message (error code: 0609 0030h).

The error numbers are of type Unsigned32 and are composed of a 16-bit error code, an 8-bit error register and an 8-bit additional error information field, which is manufacturer-specific. The error code is contained in the lower two bytes (LSB) and the additional information is included in the upper two bytes (MSB). The pre-defined error field has the following structure:

MSB LSB

Bits 24…31	**Bits 16…23**	**Bits 0…15**
Manufacturer-specific error code	Error register	Error code

For error code and error register values, see object 0x2F21.

- Object description:

Index	**1003h**
Name	Pre-defined error history
Object code	ARRAY
Data type	UNSIGNED32
Category	Mandatory

- Entry description:

Sub-index	**0**
Description	Number of actual errors
Entry category	Mandatory
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED8
Default value	0

Sub-index	**1**
Description	Standard error field
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	0

Object 0x1005: ***COB-ID SYNC message***

This object defines the COB-ID of the synchronization object (SYNC). It also defines whether or not the device generates the SYNC. The structure of the object is as follows:

MSB LSB

Bits	**31**	**30**	**29**	**28 - 11**	**10 - 0**
11-bit ID	X	0/1	0	00000000000000000	11-bit identifier

- Description of SYNC COB-ID entry:

Bit Number	**Value**	**Meaning**
31 (MSB)	X	Do not care
30	0 1	Device does not generate SYNC message Device generates SYNC message
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)
28 - 11	0 X	If bit 29 = 0 If bit 29 = 1: bits 28 - 11 of 29-bit SYNC COB-ID
10 - 0 (LSB)	X	Bits 10 - 0 of SYNC COB-ID

Bits 29 and 30 are static (unchangeable). Any attempt to modify this object will result in an Abort message. Object description:

Index	**1005h**
Name	COB-ID SYNC
Object code	VAR
Data type	UNSIGNED32
Category	Mandatory

- Entry description:

Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	0x80

According to CiA DSP 301, the 0x1005 object has read/write access, although it only has read access with *SimplIQ* digital servo drives .

Object 0x1008: ***Manufacturer device name***
This object contains the manufacturer device name, such as "Harmonica".

- Object description:

Index	**1008h**
Name	Manufacturer device name
Object code	VAR
Data type	VISIBLE STRING
Category	Optional

- Entry description:

Access	Read only
PDO mapping	No
Value range	
Default value	

Object 0x1009: ***Manufacturer hardware version***
This object contains the version number of the manufacturer's hardware. The WS[30] command contains the hardware version as a 32-bit unsigned integer, while this object conveys the information as a hexadecimal number. For example, if WS[30] is equal to 0x1FF6B, the string returned by this object will be 0x1FF6B.

- Object description:

Index	**1009h**
Name	Manufacturer hardware version
Object code	VAR
Data type	VISIBLE STRING
Category	Optional

- Entry description:

Access	Read only
PDO mapping	No
Value range	
Default value	No

Object 0x100A: ***Manufacturer software version***
This object contains the version identification of the manufacturer's software.

- Object description:

Index	**100Ah**
Name	Manufacturer software version
Object code	VAR
Data type	VISIBLE STRING
Category	Optional

- Entry description:

Access	Read only
PDO mapping	No
Value range	No
Default value	No

Object 0x100B: ***Node ID***
This object contains the node ID of the drive. If the node ID is changed, the object will return the updated value only after Reset Communication and Start Communication NMT messages have been sent.

- Object description:

Index	**100Bh**
Name	Node ID
Object code	VAR
Data type	UNSIGNED8
Category	Optional

- Entry description:

Access	Read only
PDO mapping	No
Value range	1…127
Default value	No

Object 0x1010: ***Save parameters***
This object is used to save parameters in non-volatile memory. Through read access, the drive provides information about its save capabilities, using:

- Sub-index 0: Largest supported sub-index
- Sub-index 1: Save all parameters

In order to avoid accidental storage, storage is only executed when a specific signature — "save" — is written to the appropriate sub-index.

MSB			LSB
"e"	"v"	"a"	"s"
65H	76H	61H	73H

- Object description:

Index	**1010h**
Name	Store parameters
Object code	RECORD
Data type	UNSIGNED32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Largest supported sub-index
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	1

Sub-index	**1**
Description	Save all parameters
Entry category	Mandatory
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED32
Default value	No

Object 0x1011: ***Restore parameters***
This object is used to restore parameters from non-volatile memory. Through read access, the drive provides information about its restore capabilities, using:

- Sub-index 0: Largest supported sub-index
- Sub-index 1: Restore all parameters

In order to avoid accidental storage, restore is only executed when a specific signature — "load" — is written to the appropriate sub-index.

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

- Object description:

Index	**1011h**
Name	Restore parameters
Object code	RECORD
Data type	UNSIGNED32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Largest supported sub-index
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	1

Object 0x1012: ***COB-ID time stamp***

This object defines the COB-ID of the Time Stamp object (TIME). It also indicates whether the devices consumes or generates the TIME. The structure of the object is as follows:

MSB LSB

Bits	**31**	**30**	**29**	**28 - 11**	**10 - 0**
11-bit ID	0/1	0/1	0	000000000000000000	11-bit identifier
29-bit ID	0/1	0/1	1	29-bit identifier	

- Description of COB-ID time stamp entry:

Bit Number	**Value**	**Meaning**
31 (MSB)	0 1	Devices does not consume TIME message Device consumes TIME message
30	0 1	Device does not produce TIME message Device produces TIME message
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)
28 - 11	0 X	If bit 29 = 0 If bit 29 = 1: bits 28 - 11 of 29-bit TIME COB-ID
10 - 0 (LSB)	X	Bits 10 - 0 of TIME COB-ID

Bits 29 and 30 are static (unchangeable). The value of this object cannot be modified. Writing the default COB-ID of the time stamp will not cause an abort code.

Object description:

Index	**1012h**
Name	COB-ID time stamp message
Object code	VAR
Data type	UNSIGNED32
Category	Optional

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	UNSIGNED32
Default value	100h

According to CiA DSP 301, the 0x1012 object has read/write access, although it only has read access with *SimplIQ* digital servo drives.

Object 0x1013: ***High-resolution time stamp***
This object contains a time stamp with a 1-microsecond resolution.

- Object description:

Index	**1013h**
Name	High resolution time stamp
Object code	VAR
Data type	UNSIGNED32
Category	Optional

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	0…0xFFFFFFFF
Default value	0

Object 0x1014: ***COB-ID emergency object***
This object defines the COB-ID of the Emergency object (EMCY). The structure of the object is as follows:

MSB LSB

Bits	**31**	**30**	**29**	**28 - 11**	**10 - 0**
11-bit ID	0/1	0	0	000000000000000000	11-bit identifier
29-bit ID	0/1	0	1	29-bit identifier	

- Description of EMCY COB-ID entry:

Bit Number	**Value**	**Meaning**
31 (MSB)	0 1	EMCY exists / is valid EMCY does not exist / is invalid
30	0	Reserved (always 0)
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)
28 - 11	0 X	If bit 29 = 0 If bit 29 = 1: bits 28 - 11 of 29-bit EMCY COB-ID
10 - 0 (LSB)	X	Bits 10 - 0 of EMCY COB-ID

- Object description:

Index	**1014h**
Name	COB-ID emergency message
Object code	VAR
Data type	UNSIGNED32
Category	Mandatory

- Entry description:

Access	Read only
PDO mapping	No
Value range	0x81…0xFFFFFFFF
Default value	0x81

Object 0x1016: ***Consumer heartbeat time***

The consumer heartbeat time defines the expected heartbeat cycle time and thus has to be higher than the corresponding producer heartbeat time configured on the device producing this heartbeat. Monitoring starts after the reception of the first heartbeat. If the consumer heartbeat time is 0 the corresponding entry is not used. The time has to be a multiple of 1ms.

UNSIGNED32

MSB			LSB
Bits	31…24	23…16	15…0
Value	0	Node-ID	Heartbeat time, in milliseconds
Encoding	-	Unsigned8	Unsigned16

At an attempt to configure several consumer heartbeat times unequal 0 for the same Node-ID the device aborts the SDO download with abort code 0604 0043h.

- Object description:

Index	**1016h**
Name	Consumer heartbeat time
Object code	ARRAY
Data type	UNSIGNED32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Largest supported sub-index
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	1
Default value	1

Sub-index	**1**
Description	Consumer heartbeat time
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED32
Default value	No

Object 0x1017: ***Producer heartbeat time***
This object defines the cycle time of the heartbeat, which must be a multiple of 1 millisecond. It is 0 if not used.

- Object description:

Index	**1017h**
Name	Producer heartbeat time
Object code	VAR
Data type	UNSIGNED16
Category	Mandatory

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	UNSIGNED16
Default value	0

Object 0x1018: ***Identity object***
This object stores the LSS address used for the CAN ID and baud rate setting.

- Object description:

Index	**1018h**
Name	Identity object
Object code	RECORD
Data type	UNSIGNED32
Category	Mandatory

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	No

Sub-index	**1**
Description	Vendor ID
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	No

Sub-index	**2**
Description	Product code
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	No

Sub-index	**3**
Description	Revision number
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	No

Sub-index	**4**
Description	Serial number
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	No

Note:

For the sub index 3 - Revision number, Elmo implement the follows protocol:

Major Field (16 bits)	Minor Field (16 bits)
1 (0x0001)	1000 (0x03E8)

Example for revision number 0x103E8

Major field, when modified it indicates a big modification with compatibility issues.

Minor field, when modified compatibility remains. The thousands entry means that a big change was performed (such as a new protocol). The hundreds, tens & single number indicate minor changes such as engineering version or internal modification.

Object 0x1023: ***OS command and prompt***
This object is used with the OS interpreter (see **Chapter 10**).

- Object description:

Index	**1023h**
Name	OS command
Object code	RECORD
Data type	Command Par
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	No

Sub-index	**1**
Description	Command
Entry category	Optional
Access	Write only
PDO mapping	No
Value range	String octet
Default value	None

Sub-index	**2**
Description	Status: 0: Last command completed, no errors, no reply 1: Last command completed, reply ready 3: Last command rejected, reply ready 255: Executing
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	0

Sub-index	**3**
Description	Reply
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	String octet
Default value	None

Object 0x1024: ***OS command mode***
This object is used with the OS interpreter (see **Chapter 10**).

- Object description:

Index	**1024h**
Name	OS command mode
Object code	VAR
Data type	UNSIGNED8
Category	Optional

- Entry description:

Access	Write only
PDO mapping	No
Value range	UNSIGNED8: 0: Execute next command immediately 1 - 2: Not supported 3: Abort execution
Default value	No

Object 0x1029: ***Error behavior***
This object reports the CAN communication state after a heartbeat failure. The value of the object asserts that after such a failure, the CAN communication state is:
0: Pre-operational (only if current state is operational)
1: No state change
2: Stopped

The default value is 1 (no state change).

- Object description:

Index	**1029h**
Name	Error behavior
Object code	ARRAY
Data type	UNSIGNED8
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of error classes
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	1 to 0xFE
Default value	1

Sub-index	**1**
Description	Communication error
Entry category	Mandatory
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED8
Default value	1

Object 0x1200: ***SDO server parameter***

This object is used to describe the SDO used on a device. The data type has the index 22h in the object dictionary. The number of supported entries in the SDO object record is specified by sub-index 0h. The values at 1h and 2h specify the COB-ID for this SDO. Sub-index 3 gives the server of the SDO if the record describes an SDO for which the device is a client, and it gives the client of the SDO if the record describes an SDO for which the device is the server. The structure of the SDO COB-ID entry is as follows:

MSB LSB

Bits	**31**	**30**	**29**	**28 - 11**	**10 - 0**
11-bit ID	0/1	0	0	000000000000000000	11-bit identifier
29-bit ID	0/1	0	1	29-bit identifier	

- Description of SDO COB-ID entry:

Bit Number	**Value**	**Meaning**
31 (MSB)	0 1	SDO exists / is valid SDO does not exist / is invalid
30	0	Reserved (always 0)
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)
28 - 11	0 X	If bit 29 = 0 If bit 29 = 1: bits 28 - 11 of 29-bit SDO COB-ID
10 - 0 (LSB)	X	Bits 10 - 0 of SDO COB-ID

An SDO is valid only if both SDO valid bits are 0. These objects contain the parameters for which the SDO is the server. This entry is read only.[2] COB-IDs cannot be changed.

- Object description:

Index	**1200h**
Name	Servo SDO 1 Parameter
Object code	RECORD
Data type	SDO Parameter (object 0x22)
Category	Optional

2 Ensure that the COB-IDs of the default SDO cannot be manipulated by writing to the OD.

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	2
Default value	2

Sub-index	**1**
Description	COB-ID client → server (Rx)
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	0x601…0x67F
Default value	0x601

Sub-index	**2**
Description	COB-ID client → server (Tx)
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	0x581…0x5FF
Default value	0x581

Objects 0x1400 - 0x1403: ***Receive PDO communication parameter***

- Object description:

Index	**1400h - 1403h**
Name	Receive PDO Parameter
Object code	RECORD
Data type	PDO CommPar (object 0x20)
Category	Conditional: mandatory for each supported PDO

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	2

Sub-index	**1**
Description	COB-ID used by PDO
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED32
Default value	Index 1400h: 0x27F Index 1401h: 0x37F Index 1402h: 0x47F Index 1403h: 0x57F

Sub-index	**2**
Description	Transmission type
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	255

Notes:

- Transmission type may be 255, 254 or 1. On an attempt to change the value of the transmission type to a value that is not supported by the Elmo drive an abort message (abort code: 0609 0030h) is generated.

Objects 0x1600 - 0x1603: ***Receive PDO mapping***
These objects contain the mapping for the PDOs that the Elmo drive is able to receive. The sub-index 0h contains the number of valid entries within the mapping record. This number of entries is also the number of the application variables that are received with the corresponding PDO.

- Mapping method: (for more information please refer to the "PDO mapping" section in this manual.
  The sub-indices from 1h to number of entries contain the information about the mapped application variables. These entries describe the PDO contents by their index, sub-index and length. All three values are hexadecimal coded.

  The PDO mapping element is a 32-bit field (object 0x21), divided as follows:
  MSB LSB

Index: 16-bit	Sub index: 8-bit	Object length: 8 bits

  The length of the entry contains the length of the object in bits (1..40h).

  If the change in PDO mapping cannot be executed (for example, the PDO length is exceeded or the SDO client attempts to map an object that cannot be mapped), the device responds with an Abort SDO Transfer Service.

  Sub-index 0 determines the valid number of objects that have been mapped. For changing the PDO mapping, sub-index 0 must be set to 0 (mapping is deactivated). On then can the objects be remapped.

  When a new object is mapped by writing a sub-index between 1 and 8, if the object does not exist or it cannot be mapped, the SDO transfer is aborted with the Abort SDO Transfer Service with abort code 0602 0000h or 0604 0041h. You can configure a single object without remapping the remaining objects. The previously mapped objects will not be cleared.

  After all objects are mapped, sub-index 0 is set to the valid number of mapped objects. Then the drive rechecks the mapping integrity. Finally, the PDO is created by writing to its communication parameter COB-ID. When sub-index 0 is set to a value greater than 0, the device may validate the new PDO mapping before transmitting the response of the SDO service. If an error is detected, the Elmo drive transmits the Abort SDO Transfer Service with abort codes 0602 0000h, 0604 0041h or 0604 0042h.

  When sub-index 0 is read, the actual number of valid mapped objects is returned.

- Object description:

Index	**1600h - 1603h**
Name	Receive PDO Mapping
Object code	RECORD
Data type	PDO Mapping
Category	Conditional: mandatory for each supported PDO

- Entry description:

Sub-index	**0**
Description	Number of mapped application object in PDO
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED32
Default value	1

Sub-index	**1**
Description	PDO mapping for first and last application object to be mapped
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED32
Default value	Object 1600h(1): 0x20010040 (controlword) Object 1601h(1): 0x20120040 (binary interpreter) Object 1602h: 0 Object 1603h: 0
Default value	0x581

Notes:

- Up to 8 objects can be mapped to a single RPDO.
- Dummy entries are not supported by the Elmo drive.
- The object can be mapped during the PRE OPERATIONAL stage.
- Dynamic mapping is allowed during the OPERATIONAL stage. The SDO client is responsible for data consistency.

Objects 0x1800 - 0x1803: ***Transmit PDO communication parameter***

- Object description:

Index	**1800h - 1803h**
Name	Transmit PDO parameter
Object code	RECORD
Data type	PDO CommPar (object 0x20)
Category	Conditional: mandatory for each supported PDO

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	No

Sub-index	**1**
Description	COB-ID used by PDO
Entry category	Optional
Access	Object 1800h: Read/Write Object 1801h: Read/Write Object 1802h: Read/Write Object 1803h: Read/Write
PDO mapping	No
Value range	UNSIGNED32
Default value	Index 1800h: 0x400001FF Index 1801h: 0x400002FF Index 1802h: 0x400003FF Index 1803h: 0x400004FF

Sub-index	**2**
Description	Transmission type
Entry category	Optional
Access	Object 1800h: Read/Write Object 1801h: Read/Write Object 1802h: Read/Write Object 1803h: Read/Write
PDO mapping	No
Value range	0…240
Default value	0

Sub-index	**3**
Description	Inhibit time
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	UNSIGNED16
Default value	0 (no inhibit time between messages)

Sub-index	**4**
Description	Reserved
Entry category	
Access	
PDO mapping	
Value range	
Default value	

- **COB ID used by PDO**
  Only the default COB and specific Node ID can be written to the drive. An attempt to write another COB ID will result in an abort (Abort Code 0609 0030h).

- **Event time:**
  When a TPDO transmission type is 254 or 255, an event time can be used. The event occurs when the time is elapsed. The event time elapse is a multiple of 1 millisecond of sub-index 5. It causes the transmission of this PDO in addition to other asynchronous events. The occurrence of an event sets the timer again. A value of 0 disables this function. The transmission of the TPDO has an accuracy with 2 milliseconds.

- **Inhibit time:**
  Inhibit time specifications do not relate to the generating event but to the transmission of the TPDO. The inhibit time resolution is 100 microseconds. The exact inhibit times are not very accurate and can actually be up to 2 milliseconds (20 units of inhibit time) longer than defined by sub-index 3 of this object. For example, if an inhibit time is specified as 10 milliseconds, its actual inhibit time length may vary in the range of [10…12] milliseconds. Inhibit time restrictions are explained in section 2.4.

Objects 0x1A00 - 0x1A03: ***Transmit PDO mapping***
These objects contain the mapping for the PDOs that the Elmo drive is able to transmit. Sub-index 0h contains the number of valid entries within the mapping record. This number of entries is also the number of the application variables that are transmitted with the corresponding PDO.

For more information about the PDO mapping method, refer to objects 0x1600 - 0x1603.

- Object description:

Index	**1A00h - 1A03h**
Name	Transmit PDO mapping
Object code	RECORD
Data type	PDO Mapping (object 0x21)
Category	Conditional: mandatory for each supported PDO

- Entry description:

Sub-index	**0**
Description	Number of mapped application objects in PDO
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	0

Sub-index	**1 - 8**
Description	PDO mapping for nth application object to be mapped
Entry category	Optional
Access	Object 1A00h: Read/Write Object 1A01h: Read/Write Object 1A02h: Read/Write Object 1A03h: Read/Write
PDO mapping	No
Value range	UNSIGNED32

Notes:

- Up to eight objects can be mapped to a single TPDO.
- Dummy entries are not supported by the Elmo drive.
- The object can be mapped during PRE OPERATIONAL stage.

Dynamic mapping is allowed during the OPERATIONAL stage. The SDO client is responsible for data consistency.

Chapter 13: Manufacturer-specific Objects

2001h: *PVT data*
2002h: *PT data*
2004h: *ECAM data*
2012h: *Binary interpreter input*
2013h: *Binary interpreter output*
2030h: *Recorder data*
2040h: *Coordinate system group ID*
2041h: *Amplifier-free running timer*
2082h: *CAN controller status*
208Ah: *Begin time*
2090h: *Firmware download*
20A0h: *Auxiliary position actual value*
20A1h: *Main position error*
2200h: *Digital input*
2201h: *Digital input low byte*
2F00h: *User Integer*
2F01h: *User Float Array*
2F01h: *ET Array*
2F11h: *PVT head pointer*
2F12h: *PVT tail pointer*
2F15h: *Profile position remaining points*
2F20h: *PDO events*
2F21h: *Emergency events*
2F22h: *Bus off timeout*
2F23h: *Digital input TPDO event parameters*
2F30h: *Last time stamp correction*
2F31h: *Last SYNC time*
2F40h: *Configuration object*

Object 0x2001: ***PVT data***
This object sets PVT data for the PVT motion mode.

- Object description:

Index	**2001h**
Name	PVT data
Object code	VAR
Data type	PVT DataPar (object 0x40)
Category	Optional

- Entry description:

Access	Write only
PDO mapping	Yes
Value range	No
Default value	No

Notes:

- The transmission type for mapping this object must be 255; otherwise, an Abort message (code 0604 0043h) will be transmitted.
- When this object is used to feed PVT reference points to the drive while a PVT motion is executing, the drive automatically enters Online Feed mode, in which the drive is aware of the position written and takes the following precautions:
  - If the feed is too slow, the drive needs to fetch points not yet programmed, and the motion will abort with a "Queue underflow" emergency.
  - If the feed is too fast, so that points not yet used are overstruck, the drive will reject the feed attempt with a "Queue overflow" emergency.
- The Online Feed mode will continue until at least one setting of PVT parameters has been completed without using the RPDO.

Object 0x2002: *PT data*
This object sets PT data for the PT motion mode.

- Object description:

Index	**2002h**
Name	PT data
Object code	VAR
Data type	PT DataPar (object 0x41)
Category	Optional

- Entry description:

Access	Write only
PDO mapping	Yes
Value range	No
Default value	No

Notes:

- The transmission type for mapping this object must be 255; otherwise, an Abort message (code 0604 0043h) will be transmitted.
- When this object is used to feed PT reference points to the drive while a PT motion is executing, the drive automatically enters Online Feed mode, in which the drive is aware of the position written and takes the following precautions:
  - If the feed is too slow, the drive needs to fetch points not yet programmed, and the motion will abort with a "Queue underflow" emergency.
  - If the feed is too fast, so that points not yet used are overstruck, the drive will reject the feed attempt with a "Queue overflow" emergency.
- The Online Feed mode will continue until at least one setting of PT parameters has been completed without using the RPDO.

Object 0x2004: ***ECAM data***
This object sets ECAM data for the ECAM table entries.

- Object description:

Index	**2004h**
Name	ECAM data
Object code	VAR
Data type	PT DataPar (object 0x41)
Category	Optional

- Entry description:

Access	Write only
PDO mapping	Yes
Value range	No
Default value	No

Notes:

- The transmission type for mapping this object must be 255; otherwise, an Abort message (code 0604 0043h) will be transmitted.
- When this object is used to feed ECAM reference points to the drive, a working table with a border cannot be used. (Refer to the *SimplIQ Software Reference Manual* for more information.)

Object 0x2012: ***Binary interpreter input***
This object is a binary interpreter object (refer to **Chapter 9** concerning the byte stream).

- Object description:

Index	**2012h**
Name	Binary interpreter
Object code	VAR
Data type	Binary interpreter query (object 0x42)
Category	Mandatory

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	No
Default value	No

Object 0x2013: ***Binary interpreter output***
This object is a binary interpreter object (refer to **Chapter 9** concerning the byte stream).

- Object description:

Index	**2013h**
Name	Binary interpreter
Object code	RECORD
Data type	Binary interpreter command (object 0x43)
Category	Mandatory

- Entry description:

Access	Write only
PDO mapping	Yes
Value range	No
Default value	No

Object 0x2030: ***Recorder data***
This object is used to retrieve recorder parameters according to RC and the sub-index field. The 0x1 sub-index fetches the parameter, recorded in RC = (1 << sub-index).

- Object description:

Index	**2030h**
Name	Bring recorded data
Object code	RECORD
Data type	UNSIGNED32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of supported elements
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	UNSIGNED8
Default value	16

Sub-index	**1**
Description	Main speed
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**2**
Description	Main position
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**3**
Description	Position command
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**4**
Description	Digital input
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**5**
Description	Position error for UM=4, UM=5
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**6**
Description	Torque command
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**7**
Description	Bus voltage
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**8**
Description	Auxiliary position
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**9**
Description	Auxiliary speed
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**0A**
Description	Active current
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**0B**
Description	Reactive current
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**0C**
Description	Analog input 1
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1
Default value	

Sub-index	**0D**
Description	Reserved
Entry category	
Access	
PDO mapping	
Value range	
Default value	

Sub-index	**0E**
Description	Current phase A (IA value)
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1.
Default value	

Sub-index	**0F**
Description	Current phase B (IB value)
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1.
Default value	

Sub-index	**10**
Description	Speed command
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1.
Default value	

Sub-index	**0E**
Description	Current phase A (1A value)
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	Refer to Table 13-1.
Default value	

- **The bring data upload process:**
  The SDO Upload protocol is initiated as outlined in CiA DS 301. After confirmation, a character stream is transmitted. Each octet must be answered according to the Upload SDO segmented protocol, except sub-index 0, which returns object-supported entries in expedited SDO format.

  The segmented response is built from the header and data stream.

  The sub-index in this object is a value of the RC bit field.

  The header byte sequence is as follows:

Byte Number	**Description**	**Value**	**Type**
0 - 1	Variable type for user. Field has no practical significance.	0: Integer system parameter 1: Real system parameter 2: Integer user program variable 3: Real user program variable	Byte
2 - 3	Data width: number of hex character of single transmitted data item.	4: Short integer 8: Long integer	Byte
4 - 7	Data length: actual number of transmitted data items.		Word

Table 13-1: **Upload SDO Header Byte**

The rest of the byte sequence is the data stream. All bytes are transferred according to the scheme explained in section 2.9. Sub-index 0 uploads the supported object entries.

If the recorder variables are changed during upload, the process will be aborted.

Object 0x2040: ***Coordinate system group ID***
A 7-bit identifier is used for multicasting messages to a set of servo drives. For example, a network consists of three drives with CAN IDs 1, 2 and 3, and drives 1 and 2 are assigned a group ID of 4. A CAN master message to slave ID 4 will be received by drives 1 and 2. Drive 1 will respond with ID 1 and drive 2 will respond with ID 2.

The group ID of a drive may equal its ID. In order to set the group ID, first set object 0x2040, then send the command Reset Communication NMT service, followed by the Start Remote NMT service. The group ID becomes effective only at the next CAN network reset (NMT 82h).

- Object description:

Index	**2040h**
Name	ID for synchronized commands
Object code	VAR
Data type	UNSIGNED8
Category	

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	No
Default value	128

Object 0x2041: ***Amplifier-free running timer***
This object transmits the accurate 32-bit timer of the drive. The timer has a 1-microsecond resolution that is updated once in a real-time cycle. The accuracy of the report is 1 microsecond and the resolution is real time.

This object is typically used when the host wants to synchronize records for several slave nodes. It returns the precise time when the synchronous PDO data has been sampled (refer to Chapter 8).

- Object description:

Index	**2041h**
Name	Drive free-running timer
Object code	VAR
Data type	UNSIGNED32
Category	

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	No
Default value	No

Object 0x2082: ***CAN controller status***
This object provides the status of the CAN controller status register.

- Object description:

Index	**2082h**
Name	CAN controller status
Object code	VAR
Data type	UNSIGNED32
Category	Optional

- Entry description:

Access	Read only
PDO mapping	No
Value range	No
Default value	No

- Byte stream:
  The 32-bit number has the following structure:

Bits	**31…24**	**23…16**	**15…8**	**7…0**
Description	Network status	CAN transmit error counter	CAN receive error counter	CAN receiver flag

Bits 0 to 7 of the CAN receiver flag are as follows:

Bit	**Meaning**	**Remarks**
0	Not used	
1	Overrun flag, set when data overrun occurs	1: Data overrun detected 0: No data overrun
2	Bus off flag	1: CAN in bus off state 0: No bus off state
3	Transmitter error passive flag, set when CAN enters error passive state due to transmit error counter exceeding 127, and bus off flag is clear	1: CAN in transmitter error passive state 0: No transmitter error passive state
4	Receiver error passive flag, set when CAN enters error passive state due to receive error counter exceeding 127, and bus off flag is clear	1: CAN in receiver error passive state 0: No receiver error passive state
5	Transmitter warning flag	1: CAN in transmitter warning state 0: No transmitter warning
6	Receiver warning flag	1: CAN in receiver warning state 0: No receiver warning state
7	Not used	

Table 13-2: **CAN Receiver Flag Bit 0…7**

The CAN receive error counter (bits 8…15) reflects the status of the CAN receive error counter. The CAN transmit error counter (bits 16…23) reflects the status of the CAN transmit error counter.

The network status (bits 24…31) may be one of the following values:
1: Disconnected
2: Connected
3: Preparing
4: Stopped
5: Operational
127: Pre-operational

Object 0x208A: ***Begin time***

This object receives an absolute time for a simultaneous Motion Begin (BG). The time resolution for this object is 1 microsecond. For more information, refer to the BT command section in the *SimplIQ Command Reference Manual* and to **Chapter 8** of this manual.

- Object description:

Index	**208Ah**
Name	Begin time
Object code	VAR
Data type	UNSIGNED32
Category	Optional

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	No
Default value	No

Object 0x2090: ***Firmware download***
This object is used to download controller firmware using S-record format.

- Object description:

Index	**2090h**
Name	Download firmware
Object code	DOMAIN
Data type	Visible String
Category	Optional

- Entry description:

Access	Write only
PDO mapping	No
Value range	No
Default value	No

After the final character of each S-record line, the host must send the character 0x0A to indicate end-of-line. The next S-record can be sent immediately after 0xA.

Example:
First, the host initiates an SDO segmented sequence:

Byte 0				**Byte 4**			**Byte 7**
20	0x90	0x20	0x00	0x00	0x00	0x00	0x00

After the reply from the drive, the host transmits:

Byte 0				**Byte 4**			**Byte 7**
00	"S"	"3"	"6"	"A"	"3"	"9"	"1"

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

Byte 0				**Byte 4**			**Byte 7**
10	"8"	"5"	0x0A	"S"	"6"	"0"	"B"

Object 0x20A0: ***Auxiliary position actual value***

This object returns the actual position of the auxiliary axis (PY).

- Object description:

Index	**20A0h**
Name	Auxiliary position
Object code	VAR
Data type	SIGNED32
Category	Optional

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	$-10^9 \ldots 10^9$
Default value	No

Object 0x20A1: ***Main position error***

This object returns the error between the position command and the actual position (PE).

- Object description:

Index	**20A1h**
Name	Position error
Object code	VAR
Data type	SIGNED32
Category	Optional

- Entry description:

Access	Read only
PDO mapping	No
Value range	$-10^9 \ldots 10^9$
Default value	No

Object 0x2200: ***Digital input***
This object reports an active or non-active state of a digital input. It reflects the value of the IP command. The function and the logic level of a digital input are associated with the IL command.

When this object is mapped to a TPDO and the transmission type is set to asynchronous event, it is transmitted on every change of an input.

The report value of object 0x2200 is similar to the IP command. Please refer to the *SimplIQ* Command reference manual.

- Object description:

Index	**2200h**
Name	Digital inputs
Object code	VAR
Data type	UNSIGNED32
Category	Optional

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	0…0xFFFFFFFF
Default value	No

Object 0x2201: ***Digital input low byte***
This object defines simple digital inputs for drives.

The reflected functions are:

- Negative limit switch – Similar to RLS
- Positive limit switch – Similar to FLS
- Home switch – As reflected in the IL[5] command

- Object description:

Index	**2201h**
Name	Digital inputs low byte
Object code	VAR
Data type	UNSIGNED8
Category	Optional

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	0…0xFF
Default value	0

- Data description:

7 … 4	3	2	1	0
Reserved	Interlock	Home switch	Positive limit switch	Negative limit switch

MSB
The switch must be "active high."

Notes:

- The interlock is always 0.
- "Active high" means that the bit is set to high when the switch is logically active.
- Different *SimplIQ* drives support a different number of digital inputs. It is advised to use only the relevant bits according to the specific drive.
- This object is evaluated every 3 milliseconds.

When mapped as asynchronous, this object is transmitted at every change within the calculation resolution period. Inhibit time can be used to prevent busload or to control the time latency causing the same TPDO to be transmitted due to other asynchronous events.

Object 0x2F00: *User Integer*

This object provides an array of 24 integer numbers for general-purpose use.

- Object description:

Index	**2F00h**
Name	User Integer
Object code	ARRAY
Data type	Integer32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	24
Default value	24

Sub-index	**1-24**
Description	User Array
Entry category	Optional
Access	Read/write
PDO mapping	Yes
Value range	$[(-2^{30}+1)\ldots(2^{30}-1)]$
Default value	0

Object 0x2F01: *User Float Array*

This object provides an array of 24 floating numbers for general-purpose use.

- Object description:

Index	**2F01h**
Name	User Float Array
Object code	ARRAY
Data type	Floating Point (Float)
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read
PDO mapping	No
Value range	24
Default value	24

Sub-index	**1-24**
Description	User Array
Entry category	Optional
Access	Read/write
PDO mapping	Yes
Value range	[-1e20…1e20]
Default value	0

Object 0x2F02: ***ET Array***
This object enables ECAM table variables (ET[1] to ET[255]) to be loaded.

- Object description:

Index	**2F02h**
Name	ET Array
Object code	ARRAY
Data type	Integer 32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of entries
Entry category	Optional
Access	Read
PDO mapping	No
Value range	255
Default value	255

Sub-index	**1-255**
Description	User Array
Entry category	Optional
Access	Read/write
PDO mapping	Yes
Value range	
Default value	0

Object 0x2F11: P*VT head pointer*
This object informs the host of the index location of the last updated PVT message in the PVT table. According to this information, and with object 0x2F12, the host can determine the rate and location at which the PVT table should be updated. This object can only be used when mapped into a synchronized TPDO. It cannot be read or written with an SDO protocol.

- Object description:

Index	**2F11h**
Name	PVT head pointer
Object code	VAR
Data type	UNSIGNED16
Category	

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	No
Default value	No

Object 0x2F12: *PVT tail pointer*
This object informs the host of the index of the next read PVT message in the PVT table. According to this, and object 0x2F11, the host can determine the rate and location at which the PVT table should be updated. This object can only be used when mapped into a synchronized TPDO. It cannot be read or written with an SDO protocol.

- Object description:

Index	**2F12h**
Name	PVT tail pointer
Object code	VAR
Data type	UNSIGNED32
Category	

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	No
Default value	No

Object 0x2F15: ***Profile position remaining points***
This object indicates the number of points remaining for the profile movement in DSP 402 Profile Position mode, when working in Elmo's manufacture-specific buffer mode.

- Object description:

Index	**2F15h**
Name	Profile position remaining points
Object code	VAR
Data type	UNSIGNED16
Category	

- Entry description:

Access	Read only
PDO mapping	Yes
Value range	0…32
Default value	0

Object 0x2F20: ***PDO events***

This object is used to select the events that cause asynchronous PDOs to be transmitted, with transmission type 254. PDOs with other transmission types are ignored. Sub-indices 1 to 4 define events for transmitting TPDO1, TPDO2, TPDO3 and TPDO4, respectively.

The event definition for the PDO transmission is a bit field, as follows:

Bit	Event
0	Motion complete: MS = 0
1	Main homing complete: HM[1] = 0
2	Auxiliary homing complete: HY[1] = 0
3	Motor shut down by exception: MO = 0
4	Motor started: MO = 1
5	User program "emit" command
6	OS interpreter execution complete
7	Digital input event according to object 0x2F23
8…30	Reserved
31	Binary interpreter command processing complete

Asynchronous TPDOs obey the inhibit time restrictions, as explained in section 2.4.

- Object description:

Index	**2F20h**
Name	PDO events
Object code	ARRAY
Data type	UNSIGNED32
Category	Optional

- Entry description:

Sub-index	**0**
Description	Number of sub-indices
Entry category	Mandatory
Access	Read only
PDO mapping	No
Value range	4
Default value	4

Sub-index	**1**
Description	Events for PDO1 trigger
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	0…0x4FFFFFFF
Default value	0

Sub-index	**2**
Description	Events for PDO2 trigger
Entry category	Optional
Access	Read only
PDO mapping	No
Value range	
Default value	0x80000000

Sub-index	**3**
Description	Events for PDO3 trigger
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	0…0x4FFFFFFF
Default value	0

Sub-index	**4**
Description	Events for PDO4 trigger
Entry category	Optional
Access	Read/Write
PDO mapping	No
Value range	0…0x4FFFFFFF
Default value	0

Object 0x2F21: ***Emergency events***
This object selects events as the cause for transmitting emergency objects (see Chapter 6).

The driving event definition for an emergency is a bit field, as follows:

Bit	Event	Error Code	Error Register
0	CAN message lost (corrupted or overrun)	0x8110	0x11
1	Protocol error (unrecognized NMT request)	0x8200	0x11
2	Attempt to access an unconfigured RPDO	0x8210	0x21
3	Heartbeat event	0x8130	0x11
4	Fatal CPU error: stack overflow	0x6180	0x81
5	User program aborted by an error	0x6200	0x81
6	Request by user program "emit" function	0xFF01	0x81
7	Motor shut down by fault	See Table 13-3	See Table 13-3
8	Object mapped to an RPDO returned an error during interpretation or a referenced motion failed to be performed.	0x6300	0x01
9	DS 402 IP Underflow	0xFF02	0x21

This object does not control certain emergency messages, such as PVT motion errors. Some of the uncontrolled emergency objects include manufacturer-specific information.

The manufacturer error field for all controlled emergency messages is zero, except the user program EMCY() command, in which the program code determines the value of the field. The argument of the EMCY() function is written as an Unsigned32 number, into bytes 4 to 7 of the emergency message.

Notes:

- A "CAN message lost" emergency may indicate an overrun, in which a CAN message has not been retrieved from the receiver on time. The next message to the same buffer crashes with the as-yet unread message. Both messages may be lost in the crash. There may also be more lost messages that go undetected, because they may have be sent while the message loss indication was on. The "CAN message lost" message tells where a crash occurred, but it does not tell how many messages have actually been lost. The emergency message includes some information about the type of the "lost message" where it can. Data Field 2 in the EMCY objects contains information about the lost message. When the value is –1, the overrun occurred in the CAN controller hardware buffer. When the overrun is a software buffer overrun, data field 2 contains the COB-ID of the last message received and which caused the overrun.

- In case of a failed RPDO, the emergency message with data field 1 and data field 2 includes the following information:

0	Error code	0x00
1		0x63
2	Error register	0x01
3	Elmo error code, if 0 either no error or no related error for this emergency If > 0, error is according to the EC command	EC value
4	Error code data field 1	Object index
5		
6	Error code data field 2	Object sub-index
7		Interpreter error code, similar values as EC command.

If the profile is executed by the DSP 402 protocol, byte 3 of the Elmo error code is set to 0xff. The object index and sub-index are according to the failed profile:
0x607A: profile position
0x60FF: profile velocity

Motor Fault Description	Motor Fault Value (MF Command)	Error Code	Error Register
Resolver or Analog Encoder feedback failed	1	0x7300	0x81
Reserved	1	0x7305	0x21
Reserved	2	0x7306	0x21
Feedback loss: no match between encoder and Hall locations. Available in encoder + Hall feedback systems	4	0x7380	0x81
Peak current has been exceeded due to: ▪ Drive malfunction ▪ Badly-tuned current controller	8	0x8311	0x21
Disabled by Limit switch	0x10	0x5441	0x21
Reserved	0x20	0x5280	0x81
Two digital Hall sensors changed at once; only one sensor can be changed at a time.	0x40	0x7381	0x81
Speed tracking error DV[2]-VX (for UM=2, 4 or 5) exceeded speed error limit ER[2], due to: ▪ Bad tuning of speed controller ▪ Too tight a speed-error tolerance ▪ Inability of motor to accelerate to required speed because line voltage is too low, or	0x80	0x8480	0x81

Motor Fault Description	Motor Fault Value (MF Command)	Error Code	Error Register
motor is not powerful enough			
Position tracking error DV[3]-PX (UM5) or DV[3]-PY (UM=4) exceeded position error limit ER[3], due to: ▪ Bad tuning of position or speed controller ▪ Too tight a position error tolerance ▪ Abnormal motor load, a mechanical limit reached	0x100	0x8611	0x21
Cannot start due to inconsistent database. This type of database inconsistency is reflected in status SR report and in CD CPU dump report.	0x200	0x6320	0x21
Too large a difference in ECAM table entries.	0x400	0x5280	0x81
Heartbeat failure, occurring only if drive is set to "Abort under heartbeat failure" in a CANopen network (object 0x6007 in CAN object dictionary set to 1, malfunction).	0x800	0x8130	0x11
Cannot find electrical zero of motor when attempting to start motor with an incremental encoder and no digital Hall sensors. Applied motor current may not suffice for moving motor from its place.	0x10000	0x8380	0x81
Speed limit exceeded: VX < LL[2] or VX > HL[2]	0x20000	0x8481	0x81
Stack overflow: fatal exception. May occur if CPU cannot handle a real-time load due to too low a sampling time.	0x40000	0x6180	0x81
CPU exception: fatal exception.	0x8000	0x6181	0x81
Timing Error	0x100000	0x5281	0x81
Motor stuck: motor powered but not moving according to definition of CL[2] and CL[3].	0x200000	0x7121	0x21
Position limit exceeded: PX < LL[3] or PX > HL[3] (UM=5), or PY < LL[3] or PY > HL[3] (UM=4).	0x400000	0x8680	0x81
Reserved	0x8000000	0x1000	0x81
Cannot tune current offsets	0x10000000	0x8381	0x81
Cannot start motor	0x20000000	0xFF10	0x81
Reserved	0x800000	0x8680	0x81
Reserved	0x1000000	0x8312	0x3
Cannot start motor	0x20000000	0x5400	0x21
Under-voltage: power supply is shut down or it has too high an output impedance.	0x3000	0x3120	0x5

Motor Fault Description	Motor Fault Value (MF Command)	Error Code	Error Register
Over-voltage: power-supply voltage is too high or servo drive could not absorb kinetic energy while braking a load. A shunt resistor may be required.	0x5000	0x3310	0x5
Reserved	0x7000	0x3100	0x5
Reserved	0x9000	0x2311	0x3
Short circuit: motor or its wiring may be defective, or drive is faulty.	0xb000	0x2340	0x3
Temperature: drive overheating. The environment is too hot or heat removal is not efficient. Could be due to large thermal resistance between drive and its mounting.	0xd000	0x4310	0x9
Reserved	0xf000	0x5282	0x81

Table 13-3: **Motor Shut Down by Fault Emergency Error Codes**

The following CAN emergencies are supported in PVT/PT modes:

Error Code (Hex)	Error Code (Dec)	Reason	Data Field
0x56	86	Queue is low. Number of yet unexecuted PVT table rows has dropped below the value stated in MP[4].	Field 1: Write pointer Field 2: Read pointer
0x5b	91	Write pointer is out of physical range ([1…64]) of PVT table. Reason may be an improper setting of MP[6].	Value of MP[6]
0x5c	92	PDO 0x3xx is not mapped.	
0x34	52	An attempt has been made to program more PVT points than are available in queue.	Field 1: Index of PVT table entry that could not be programmed
0x7	7	Cannot initialize motion due to bad setup data. The write pointer is outside the range specified by the start and end pointers.	
0x8	8	Mode terminated and motor has been automatically stopped (in MO=1).	Data field 1: Write pointer Data field 2: 1: End of trajectory in non-cyclical mode 2: A zero or negative time

Error Code (Hex)	Error Code (Dec)	Reason	Data Field
			specified for a motion interval 3: Read pointer reached write pointer
0x9	9	A CAN message has been lost.	

Table 13-4: **PVT CAN Emergency Messages**

Setting a new value for object 0x2F21 deletes all emergency events queued for transmission.

- Object description:

Index	**2F21h**
Name	Emergency events
Object code	VAR
Data type	UNSIGNED16
Category	

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	No
Default value	0xFF (all emergencies on)

Object 0x2F22: ***Bus off time out***
This object defines the bus off timeout in milliseconds. If the device enters bus off, it will attempt to renew CAN communication after the specified time elapses. If the device recovers the CAN communication, the drive will enter pre-operational state; otherwise, it will remain disconnected. If the value of the object is set to zero, the timeout is infinite; that is, the device never recovers from the bus off.

A node goes into bus off state when more than 255 errors have been counted by the transmitter error counter. If this state, the node cannot participate in the bus activities. A node in bus off state can become "error active" (no longer in bus off state) with its error counter set to 0 after 128 occurrences of 11 consecutive recessive bits (logical 1) have been monitored on the bus.

When object 0x2F22 is 0, the node can never leave bus off state.

- Object description:

Index	**2F22h**
Name	Bus-off time out
Object code	VAR
Data type	UNSIGNED16
Category	

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	0…65,535
Default value	500

Object 0x2F23: ***Digital input TPDO event parameters***
When a TPDO is driven by a "digital input event" (DIN event) according to object 0x2F20, this object defines which digital input transitions will activate the TPDO.

Logic digital inputs (logic DINs) are physical or simulated input in values. The polarity for physical inputs is set according to the IL[N] command and to the debouncer according to the IF[N] command. The logic DINs are sampled each real-time cycle. The decisions about an event and its transmission are made in the background.

The following table lists the values used to define the digital input transitions:

DIN Mask Value	**DIN Logic Level for Activating an Event**
0	No event occurs.
1	Switched from low to high (↑).
2	Switched from high to low (↓).
3	Switched (↑ or ↓).

When a DIN event occurs, the mapped TPDO is transmitted. The DIN mask value is determined according to a 2-bit field of the digital input.

Example:
The value 0x00A1 sets the DIN4 event to "Event on every switch" and the DIN1 event to "Switch from low active to high active." every other digital input switch does not produce an event:

DIN6		DIN5		DIN4		DIN3		DIN2		DIN1	
0	0	0	0	1	1	0	0	0	0	0	1

TPDOs are transmitted only when the following steps are performed:

1. An object is mapped to the relevant TPDO and the transmission type is set to 254.
2. The logic level of the relevant digital input is adjusted (IL[N] command).
3. The event parameter of the input is determined by object 0x2F23.
4. The "digital input event" bit is set in object 0x20F0.

From this point on, as soon as the event occurs, the requested TPDO is transmitted.

Notes:

- If several inputs are selected, a DIN event will occur when at least one of the selected digital inputs is switched.
- A DIN event occurs according to switch level, regardless of the function selected for that switch (details given in the IL[N] and IP command sections of the *SimplIQ Command Reference Manual*).

- Object description:

Index	**2F23h**
Name	Digital input TPDO event parameters
Object code	VAR
Data type	UNSIGNED16
Category	

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	0x0FFF
Default value	0x0FFF (event on every switch)

Object 0x2F30: ***Last time stamp correction***
This object reads the difference between the last Sync time and the last Time Stamp. It serves to estimate how accurately the internal drive clock is locked onto the master clock.

- Object description:

Index	**2F30h**
Name	Last time stamp correction
Object code	VAR
Data type	UNSIGNED32
Category	

- Entry description:

Access	Read only
PDO mapping	No
Value range	No
Default value	No

Object 0x2F31: ***Last SYNC time***
This object is used with the internal microsecond counter of the drive, sampled at the last SYNC. It is useful when one drive is used to synchronize the entire network. To use this object for synchronization:

1. Broadcast a SYNC (object 80H).
2. Read this object from one drive (can be mapped to a TPDO).
3. Transmit the accepted value in a Time Stamp object (object 100H).

- Object description:

Index	**2F31h**
Name	Last SYNC time
Object code	VAR
Data type	UNSIGNED32
Category	

- Entry description:

Access	Read only
PDO mapping	No
Value range	No
Default value	0

Object 0x2F40: ***Configuration object***
This bit field object gives several configuration options to the drive. It resets to 0 after boot reset and must be set again in such cases

- Object description:

Index	**2F40h**
Name	Configuration object
Object code	VAR
Data type	UNSIGNED32
Category	

- Entry description:

Access	Read/Write
PDO mapping	No
Value range	No
Default value	No

- Data description:

Bit	**31** ... **1**	**0**
	Reserved	T Stamp master

- **T Stamp master**:
  The node is defined as a Time Stamp master. The drive's internal 32-bit, 1-microsecond resolution global timer is used as the system time stamp. The drive does not response to Time Stamp messages (COB-ID 0x100 or object 0x1013). In order to prevent motor jumps, setting this bit is allowed when the motor is off (MO=0).

Chapter 14: Error Control Protocol

For node guarding and life guarding, *SimplIQ* digital servo drives implement the heartbeat mechanism, as defined by CiA DS 301, version 4.0. With this process, a CAN slave can monitor other devices to determine if they are active, according to their periodic heartbeat messages. The CAN slave can send heartbeats to consumers, which may want to monitor if this slave is active.

Heartbeats are sent from the moment of initialization, even if the slave state is stopped. Object 0x1016 is a list of heartbeats that must be accepted from external devices. Failure to accept an expected heartbeat leads to behavior described by object 0x1029 and object 0x6007. Non-acceptance of a consumer heartbeat message at the expected time is called a "heartbeat event," which creates an emergency only if the relevant bit of EMCY is set to 1 (see object 0x2F21). A single emergency object may be transmitted per one heartbeat event. The heartbeat event is reset by one of the following events:

- A new heartbeat message arrives from the producer.
- The heartbeat consumer time object (0x1016) is written.

SimplIQ digital servo drives use a minimal implementation of object 0x1016. The sole heartbeat that it monitors is that of the network master. Object 0x1017 sets the period for sending heartbeats.

A heartbeat is a single-byte message, with the following options:

Heartbeat	Description
0	Boot-up
4	Stopped
5	Operational
127	Pre-operational

The COB-ID for a heartbeat message is 0x700 + Device ID. For example, a first servo drive with the unit ID of 1 will have heartbeat COB-ID of 0x701 = 1793.

Chapter 15: Downloading Firmware

New firmware versions can be loaded via CAN communications by writing the new firmware as S-records to object 0x2090. The S-records are written as string SDOs to object 0x2090.

After the firmware is downloaded, the drive continues to communicate using the previous firmware. If the download fails, a retry is possible.

Chapter 16: Initial CAN Communication Setup

16.1 Setup Using RS-232

All communication parameters — such as the CAN baud rate for the targets — are programmed via the PP[N] command. In order to program the communication parameters, communication must first be established with the servo drive, so that the RS-232 communication channel, which is always active, can be used. In cases where the servo drive is programmed to RS-232 parameters that differ form the default, the servo drive can be forced to its default RS-232 communication parameters, as outlined in the *SimplIQ Software Manual*. It is also possible to try all the supported CAN baud rates until the unit responds.

The following parameters affect CAN communication:

Parameter	Description	Range
PP[13]	Servo drive CAN ID	1…127
PP[14]	CAN baud rate	1…8 8: 800,000 7: 50,000 6: 50,000 5: 50,000 4: 100,000 3: 125,000 2: 250,000 1: 500,000 0: 1,000,000
PP[15]	CAN group ID	1…127
PP[16]	Standard/extended arbitration field	Must be zero (only standard arbitration presently supported)

Table 16-1: **CAN Communication Parameters**

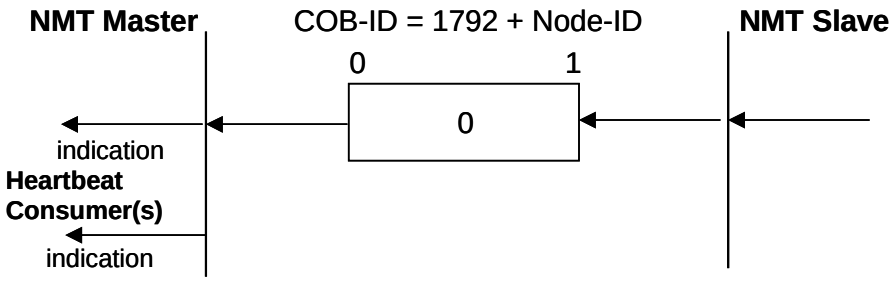
Setting the PP[13] and PP[14] parameters does not result in immediately changes. In order to activate the new communication parameters, an NMT reset communication command must be sent, or the servo drive must be rebooted.

If the servo drive is to be rebooted, either by the NMT command or by a power-on sequence, the SV (save parameters) command must be used so that the communication parameters become permanent.

In order to start CAN PDO communication, an NMT network start command must be issued. Afterwards, the servo drive is in CAN operational state.

16.2 Bootup Protocol

This protocol is used to signal that an NMT salve has entered the pre-operational node state after the initializing state. The protocol uses the same identifier as the error control protocols.



One data byte is transmitted with value 0.

Appendix: Little and Big Endians

The "end" in "endians" refers to the address of the most significant or least significant byte in a multiple-byte data type (such as short, long or float). The address of big endians is the most significant byte (the "big" end) while the address of little endians is the least significant byte (the "little" end).

Example using standard C conventions:

```
long lType;
short sType[2]
char cType[5] = "ABCD";

lType = 0x12345678;
sType[1] = 0x1234 ;
sType[2] = 0x5678 ;
```

Memory structure: big endians (starts with MSB):

00 08
12 34 56 78 - lType
12 34 56 78 - sType
AB CD 0 - cType

Memory structure: little endians (starts with LSB):

00 08
78 56 34 12 - lType
34 12 78 56 - sType
AB CD 0 - cType

The CANopen protocol supports the little endian method; for example, a node replies to a TPDO with three objects:
Object 1: Signed24 - 0x12ABCD
Object 2: Unsigned32 - 0x123456AB
Object 3: Unsigned8 - 0x1F

The CAN octet will be as follows:
CD AB 12 AB 56 34 12 1F