

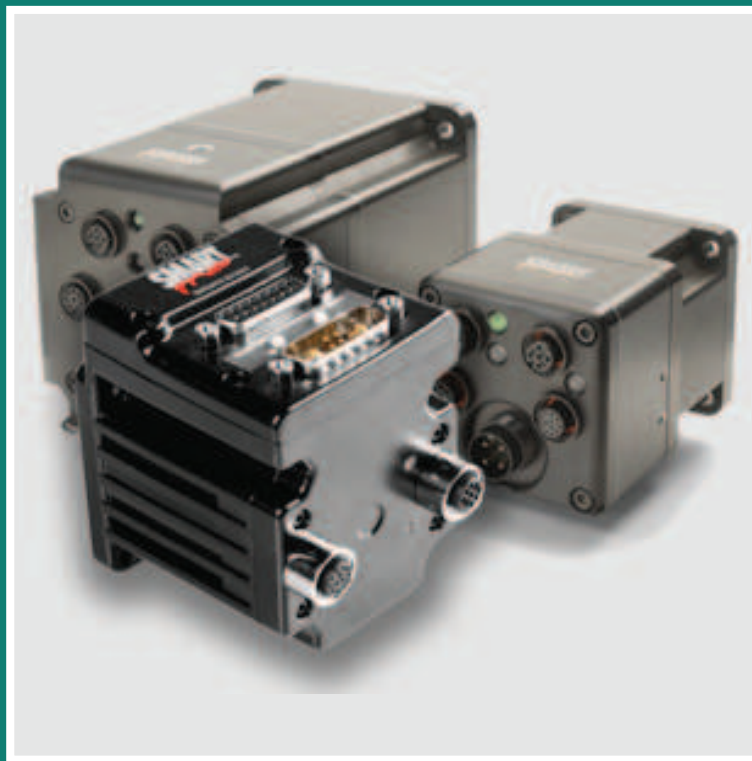
For the mobile version of this guide, see:
animatics.com/docs/guides-html/c5_canopen/

SMART
Motor™

CANopen®
Guide

Class 5 SmartMotor
Technology with

COMBITRONIC™



Copyright Notice

©2013-2019, Moog Inc., Animatics.

Moog Animatics Class 5 SmartMotor™ CANopen Guide, Rev. H, PN:SC80100001-001.

This manual, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. The content of this manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Moog Inc., Animatics. Moog Inc., Animatics assumes no responsibility or liability for any errors or inaccuracies that may appear herein.

Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Moog Inc., Animatics.

The programs and code samples in this manual are provided for example purposes only. It is the user's responsibility to decide if a particular code sample or program applies to the application being developed and to adjust the values to fit that application.

Moog Animatics and the Moog Animatics logo, SmartMotor and the SmartMotor logo, Combitronic and the Combitronic logo are all trademarks of Moog Inc., Animatics. CiA and CANopen are registered community trademarks of CAN in Automation e.V. Other trademarks are the property of their respective owners.

Please let us know if you find any errors or omissions in this manual so that we can improve it for future readers. Such notifications should contain the words "CANopen Guide" in the subject line and be sent by e-mail to: animatics_marcom@moog.com. Thank you in advance for your contribution.

Contact Us:

Americas - West

Moog Animatics
2581 Leghorn Street
Mountain View, CA 94043
USA
Tel: 1 650-960-4215

Americas - East

Moog Animatics
750 West Sproul Road
Springfield, PA 19064
USA
Tel: 1 610-328-4000 x3999
Fax: 1 610-605-6216

Support: 1 (888) 356-0357

Website: www.animatics.com

Email: animatics_sales@moog.com

Table of Contents

Introduction	10
Purpose	11
Combitronic Technology	11
I/O Device CAN Bus Master	11
Time Sync for Electronic Gearing and Camming	12
Abbreviations	14
Safety Information	15
Safety Symbols	15
Other Safety Considerations	15
Motor Sizing	15
Environmental Considerations	15
Machine Safety	16
Documentation and Training	17
Additional Equipment and Considerations	17
Safety Information Resources	17
Additional Documents	18
Related Guides	18
Other Documents	18
Additional Resources	19
CANopen Resources	19
CANopen Overview	20
CANopen Description	21
CAN (CAN Bus)	21
CANopen	21
PDO and SDO Communication	22
SDO	22
PDO	23
COB-ID Allocation	24
NMT States	26
NMT Control	27
NMT Summary	27
NMT State Machine Diagram	28
PDO Communications	28
Peer-to-Peer Communications	29
Synchronous Communications	29

Supported Features	30
Supported	31
Motion Modes	31
NMT State Machine Master	31
PDO Transmit on Event	31
PDO Transmit on Timer Only	32
PDO Transmit on Sync	32
Dynamic PDO Mapping	32
Heartbeat Producer	32
Sync Producer	32
Not Supported	33
Emergency Messages	33
Saving Parameters	33
Heartbeat Consumer	33
MPDO Communications	33
CAN Bus Bit Rate	33
PDO Transmit on RTR (Remote frames)	33
Node Guarding	33
TIME Service	33
Sync Start	33
Connections, Wiring and Status LEDs	34
Connectors and Pinouts	35
D-Style Motor Connectors and Pinouts	35
D-Style Motors: CDS Option Schematic	35
CDS on the DA-15 Connector	35
CDS on the 7W2 Connector (CDS7)	36
M-Style Motor Connectors and Pinouts	39
Cable Diagram	39
CAN Multidrop Cable Diagram	40
Bus Termination	40
Maximum Bus Length	41
Status LEDs	42
Other Communications with the Motor	43
Manufacturer-Specific Objects	44
I/O	45
User Variables	45

Calling Subroutines	47
Command Interface (Object 2500h)	48
Command Interface	48
Program Upload/Download	49
Upload from Motor	49
Download to Motor	49
CiA 402 Drive and Motion Control Profile	50
CiA 402 Profile Motion State Machine	51
Control Words, Status Words and the Drive State Machine	51
Status Word (Object 6041h)	52
Control Word (Object 6040h)	53
Motion Profiles	54
Position Mode	54
Absolute Position Mode Summary	55
Absolute Position Mode Example	55
Relative Position Example	57
Velocity Mode	58
Velocity Mode Summary	59
Velocity Mode Example	59
Torque Mode	60
Torque Mode Summary	61
Torque Mode Example	61
Interpolated Position Mode	62
Interpolated Position Mode Summary	63
Example: Short Run on a Single Motor	64
Example: Continuous Run on a Single Motor	65
Example: Resuming Motion in IP Mode	66
Synchronization	66
User Bits	67
Splining	68
Variable-Length Segments	68
Homing Mode	68
Homing Summary	69
Homing Example	69
PDO Mapping	71
Overview	72
Mapping and Communication Parameters Objects	73

Communications Parameters Objects	74
Mapping Parameters Objects	75
Mapping Entries	75
Mapping Procedure	76
Time Sync Motors Mapping Procedure	76
Example Start-up Sequence	78
CANopen User Program Commands	79
Address and Baud Rate Commands	80
CADDR=frm	80
CBAUD=frm	80
CAN Error Reporting Commands	80
=CAN, RCAN	80
RB(2,4), x=B(2,4)	83
Network Control Commands	83
CANCTL(action, value)	83
NMT(address, command code)	85
SDORD(address, obj index, subindex, bytecount)	86
SDOWR(address, obj index, subindex, bytecount, data)	86
Exceptions to NMT, SDORD and SDOWR Commands	87
Troubleshooting	88
SDO Response Error Codes	90
Object Reference	92
Object Categories	96
Communication Profile	97
Object 1000h: Device Type	99
Object 1001h: Error Register	100
Object 1005h: COB-ID SYNC	101
Object 1006h: Communication Cycle Period	103
Object 1008h: Manufacturer Device Name	105
Object 1009h: Manufacturer Hardware Version	106
Object 100Ah: Manufacturer Software Version	107
Object 1013h: High-Resolution Timestamp	108
Object 1017h: Producer Heartbeat Time	109
Object 1018h: Identity Object	110
Object 1200h: Server SDO Parameter 1	111
Object 1400h: Receive PDO Communication Parameter 1	112

Object 1401h: Receive PDO Communication Parameter 2	113
Object 1402h: Receive PDO Communication Parameter 3	114
Object 1403h: Receive PDO Communication Parameter 4	115
Object 1404h: Receive PDO Communication Parameter 5	116
Object 1600h: Receive PDO Mapping Parameter 1	117
Object 1601h: Receive PDO Mapping Parameter 2	118
Object 1602h: Receive PDO Mapping Parameter 3	119
Object 1603h: Receive PDO Mapping Parameter 4	120
Object 1604h: Receive PDO Mapping Parameter 5	121
Object 1800h: Transmit PDO Communication Parameter 1	122
Object 1801h: Transmit PDO Communication Parameter 2	123
Object 1802h: Transmit PDO Communication Parameter 3	124
Object 1803h: Transmit PDO Communication Parameter 4	125
Object 1804h: Transmit PDO Communication Parameter 5	126
Object 1A00h: Transmit PDO Mapping Parameter 1	127
Object 1A01h: Transmit PDO Mapping Parameter 2	128
Object 1A02h: Transmit PDO Mapping Parameter 3	129
Object 1A03h: Transmit PDO Mapping Parameter 4	130
Object 1A04h: Transmit PDO Mapping Parameter 5	131
Manufacturer-Specific Profile	132
Object 2000h: Node Id	134
Object 2001h: Bit Rate Index	135
Object 2100h: Port Configuration	136
Object 2101h: Bit IO	137
Object 2200h: User EEPROM	138
Object 2201h: User Variable	139
Object 2202h: Set Position Origin	140
Object 2203h: Shift Position Origin	141
Object 2204h: Mappable 32-bit Variables	142
Object 2205h Negative Software Position Limit	143
Object 2206h Positive Software Position Limit	144
Object 2207h Encoder Modulo Limit	145
Object 2208h Encoder Follow Data	146
Object 2209h Encoder Follow Control	147
Start/Stop Capability	147
Object 220Ah MFMUL	149
Object 220Bh MFDIV	150
Object 220Ch MFA	151

Object 220Dh MFD	152
Object 2220h: 8-Bit Mappable Variables	153
Object 2221h: 16-Bit Mappable Variables	154
Object 2300h: Bus Voltage	155
Object 2301h: RMS Current	156
Object 2302h: Internal Temperature	157
Object 2303h: Internal Clock	158
Object 2304h: Motor Status	159
Object 2305h: Motor Control	168
Object 2306h: Motor Subroutine Index	169
Object 2307h: Sample Period	170
Object 2308h: Microsecond Clock	171
Object 2309h: GOSUB R2	172
Object 2400h: Interpolation Mode Status	173
Object 2401h: Buffer Control	174
Object 2402h: Buffer Setpoint	175
Object 2403h: Interpolation User Bits	176
Object 2404h: Interpolation Sample Clock	177
Object 2500h: Encapsulated SmartMotor Command	178
Drive and Motion Control Profile	179
Object 6040h: Control Word	181
Object 6041h: Status Word	183
Object 605Ah: Quick Stop Option Code	184
Object 605Dh: Halt Option Code	185
Object 605Eh: Fault Reaction Option Code	186
Object 6060h: Modes of Operation	187
Object 6061h: Modes of Operation Display	189
Object 6062h: Position Demand Value	190
Object 6063h: Position Actual Internal Value	191
Object 6064h: Position Actual Value	192
Object 6065h: Following Error Window	193
Object 606Bh: Velocity Demand Value	194
Object 606Ch: Velocity Actual Value	195
Object 6071h: Target Torque	196
Object 6074h: Torque Demand Value	197
Object 6077h: Torque Actual	198
Object 6079h: DC Link Circuit Voltage	199
Object 607Ah: Target Position	200

Object 607Ch: Home Offset	201
Object 6080h: Max Motor Speed	203
Object 6081h: Profile Velocity in PP Mode	204
Object 6083h: Profile Acceleration	205
Object 6084h: Profile Deceleration	206
Object 6085h: Quick Stop Deceleration	207
Object 6087h: Torque Slope	208
Object 608Fh: Position Encoder Resolution	209
Object 6098h: Homing Method	210
Object 6099h: Homing Speeds	213
Object 609Ah: Homing Acceleration	214
Object 60C0h: Interpolation Sub-Mode Select	215
Object 60C1h: Interpolation Data Record	216
Object 60C2h: Interpolation Time Period	217
Object 60C4h: Interpolation Data Configuration	219
Object 60F4h: Following Error Actual Value	220
Object 60FBh: Position Control Parameter Set	221
Object 60FCh: Position Demand Internal Value	223
Object Description	223
Entry Description	223
Object 60FDh: Digital Inputs	224
Object 60FEh: Digital Outputs	226
Object 60FFh: Target Velocity	228
Object 6402h: Motor Type	229
Object 6502h: Supported Drive Modes	230
Object 67FFh: Single Device Type	231
Reference Documents	232

Introduction

This chapter provides information on the purpose and scope of this manual. It also provides information on safety notation, related documents and additional resources.

Purpose	11
Combitronic Technology	11
I/O Device CAN Bus Master	11
Time Sync for Electronic Gearing and Camming	12
Abbreviations	14
Safety Information	15
Safety Symbols	15
Other Safety Considerations	15
Motor Sizing	15
Environmental Considerations	15
Machine Safety	16
Documentation and Training	17
Additional Equipment and Considerations	17
Safety Information Resources	17
Additional Documents	18
Related Guides	18
Other Documents	18
Additional Resources	19
CANopen Resources	19

Purpose

This manual explains the Moog Animatics Class 5 SmartMotor™ support for the CANopen® protocol. It describes the major concepts that must be understood to integrate a SmartMotor slave with a PLC or other CANopen master. However, it does not cover all the low-level details of the CANopen protocol.

NOTE: The feature set described in this version of the manual refers to motor firmware 5.x.4.30 or later.

This manual is intended for programmers or system developers who have read and understand the CiA 402 CANopen specification. Therefore, this manual is not a tutorial on that specification or the CANopen protocol. Instead, it should be used to understand the specific implementation details for the Moog Animatics SmartMotor. Additionally, examples are provided for the various modes of motion and accessing those modes through CANopen to operate the SmartMotor.

The Object Reference chapter of this manual includes details about the specific objects available in the SmartMotor through CANopen. The objects include those required by CANopen, the CiA 402 motion profile, and manufacturer-specific objects added by Moog Animatics. For details, see Object Reference on page 92.

Combitronic Technology

The most unique feature of the SmartMotor is its ability to communicate with other SmartMotors and share resources using Moog Animatics' Combitronic™ technology. Combitronic is a protocol that operates over a standard CAN interface. It may coexist with either CANopen or DeviceNet protocols. It requires no single dedicated master to operate. Each SmartMotor connected to the same network communicates on an equal footing, sharing all information, and therefore, sharing all processing resources.

For additional details, see the *SmartMotor™ Developer's Guide*.

I/O Device CAN Bus Master

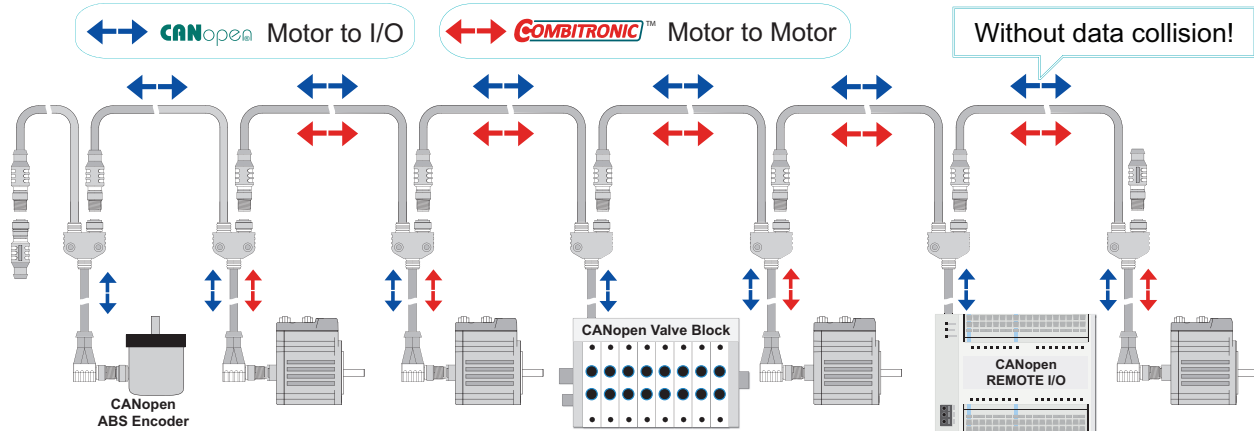
For firmware version 5.x.4.30 or later, the SmartMotor can interface with standard CiA 301 CANopen devices, such as CANopen valve blocks, CANopen I/O blocks, CANopen encoders, and many other devices. This means through CAN and Combitronic communications, you now have full machine control with just a SmartMotor as the CAN bus master—no other external bus master is required. Objects and commands have been added/modified to provide this functionality.

NOTE: This capability is currently available on Class 5 SmartMotors only.

Basic control allows 8, 16, or 32-bit sized data objects with support for both PDO and SDO protocols. The supported profiles include but are not limited to I/O profile, encoder profile, and DS4xx profile. This provides the ability to:

- Dynamically map SmartMotor PDOs, map another device's PDOs, start the NMT state
- A SmartMotor can send/receive up to 5 PDOs each for Rx (receive) and Tx (transmit)
- Read/write SDOs in expedited mode only, which works for up to 32-bit data

Multiple SmartMotors and multiple I/O devices may be on the same CAN bus. This combined with Combitronic motor-to-motor communications allows for complex, multi-axis, multi-I/O-device network control. Refer to the following figure.



Be sure to follow proper guidelines for CAN bus cabling and termination.

SmartMotor as a CAN Bus Master

Related objects are: 2220h, 2221h and 2204h. For details, see Object Reference on page 92.

Related commands are: NMT, SDORD, SDOWR, CANCTL, and B/RB. For details, see the descriptions in this guide and in the *SmartMotor Developer's Guide*.

Example user programs are shown in the *SmartMotor Developer's Guide*, Part 3: Examples.

Time Sync for Electronic Gearing and Camming

Beginning with firmware 5.x.4.30 or later, the SmartMotor provides precise time synchronization between motors for electronic gearing and camming applications (for example, traverse and take-up spooling).

NOTE: This capability is currently available on Class 5 SmartMotors only.

The CANopen objects related to this are:

- 1005h: Specifies the COB-ID used for the Synchronization object (transmit or receive).
- 1006h: Defines the communication cycle period in microseconds for transmission of the sync message.
- 2207h: Defines the encoder modulo limit in units of encoder counts.
- 2208h: Accepts data from a network (CANopen) based encoder. Three different data sizes are provided to handle PDO mapping to data sources of 8, 16, and 32 bits.
- 2209h: Controls the behavior for the mode of following a network encoder and/or use of objects 220Ch and 220Dh.
- 220Ah: Specifies the multiplier for external encoder mode follow with ratio MFMUL/MFDIV.
- 220Bh: Specifies the divisor for external encoder mode follow with ratio MFMUL/MFDIV.

- 220Ch: Sets the ascend ramp to the specified sync ratio from a ratio of zero.
- 220Dh: Sets the descend ramp from the specified sync ratio to a ratio of zero.

For details on these objects, refer to the corresponding object descriptions in the Object Reference chapter of this guide.

For a detailed description of motor following, electronic gearing and camming operations, refer to the *SmartMotor Developer's Guide*.

For an example PDO mapping and application start up sequence, see Time Sync Motors Mapping Procedure on page 76. This is based on an external PLC/master.

An example user program is shown in the *SmartMotor Developer's Guide*, Part 3: Examples. This is based on a SmartMotor acting as the CANopen master.

Abbreviations

The following table provides a list of abbreviations used in this manual and their descriptions.

Abbreviation	Description
ACK	Acknowledgment
ADU	Acceleration/Deceleration Units
CiA	CAN in Automation
COB	Communication Object
COB-ID	Communication Object Identification
CSP	Cyclic Synchronous Position (mode)
CST	Cyclic Synchronous Torque (mode)
CSV	Cyclic Synchronous Velocity (mode)
DC	Direct Current
FSA	Finite State Automaton
HM	Homing (mode)
IN	Input
INIT	Initialization (state)
NMT	Network Management (state)
OP	Operational (state)
OUT	Output
PDO	Process Data Object
PDS	Power Drive System
PDS FSA	Power Drive System Finite State Automaton
PP	Profile Position (mode)
PREOP	Pre-Operational (state)
PU	Position Units
PV	Profile Velocity (mode)
RxPDO	Receive PDO
SDO	Service Data Object
SMI	SmartMotor Interface (software)
TQ	Torque (mode)
TxPDO	Transmit PDO
VU	Velocity Units

Safety Information

This section describes the safety symbols and other safety information.

Safety Symbols

The manual may use one or more of the following safety symbols:



WARNING: This symbol indicates a potentially nonlethal mechanical hazard, where failure to follow the instructions could result in serious injury to the operator or major damage to the equipment.



CAUTION: This symbol indicates a potentially minor hazard, where failure to follow the instructions could result in slight injury to the operator or minor damage to the equipment.

NOTE: Notes are used to emphasize non-safety concepts or related information.

Other Safety Considerations

The Moog Animatics SmartMotors are supplied as components that are intended for use in an automated machine or system. As such, it is beyond the scope of this manual to attempt to cover all the safety standards and considerations that are part of the overall machine/system design and manufacturing safety. Therefore, the following information is intended to be used only as a general guideline for the machine/system designer.

It is the responsibility of the machine/system designer to perform a thorough "Risk Assessment" and to ensure that the machine/system and its safeguards comply with the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the locale where the machine is being installed and operated. For more details, see Machine Safety on page 16.

Motor Sizing

It is the responsibility of the machine/system designer to select SmartMotors that are properly sized for the specific application. Undersized motors may: perform poorly, cause excessive downtime or cause unsafe operating conditions by not being able to handle the loads placed on them. The *System Best Practices* document, which is available on the Moog Animatics website, contains information and equations that can be used for selecting the appropriate motor for the application.

Replacement motors must have the same specifications and firmware version used in the approved and validated system. Specification changes or firmware upgrades require the approval of the system designer and may require another Risk Assessment.

Environmental Considerations

It is the responsibility of the machine/system designer to evaluate the intended operating environment for dust, high-humidity or presence of water (for example, a food-processing environment that requires water or steam wash down of equipment), corrosives or chemicals that may come in contact with the machine, etc. Moog Animatics manufactures specialized

IP-rated motors for operating in extreme conditions. For details, see the *Moog Animatics Product Catalog*, which is available on the Moog Animatics website.

Machine Safety

In order to protect personnel from any safety hazards in the machine or system, the machine/system builder must perform a "Risk Assessment", which is often based on the ISO 13849 standard. The design/implementation of barriers, emergency stop (E-stop) mechanisms and other safeguards will be driven by the Risk Assessment and the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the locale where the machine is being installed and operated. The methodology and details of such an assessment are beyond the scope of this manual. However, there are various sources of Risk Assessment information available in print and on the internet.

NOTE: The following list is an example of items that would be evaluated when performing the Risk Assessment. Additional items may be required. The safeguards must ensure the safety of all personnel who may come in contact with or be in the vicinity of the machine.

In general, the machine/system safeguards must:

- Provide a barrier to prevent unauthorized entry or access to the machine or system. The barrier must be designed so that personnel cannot reach into any identified danger zones.
- Position the control panel so that it is outside the barrier area but located for an unrestricted view of the moving mechanism. The control panel must include an E-stop mechanism. Buttons that start the machine must be protected from accidental activation.
- Provide E-stop mechanisms located at the control panel and at other points around the perimeter of the barrier that will stop all machine movement when tripped.
- Provide appropriate sensors and interlocks on gates or other points of entry into the protected zone that will stop all machine movement when tripped.
- Ensure that if a portable control/programming device is supplied (for example, a hand-held operator/programmer pendant), the device is equipped with an E-stop mechanism.

NOTE: A portable operation/programming device requires *many* additional system design considerations and safeguards beyond those listed in this section. For details, see the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the locale where the machine is being installed and operated.

- Prevent contact with moving mechanisms (for example, arms, gears, belts, pulleys, tooling, etc.).
- Prevent contact with a part that is thrown from the machine tooling or other part-handling equipment.
- Prevent contact with any electrical, hydraulic, pneumatic, thermal, chemical or other hazards that may be present at the machine.
- Prevent unauthorized access to wiring and power-supply cabinets, electrical boxes, etc.

- Provide a proper control system, program logic and error checking to ensure the safety of all personnel and equipment (for example, to prevent a run-away condition). The control system must be designed so that it does not automatically restart the machine/system after a power failure.
- Prevent unauthorized access or changes to the control system or software.

Documentation and Training

It is the responsibility of the machine/system designer to provide documentation on safety, operation, maintenance and programming, along with training for all machine operators, maintenance technicians, programmers, and other personnel who may have access to the machine. This documentation must include proper lockout/tagout procedures for maintenance and programming operations.

It is the responsibility of the operating company to ensure that:

- All operators, maintenance technicians, programmers and other personnel are tested and qualified before acquiring access to the machine or system.
- The above personnel perform their assigned functions in a responsible and safe manner to comply with the procedures in the supplied documentation and the company safety practices.
- The equipment is maintained as described in the documentation and training supplied by the machine/system designer.

Additional Equipment and Considerations

The Risk Assessment and the operating company's standard safety policies will dictate the need for additional equipment. In general, it is the responsibility of the operating company to ensure that:

- Unauthorized access to the machine is prevented at all times.
- The personnel are supplied with the proper equipment for the environment and their job functions, which may include: safety glasses, hearing protection, safety footwear, smocks or aprons, gloves, hard hats and other protective gear.
- The work area is equipped with proper safety equipment such as first aid equipment, fire suppression equipment, emergency eye wash and full-body wash stations, etc.
- There are no modifications made to the machine or system without proper engineering evaluation for design, safety, reliability, etc., and a Risk Assessment.

Safety Information Resources

Additional SmartMotor safety information can be found on the Moog Animatics website; open the file "109_Controls, Warnings and Cautions.pdf" located at:

<http://www.animatics.com/support/moog-animatics-catalog.html>

OSHA standards information can be found at:

<https://www.osha.gov/law-regs.html>

ANSI-RIA robotic safety information can be found at:

<http://www.robotics.org/robotic-content.cfm/Robotics/Safety-Compliance/id/23>

UL standards information can be found at:

<http://ulstandards.ul.com/standards-catalog/>

ISO standards information can be found at:

<http://www.iso.org/iso/home/standards.htm>

EU standards information can be found at:

http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/index_en.htm

Additional Documents

The Moog Animatics website contains additional documents that are related to the information in this manual. Please refer to the following list.

Related Guides

- *Class 5 SmartMotor™ Installation & Startup Guide*
<http://www.animatics.com/cl-5-install-startup-guide>
- *SmartMotor™ Developer's Guide*
<http://www.animatics.com/smartmotor-developers-guide>
- *SmartMotor™ System Best Practices*
<http://www.animatics.com/system-best-practices-application-note>

Other Documents

- *SmartMotor™ Product Certificate of Conformance*
<http://www.animatics.com/download/Declaration of Conformity.pdf>
- *SmartMotor™ UL Certification*
http://www.animatics.com/download/MA_UL_online_listing.pdf
- *SmartMotor Developer's Worksheet*
(interactive tools to assist developer: Scale Factor Calculator, Status Words, CAN Port Status, Serial Port Status, RMODE Decoder and Syntax Error Codes)
<http://www.animatics.com/tools>
- *Moog Animatics Product Catalog*, which is available on the Moog Animatics website
<http://www.animatics.com/support/moog-animatics-catalog.html>

Additional Resources

The Moog Animatics website contains useful resources such as product information, documentation, product support and more. Please refer to the following addresses:

- General company information:
<http://www.animatics.com>
- Product information:
<http://www.animatics.com/products.html>
- Product support (Downloads, How To videos, Forums, Knowledge Base, and FAQs):
<http://www.animatics.com/support.html>
- Sales and distributor information:
<http://www.animatics.com/sales-offices.html>
- Application ideas (including videos and sample programs):
<http://www.animatics.com/applications.html>

CANopen Resources

CANopen is a common standard maintained by CAN in Automation (CiA):

- CAN in Automation website:
<http://www.can-cia.org/>
- CAN in Automation website — CANopen description:
<http://www.can-cia.org/index.php?id=canopen>

CANopen Overview

This chapter provides an overview of the CANopen communications protocol implementation on the Moog Animatics SmartMotor.

CANopen Description	21
CAN (CAN Bus)	21
CANopen	21
PDO and SDO Communication	22
SDO	22
PDO	23
COB-ID Allocation	24
NMT States	26
NMT Control	27
NMT Summary	27
NMT State Machine Diagram	28
PDO Communications	28
Peer-to-Peer Communications	29
Synchronous Communications	29

CANopen Description

CANopen is a standard that allows industrial devices to communicate over the CAN bus (the CAN bus alone does not provide enough functionality for most industrial applications).

The terms CANopen, CAN and CAN bus are often used interchangeably in technical conversations, but they are not the same. Therefore, it is important to understand their differences, which are described in the next two sections.

CAN (CAN Bus)

CAN or CAN bus is a low-level communication system. It defines a set of electrical standards (voltages, differential signaling method, impedance, etc.) as well as some very basic data formatting. The data formatting permits up to eight bytes of data in a packet. This packet is transmitted with an 11-bit identifier. There is no "to" or "from" field to indicate a specific destination for a packet. A device can also transmit several different sets of data, each with a unique identifier. The identifier essentially gives that data a unique meaning. However, that meaning can depend entirely on the intent of the system designer.

Each device on the network can decide what data it wants to monitor. Typical CAN bus hardware provides mechanisms to the software for filtering out specific identifiers. CAN also provides features that detect errors to ensure data integrity.

When two devices attempt to transmit at the same time (which causes collisions), the device sending data with a lower identifier will continue, while the other device will stop transmitting and retry as soon as possible. This simple arbitration is reliable and efficient without introducing unpredictable delays, which makes it suitable for industrial networks.



CAUTION: Two devices should never transmit with the same identifier. If that occurs, then the situation cannot be resolved and will cause a network error.

CANopen

CANopen builds onto the basic CAN bus functionality. It also defines events driven by timers and synchronization signals.

An address is assigned to each device on the network. This address allows a client-server relationship to be established from a master to each device (SDO, NMT, etc.). This relationship allows device configuration at startup so that process-specific data can be exchanged later through PDO communications.

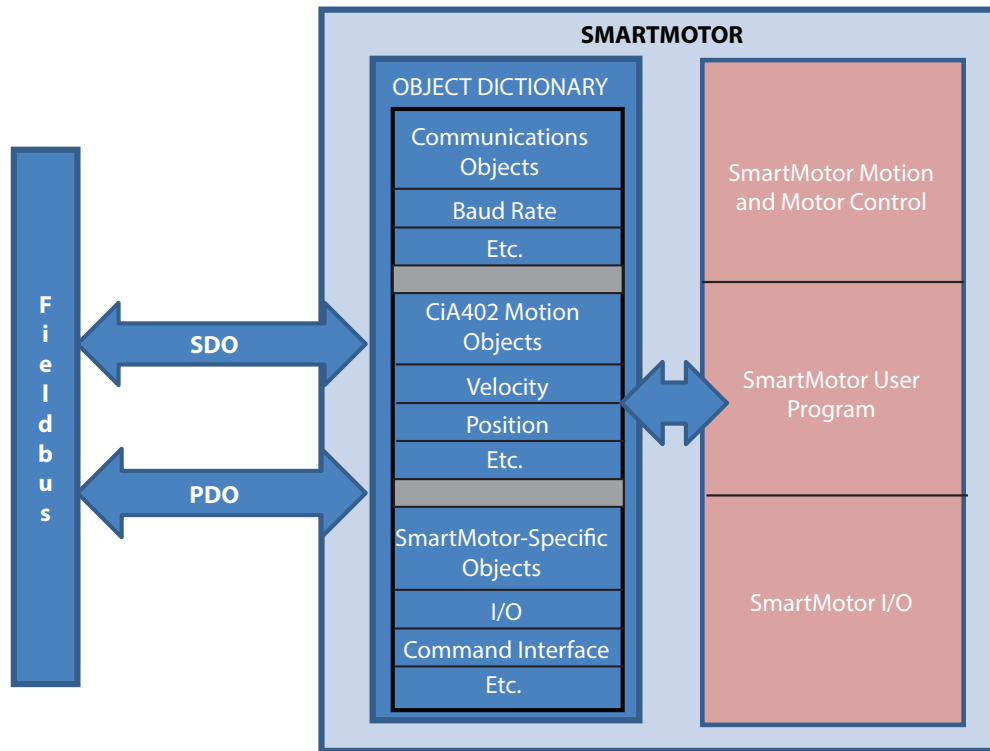
All data in a device is organized into a common list of available objects. This is called the "object library" or "object dictionary". It allows the master to obtain some basic information directly from the device such as range limits and descriptions.

Electronic Data Sheet (EDS) files provide details to PLCs and system integrators that describe this organization:

- A structure is put into place to define basic data types.
- Profiles are defined for specific applications. For the SmartMotor, this means that features common to motor control are defined, and specific data objects are assigned to specific object numbers.

PDO and SDO Communication

In CANopen, there are two different modes used for passing data: PDO and SDO. In both forms of communication, data is accessed through the same object dictionary and object-numbering scheme. The same list of objects (position target, velocity actual, status word, control word, etc.) applies to both PDO and SDO communications. However, there are some objects that are deliberately restricted and only accessed through SDO communication. For specific object details, see Object Reference on page 92.



PDO and SDO Communications

SDO

A Service Data Object (SDO) communication is intended for initial setup and occasional access to objects that are seldom needed. Also, some CANopen masters may use SDO communications if they don't intend to configure any PDO communications.

- The SmartMotor provides access to SDO communications in the Pre-Operational *and* Operational NMT states.
- Many PLCs only use access through SDO during a setup phase of operation, and they do so through pre-scripted setup actions.

SDO communications have more overhead per communication due to the following reasons:

- The full object and subindex value are encoded in each SDO communication. This allows easy access to any object, but it limits the amount of payload space available for data in each packet.

- SDO communications also expect a response from the slave back to the master. Both read and write operations confirm by either sending the requested data (read) or confirming that a command was received (write).

SDO communications have the ability to send lengthy amounts of data. For example, string data types are best sent through SDO. In these cases, the data is split up and sent using several CAN bus packets. The recipient of the data will reassemble the CAN bus packets and process the object normally.

PDO

A Process Data Object (PDO) communication allows for minimal overhead when transmitting frequently-used data. Typically, this is used for information that is critical to an ongoing process, which could include the speed, position, control word, etc.

The PDO communication does not specifically encode the object and sub-object information in each packet. This information is agreed on between the master and the slave before entering the Operational state. For further information, see PDO Mapping on page 71.

The following is a list of considerations for using and configuring PDO communication.

- Not all objects are suitable for access through PDO communication. Therefore, many objects are disabled from PDO access.
- Some objects may be overwhelmed if they are only intended to be called intentionally. For example, object 2500h should only be written to occasionally and the response must be examined by the host.
- Data types that are too large to fit in a PDO communication will not work.
- PDO communications do not give a response when received. This makes each transaction more efficient but also does not provide feedback (for example, if a value is out of range).

- To facilitate user programs in the SmartMotor, the arrival of PDOs are indicated by a status bit in Status Word 10, see Object 2304h: Motor Status on page 159. This feature allows user programs to handle the arrival of data as an event. The user program should clear these status bits with a Z(10,bit) command, where bit is values 1–5, after the event handler part of the user program is executed, for example:

```

WHILE 1
  IF B(10,1)==1
    Z(10,1)  ' Clear event flag
    PRINT("Rx PDO 1",#13)
  ENDIF
  IF B(10,2)==1
    Z(10,2)  ' Clear event flag
    PRINT("Rx PDO 2",#13)
  ENDIF
  IF B(10,3)==1
    Z(10,3)  ' Clear event flag
    PRINT("Rx PDO 3",#13)
  ENDIF
  IF B(10,4)==1
    Z(10,4)  ' Clear event flag
    PRINT("Rx PDO 4",#13)
  ENDIF
  IF B(10,5)==1
    Z(10,5)  ' Clear event flag
    PRINT("Rx PDO 5",#13)
  ENDIF
LOOP
END

```

NOTE: Status Word 10, bit 0 cannot be cleared—it is an indication of the master status, see Network Control Commands on page 83. Also, the ZS command will have no effect on these bits.

For more details on the B, Z and ZS commands, see the *SmartMotor Developer's Guide*.

COB-ID Allocation

A Communication Object Identifier (COB-ID) is the unique identifier assigned to a CAN packet. CAN packets do not have a specific destination or source identifier. The sender of a packet, whether a master or slave, will attach an identifier depending on the purpose of the packet. In many cases, the COB-ID is a combination of the node ID and a function code. In other cases, the COB-ID is assigned to a special purpose and does not specifically include a node ID. Many COB-IDs are permanently assigned or reserved.

For example, the SDO communication channel between the master and a particular motor has a COB-ID for master-to-slave packets, and another COB-ID for slave-to-master packets.

- Master-to-motor SDO COB-ID: 1536 (decimal) + node ID
- Motor-to-master SDO COB-ID: 1408 (decimal) + node ID

While it is possible to reassign many COB-IDs, it is not recommended. The "default connection set" is a common way to assign these COB-IDs to a particular function and is adequate (and recommended) for most purposes. Typically, the term "default connection set" is used to

describe a scheme where receive and transmit PDO numbers 1 through 4 are allocated sequentially for the 127 nodes.

NOTE: While recommended, it is not a requirement to follow the default connection set.

The sync packet is an example where the node ID is not relevant to the COB-ID. In other words, it is a COB-ID that *is not* constructed from the node ID of the slave (in contrast with the SDO communications, described above, where the node ID *is* included as part of the COB-ID). The sync packet provides a network pulse that is used by the master and all nodes to coordinate activity. The sync producer simply sends the COB-ID of the sync packet, and its own node ID is not part of the sync's COB-ID.

The only recommended exception to using the default connection set is in the assignment of COB-IDs to PDOs. Note that when configuring PDO communications, there are some choices to make in the assignment of COB-IDs to specific PDOs. There are enough available COB-IDs to assign at least eight to each of 127 nodes. The following are some typical reasons why a network may require a change to the default assignment of COB-IDs to PDOs:

1. If a device needs PDOs other than PDO numbers 1 through 4, then the higher-numbered PDOs must be assigned COB-IDs. For instance, the SmartMotor has a PDO number of 5. However, the default connection set does not provide enough COB-IDs for PDO numbers above 4.
2. By carefully assigning COB-IDs to PDOs, it is possible to have the transmit PDO of one motor be received by other motors. This is accomplished by assigning the same COB-ID to one transmitting motor and one or more receiving motors. This does not follow the default connection set because a COB-ID that would typically be a transmit PDO fills the receiving role in other motors.
3. Lower-numbered COB-IDs have a higher priority in the event of network congestion. It may be important for an application to assign COB-IDs to a particular PDO on a particular node that are lower than those provided by the default connection set.

The following table shows the assigned COB-ID ranges.

COB-ID		
Decimal	Hex	Description
0	0	NMT control
1	1	Reserved
128	80	Sync event
129–255	81–FF	Emergency
256	100	Timestamp
257–384	101–180	Reserved
385–1407	181–57F	Available for assignment to PDO
1409–1535	581–5FF	SDO Transmit (slave to master)
1537–1663	601–67F	SDO Receive (master to slave)
1760	6E0	Reserved
1793–1919	701–77F	NMT error control
2020–2047	780–7FF	Reserved
2047	7FF	(Largest possible COB-ID) Reserved

The following table shows the default connection set for PDO communications based on the CANopen standards.

NOTE: These are recommendations, but they do not need to be strictly followed.

COB-ID		
Decimal	Hex	Description
385–511	181–1FF	Transmit PDO 1 of nodes 1–127
513–639	201–27F	Receive PDO 1 of nodes 1–127
641–767	281–2FF	Transmit PDO 2 of nodes 1–127
769–895	301–37F	Receive PDO 2 of nodes 1–127
897–1023	381–3FF	Transmit PDO 3 of nodes 1–127
1025–1151	401–47F	Receive PDO 3 of nodes 1–127
1153–1279	481–4FF	Transmit PDO 4 of nodes 1–127
1281–1407	501–57F	Receive PDO 4 of nodes 1–127

NMT States

The network management state (NMT) is used to control the general communication functions in the CANopen devices on the network.

The primary states that are used are Pre-Operational and Operational; there are also the Initialization and Stopped states:

- Pre-Operational state allows SDO reads/writes to the motor but prevents PDO communications
- Operational state allows all SDO and PDO communications
- Initialization state starts up the SmartMotor and sets the internal parameters
- Stopped state blocks all commands except the NMT command

The Initialization state is typically not of concern because the motor will automatically transition to the Pre-Operational state. During this transition, the motor will send a startup message. This startup message uses the same COB-ID as a heartbeat message, but it is a one-time event with a data value of 0.

It is also possible to restart the network stack of the motor or to reboot the motor entirely through the NMT control. These are considered initialization states that will return to the Pre-Operational state automatically.

The Stopped state can be used to block commands except the NMT command itself. This means that SDO and PDO access to objects ceases to function. The SYNC, TIME, and EMCY services are also stopped for devices that support these services.

If the heartbeat function of the motor is activated, then the motor will report the current NMT state with each heartbeat message.

NMT Control

NOTE: See associated command when motor is sending NMT commands: NMT (address, command code) on page 85.

The current NMT state is set when the NMT master sends a special packet with a COB-ID of 0. This packet contains two individual bytes of data: the first byte indicates the commanded state that the addressed devices will switch to; the second byte addresses the nodes, either globally or individually.

Byte 1 Value (command code, argument 2 of NMT command)	Byte 1 Command
80h (128 dec)	Go to Pre-Operational state
01h (1 dec)	Go to Operational state
02h (2 dec)	Go to Stopped state
82h (130 dec)	Reset communications (clear objects in the 1xxxh range)
81h (129 dec)	Reset application (resets the SmartMotor)

Byte 2 Value (address, argu- ment 1 of NMT command)	Byte 2 Addressed Devices
0h (0 dec)	All devices on network
01-7Fh (1-127 dec)	Change the state of only the specified SmartMotor

NMT Summary

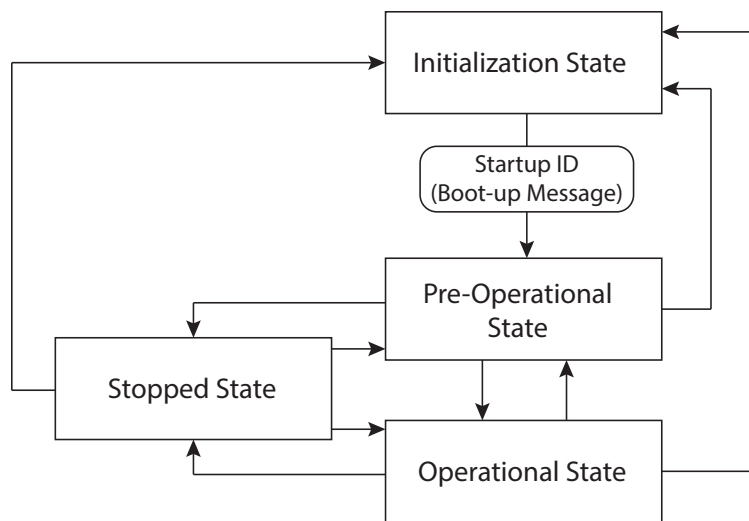
The following table provides a summary of the NMT states. Also, see the NMT State Machine diagram in the next section. The SmartMotor =CAN and RCAN commands can be used to assign/report the value of the NMT state, control word (object 6040h) and status word (object 6041h). For details, see =CAN, RCAN on page 80.

NMT State	NMT Cmnd. code	Reported Value (heartbeat)	SDO funct'l	PDO funct'l	Auto- transition to:	Effect
Initialization (power up)	N/A	N/A	No	No	Pre-Operational	Sends startup message
Initialization (Reset communication)	130	N/A	No	No	Pre-Operational	Clears objects in the 1xxxh range Sends startup message

NMT State	NMT Cmd. code	Reported Value (heartbeat)	SDO funct'l	PDO funct'l	Auto-transition to:	Effect
Initialization (Reset Application)	129	N/A	No	No	Pre-Operational	Reboots the SmartMotor Sends startup message
Pre-Operational	128	127	Yes	No	-	
Operational	1	5	Yes	Yes	-	
Stopped	2	4	No	No	-	

NMT State Machine Diagram

The following diagram shows the relationship and interaction between the possible NMT states.



NMT State Machine

For more details on CANopen network management, see the CAN in Automation (CiA) website at:

<http://www.can-cia.org/index.php?id=155>

PDO Communications

There are two methods of PDO communications: peer-to-peer (versus master-to-slave), and synchronous (versus asynchronous). These communication methods are described in the following sections. Note that these communications methods are not mutually exclusive. For example, peer-to-peer means that motor 1 and send a PDO and motor 2 can receive that same PDO. This can be done through either of the following methods:

- Synchronous: Motor 1 transmits when a sync packet is seen
- Asynchronous: Motor 1 transmits based on its own internal timer

Peer-to-Peer Communications

An advantage to the peer-to-peer method of PDO communication is that any node can be a recipient of any PDO. This allows for data to flow peer-to-peer rather than always going to the master. It also allows for broadcasting to multiple nodes (for example, there may be an I/O input device on the CANopen network that all devices wish to monitor for a button press).

The CANopen master must configure this peer-to-peer relationship. However, once it is configured and the network is in the Operational state, the process will continue without constant intervention from the master.

To establish a peer-to-peer relationship, one node will transmit a data object using a particular COB-ID. Any device that wishes to receive this information should allocate this COB-ID to a receive PDO and map that PDO to the desired object to accept the data. For details about how PDOs are mapped, see PDO Mapping on page 71 and COB-ID Allocation on page 24.

Synchronous Communications

PDOs may be configured to transmit from a node's own internal timer, or they may be transmitted based on the sync event on the network. The sync event is simply a special CAN frame produced by the node or master that is assigned as the sync producer. PDO Mapping on page 71 describes the details for configuring these two modes of PDO transmission.

When the sync method is chosen, it is possible to transmit on every sync message, or to subdivide the transmission rate by up to 240. In other words, transmission can be set to occur on every sync, every other sync, every third sync, and so on... up to every 240th sync.

Supported Features

This chapter provides information on the supported and unsupported features of the CANopen specification.

Supported	31
Motion Modes	31
NMT State Machine Master	31
PDO Transmit on Event	31
PDO Transmit on Timer Only	32
PDO Transmit on Sync	32
Dynamic PDO Mapping	32
Heartbeat Producer	32
Sync Producer	32
Not Supported	33
Emergency Messages	33
Saving Parameters	33
Heartbeat Consumer	33
MPDO Communications	33
CAN Bus Bit Rate	33
PDO Transmit on RTR (Remote frames)	33
Node Guarding	33
TIME Service	33
Sync Start	33

Supported

This section describes the CANopen features that are supported by the SmartMotor.

Motion Modes

The following motion modes are supported:

- Profile Position (PP, mode of operation: 1) — behaves like the SmartMotor MP mode; supports "single setpoint" and "set of setpoints" modes
- Profile Velocity (PV, mode of operation: 3) — behaves like the SmartMotor MV mode
- Interpolation (IP, mode of operation: 7) — behaves like the SmartMotor MD mode
- Torque (TQ, mode of operation: 4) — behaves like the SmartMotor MT mode
- Homing (HM mode, mode of operation: 6) — only methods 1, 2, 17, 18, 33, 34 and 35 are supported—all others are not supported; homing offset, homing speeds and homing acceleration are supported
- Follow with Ratio (electronic gearing) & Cam (electronic camming): Allows one or more SmartMotors to receive data from an encoder on the CANopen bus, and then rotate at a specific ratio relative to the input encoder. Includes objects to support gearing over CANopen, such as MFMUL, MFDIV, MFA and MFD, and to select follow or cam modes of operation. Related objects are: 2207h, 2208h, 2209h, and 220Ah-220Dh.

The Supported Drive Modes object (6502h) is used to report the modes of operation that are available. The Modes of Operation object (6060h) is used to request the desired mode of operation before setting the Control Word object (6040h).

NMT State Machine Master

Required for mastering I/O expansion across CANopen. This expanded I/O capability allows the SmartMotor to interface with standard CiA 301 CANopen devices and function as the I/O device CAN bus master (i.e., no external bus master needed). See the overview of this capability in I/O Device CAN Bus Master on page 11.

It includes capability to support PDO operation as follows:

- NMT control
- 8, 16, and 32-bit data objects that can be mapped to PDOs
- Status word indication of Rx PDO data arrival

Related objects are: 2220h, 2221h and 2204h. For details, see Object Reference on page 92.

Related commands are: NMT, SDORD, SDOWR, CANCTL, and B/RB. For details, see the descriptions in this guide and in the *SmartMotor Developer's Guide*.

Example user programs are shown in the *SmartMotor Developer's Guide*, Part 3: Examples.

PDO Transmit on Event

Process Data Objects (PDOs) can be configured to transmit on a change of value within the motor (Transmission type: 255). Transmission type 255 also transmits on the transmit timer

event configured in the PDO's corresponding communications parameter object. The transmit timer provides a minimum rate at which the data is transmitted.

- The transmission type is set using subindex 2 of objects 1800h, 1801h, 1802h, 1803h and 1804h.
- The transmission timer is set using subindex 5 of objects 1800h, 1801h, 1802h, 1803h and 1804h.

PDO Transmit on Timer Only

Transmit PDOs can be configured to transmit on a timer using a transmission type setting of 254.

- The transmission type is set using subindex 2 of objects 1800h, 1801h, 1802h, 1803h and 1804h.
- The transmission timer is set using subindex 5 of objects 1800h, 1801h, 1802h, 1803h and 1804h.

PDO Transmit on Sync

Transmit PDOs can be configured to transmit in response to a sync packet. Transmit types 1-240 in the transmission type setting are used to configure this. The value of the transmission type controls how often the transmit PDO is sent in response to a sync (e.g., transmit type = 1 is sent in every sync packet; transmit type = 240 is sent in every 240th sync packet).

The transmission type is set using subindex 2 of objects 1800h, 1801h, 1802h, 1803 and 1804h.

Dynamic PDO Mapping

There are objects used to simultaneously configure (map) up to five Receive PDOs and five Transmit PDOs. These mappings are dynamic — any object with "PDO mappable" in its description can be mapped to a PDO through the standard CANopen mapping procedure.

Dynamic mapping of objects to PDO is configured using objects 1600h, 1601h, 1602h, 1603h, 1604h, 1A00h, 1A01h, 1A02, 1A03h and 1A04h. For details, see PDO Mapping on page 71.

Heartbeat Producer

The motor can be configured to transmit a heartbeat at a configurable rate. For details, see Object 1017h: Producer Heartbeat Time on page 109.

Sync Producer

The SmartMotor can produce sync messages. This requires setting the Communication Cycle Period object (1006h) and the COB-ID SYNC object (1005h). There is a specific order to configuring these objects, and object 1005h requires an additional bit setting. Therefore, it is important to review the descriptions of both objects. For details, see Object 1005h: COB-ID SYNC on page 101 and Object 1006h: Communication Cycle Period on page 103.

Not Supported

This section describes the CANopen features that are not supported by the SmartMotor.

Emergency Messages

Emergency (EMCY) object messages are not produced or consumed by the SmartMotor. The associated objects, 1014h and 1015h, do not exist.

Saving Parameters

The SmartMotor does not support parameter data saving. Objects 1010h and 1011h are not implemented.

Heartbeat Consumer

The SmartMotor does not consume heartbeat messages. Therefore, it will not take action on the presence or absence of any heartbeat messages. However, the SmartMotor can be a heartbeat producer. For details, see Object 1017h: Producer Heartbeat Time on page 109.

MPDO Communications

The SmartMotor does not support the multiplexed-PDO (MPDO) method of communication. Ordinary transmit and receive PDOs are supported.

CAN Bus Bit Rate

The CAN bus bit rate of 10000 bits/sec is not supported.

PDO Transmit on RTR (Remote frames)

PDO Transmit types 252 and 253 are not supported. Remote (RTR) frames are not supported.

Node Guarding

Node Guarding is not supported.

TIME Service

TIME service is not supported.

Sync Start

Sync Start value is not present or supported. This refers specifically to subindex 6 of the Transmit PDO Communication Parameter objects 1800h–1804h.

Connections, Wiring and Status LEDs

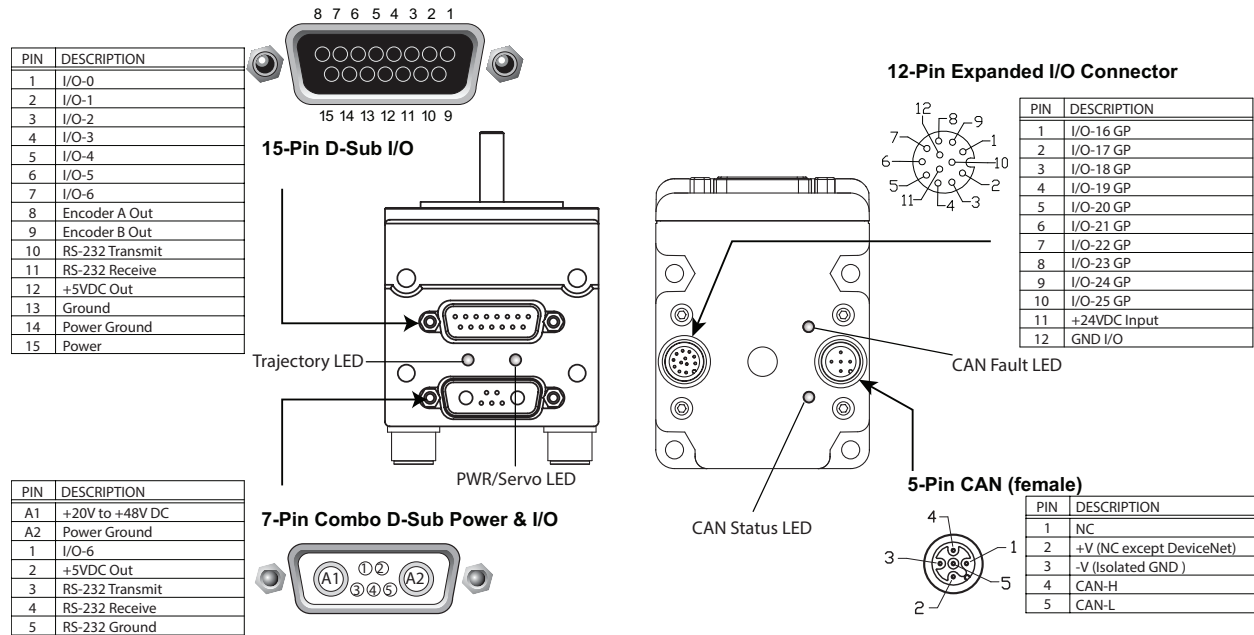
This chapter provides information on the SmartMotor connectors, a multidrop cable diagram, and a description of the SmartMotor status LEDs.

Connectors and Pinouts	35
D-Style Motor Connectors and Pinouts	35
D-Style Motors: CDS Option Schematic	35
CDS on the DA-15 Connector	35
CDS on the 7W2 Connector (CDS7)	36
M-Style Motor Connectors and Pinouts	39
Cable Diagram	39
CAN Multidrop Cable Diagram	40
Bus Termination	40
Maximum Bus Length	41
Status LEDs	42
Other Communications with the Motor	43

Connectors and Pinouts

D-Style Motor Connectors and Pinouts

The following figure provides a brief overview of the connectors and pinouts available on the D-style SmartMotors. For details, see the *Class 5 SmartMotor™ Installation and Startup Guide*.



NOTE: The DE power option is recommended. For details, see the *Class 5 SmartMotor™ Installation and Startup Guide*.

D-Style Motors: CDS Option Schematic

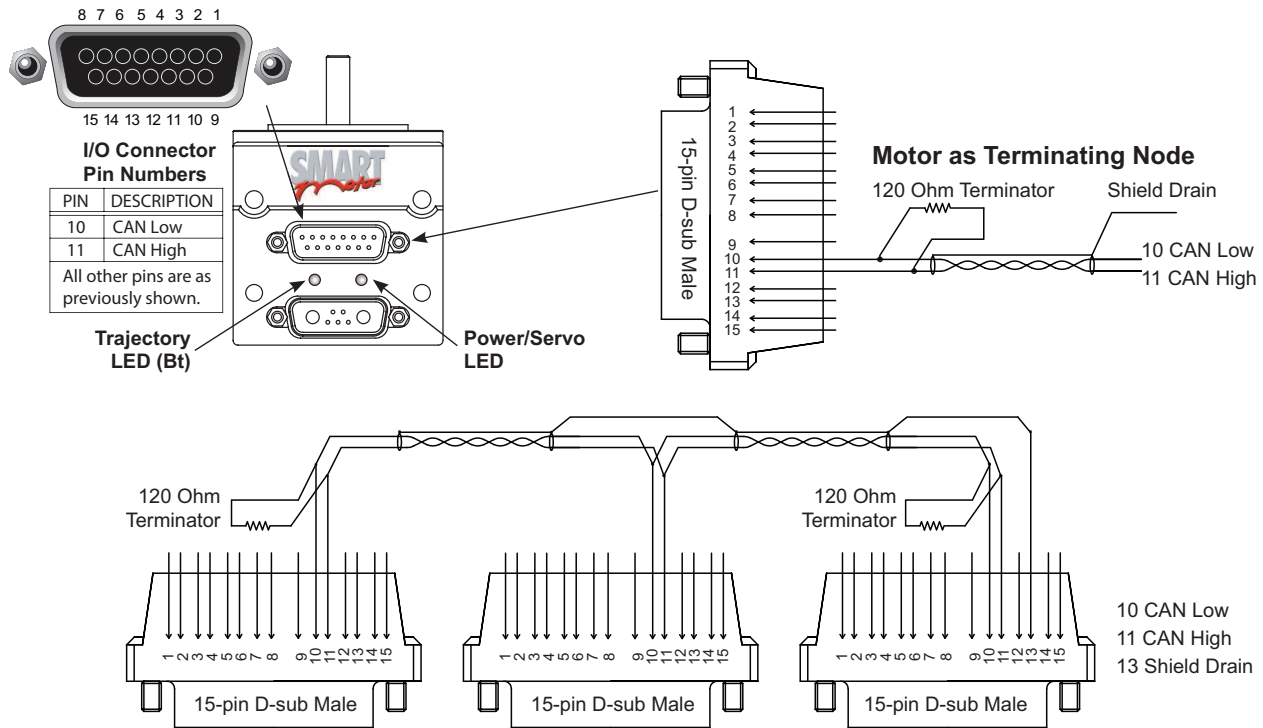
This section provides schematics for wiring a D-styleSmartMotor equipped with the CDS CAN connection option. This option can be used for Combitronic communications. For details on Combitronic communications, see Combitronic Communications in the *SmartMotor™ Developer's Guide*. The option also allows the integrated brake option to be used with CAN, which is useful for many vertical-axis applications.

NOTE: The CDS Option is available only on specially equipped D-styleSmartMotors. Contact Moog Animatics for details.

CDS on the DA-15 Connector

A special version of the D-styleSmartMotor with the CDS CAN connector option allows CAN bus network wiring through the DA-15 connector (15-pin D-sub I/O connector shown in the following figure). This is an advantage when the M-style CAN connector is not desired. The D-style motor with the CDS CAN connection option can be used as the terminating node. To enable this, a 120 ohm terminating resistor (shunt) must be placed across pins 10 and 11. For details, see the following figure.

NOTE: Terminating resistors (shunts) must always be used at both ends of a CAN bus network.



NOTES: A terminating resistor (shunt) is required at each end of the bus.
 Bus must be multi-drop as shown, *not* a star network.
 24V CAN bus power connection is *not* required at the motor.

Schematic for CDS Option, D-Style SmartMotor Used as Terminating Node

CDS on the 7W2 Connector (CDS7)

Alternatively, the CDS7 option on the D-style SmartMotor now allows CDS to be wired through pins 1 and 2 of the 7-pin D-Sub (7W2) connector as shown in the following figure. The wiring is accomplished through the use of a single Add-A-Motor cable (PN: CBLSMCDS) that carries Power, RS-232, and CAN bus to the next motor in the chain. This method allows pins 10 and 11 on the 15-pin D-sub connector to be used as an easy terminating point on the last motor by simply placing a 120 Ohm terminating resistor (shunt) across those pins OR installing the Pass-Thru Terminator (PN: CBLSM-TR120) on that connector.

This design not only greatly simplifies motor installation, but also does the same for CAN bus addressing. With CDS7, the SmartMotor is the first and only single-cable, point-to-point integrated motor to have both motor-to-motor full communications and control capability, and only single-cable, point-to-point for both power and communications in general.

As a result:

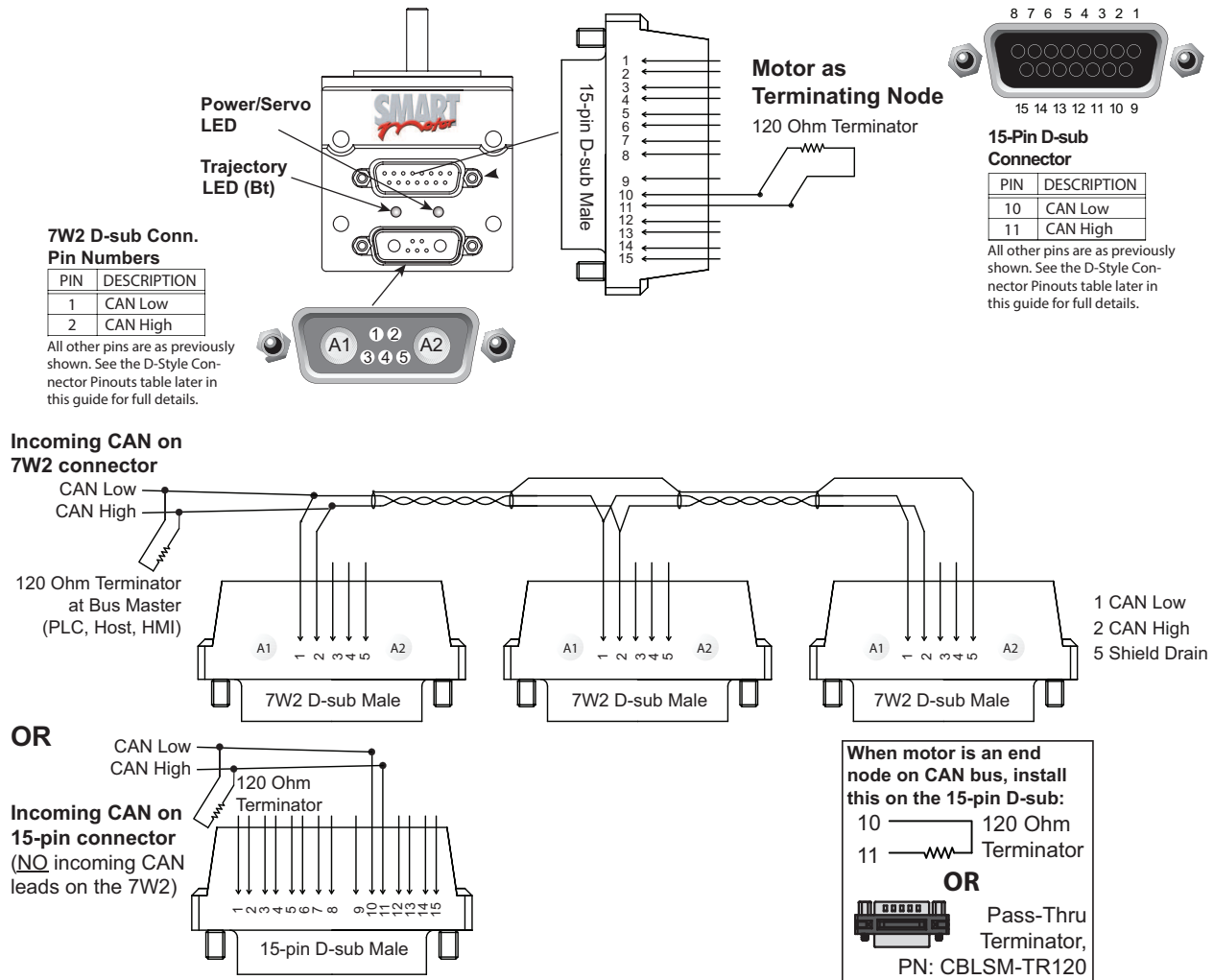
- When you order the D-series CDS7 SmartMotors and new Add-A-Motor cables, you get prewired, instant, full-network capability, and those networked motors are auto-detectable directly from SMI.
- This allows for auto-addressing and ease of reconnection at the next power up.

- Unlike competitive CANopen devices, the CDS7 SmartMotors do not require single-node power up to set addresses. You can power up all at once, have SMI detect them and set all motor CAN bus addresses in a single line of code. For details, see Detecting and Addressing the SmartMotors in the *Class 5 SmartMotor Installation and Startup Guide*.

CDS7 is fully backward compatible with the wiring method described in the previous section. Therefore, all -CDS7 option SmartMotors can be wired either through the D-sub connector or the 7W2 connector, and there is no change in motor part numbers for the 7W2 method. The only part change is the cabling used to connect the motors - refer to the figures, descriptions and part numbers later in this section.

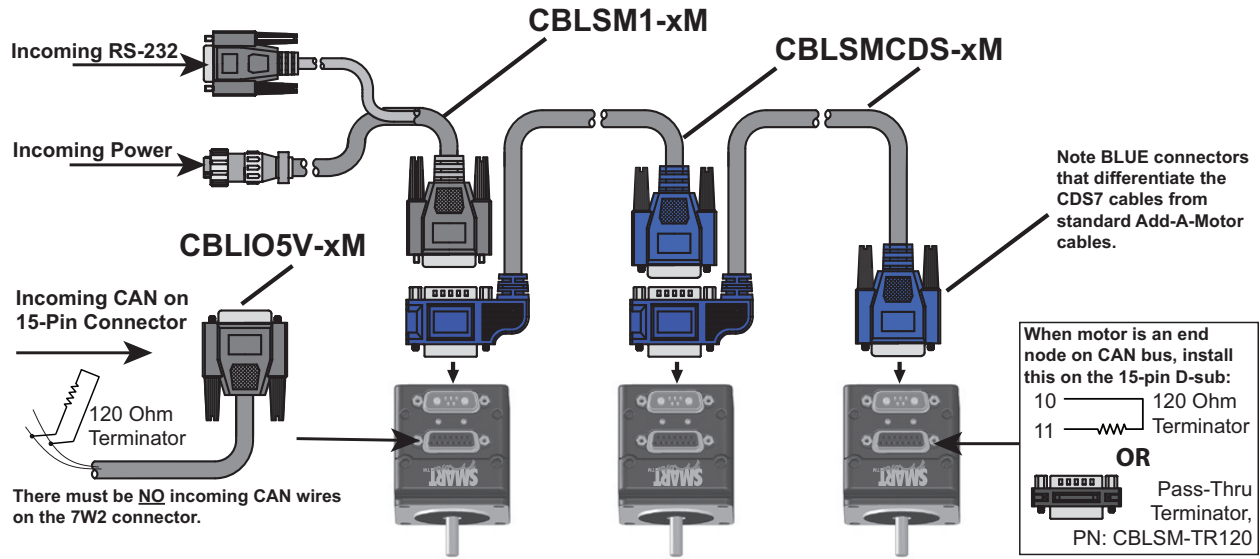
When the SmartMotor is a terminating node on the CAN bus, the 120 Ohm terminating resistor (shunt) must be wired to pins 10 and 11 of the 15-pin D-sub connector. A 120 Ohm terminating resistor (shunt) is also required at the beginning of the CAN bus. See the following figures.

For further convenience, a 15-pin pass-through D-sub connector (Pass-Thru Terminator, PN: CBLSM-TR120) is available that contains a built-in 120 Ohm terminating resistor (shunt). When a SmartMotor is the terminating node and the 7W2 wiring method is used (described previously), you can simply install the pass-through connector on the SmartMotor's 15-pin D-sub connector to serve as the terminator.



For the above methods, the incoming CAN bus signal is connected to the 7W2 connector OR the 15-pin connector, and the Class 5 CDS Add-A-Motor cable (PN: CBLSMCDS-xM) will automatically provide CAN bus, RS-232 and power wiring in one cable between motors.

The following figure shows how factory cables are used to attach the incoming CAN bus signal to the 15-pin connector, and incoming power and RS-232 communications to the 7W2 connector.



NOTES: A terminating resistor or pass-thru terminator is required at each end of the CAN bus. Bus must be multi-drop as shown, *not* a star network. 24V CAN bus power connection is *not* required at the motor.

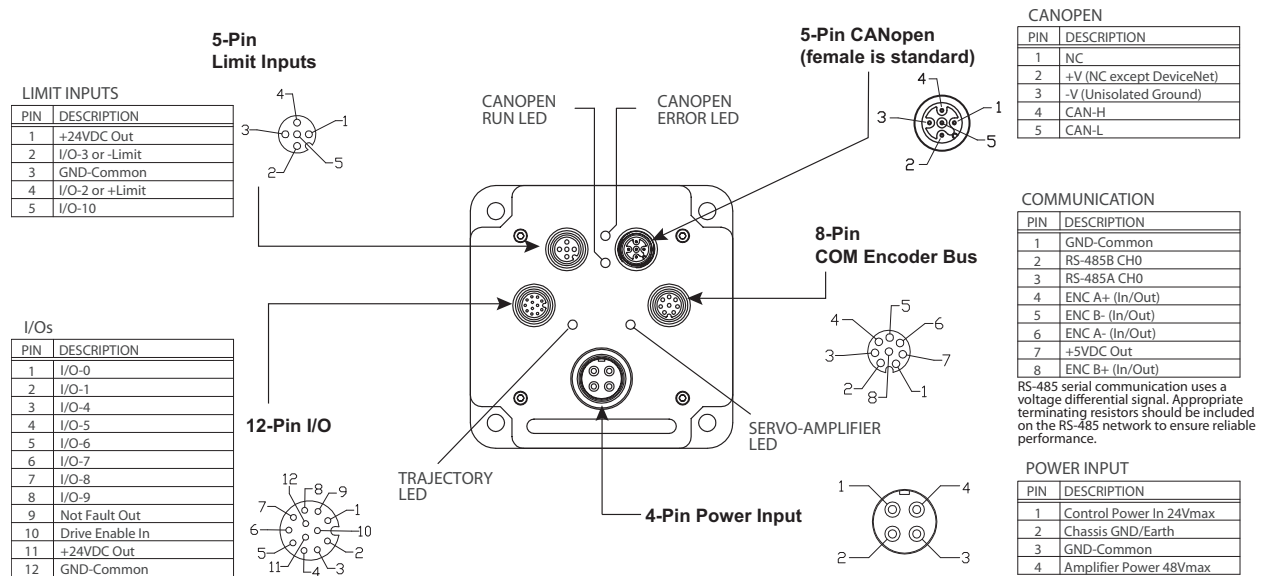
Refer to the following tables for part numbers and cable lengths.

CDS7 Cable PN	Length (Meters)	Power/Com Cable PN	Length (Meters)
CBLSMCDS-0.3M	0.3	CBLSM1-xM	3-10
CBLSMCDS-0.9M	0.9	CBLIO5V-xM	3-10
CBLSMCDS-3.0M	3.0		
CBLSMCDS-7.5M	7.5		
CBLSM-TR120	Pass-thru terminator		
CAUTION: As noted by the BLUE connectors, these are NOT the same as the standard Add-A-Motor cables and they are NOT interchangeable.			

See the cable datasheets on the website or consult the factory for the schematic diagrams.

M-Style Motor Connectors and Pinouts

The following figure provides a brief overview of the connectors and pinouts available on the M-style SmartMotors. For details, see the *Class 5 SmartMotor™ Installation and Startup Guide*.



Cable Diagram

CAN bus wiring is most reliable when a straight bus is used (see the following figure).

Common problems with CAN bus wiring are often traced to branches or star configurations. These configurations often create multipath signal reflections that cause communication errors.



CAUTION: If a branch is absolutely necessary due to wiring constraints, it is the responsibility of the system designer to test and prove the layout is not causing communication errors. Moog Animatics cannot ensure the success of branched layouts.

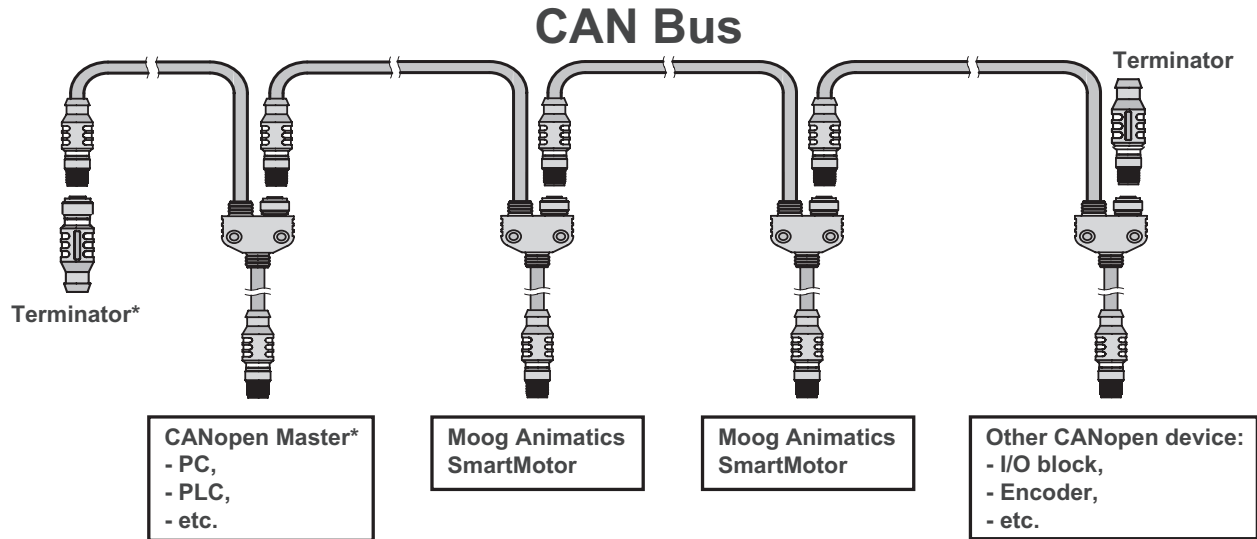
The following figure shows a straight network with no branches. The short drop to each motor is acceptable. These drops from the Y connector to the motor should be 0.3 meters or less.



CAUTION: If drops from the Y connector to the motor need to exceed 0.3 meters, it is up to the system designer to test and prove the additional drop length is not causing communication errors. Moog Animatics cannot ensure the success of longer drops.

The wire length between any two motors should be at least 0.1 meter, including the drop length. For details, see Maximum Bus Length on page 41.

CAN Multidrop Cable Diagram



*Master may have termination option; see master's documentation for details.

Bus Termination

Proper termination is critical for successful network communications. There must be two terminators (120 Ohms each), and they must be located at the two ends of the network. Because the network is a straight line, there are exactly two ends of the network to place the terminators.



CAUTION: Using less than two terminators is not acceptable; using more than two terminators is not acceptable.

In the event that the master device specifically provides a terminating resistor, then that may be used instead of the terminator plug. However, the master must be at the end of the network in that case; it cannot be in the middle.

Maximum Bus Length

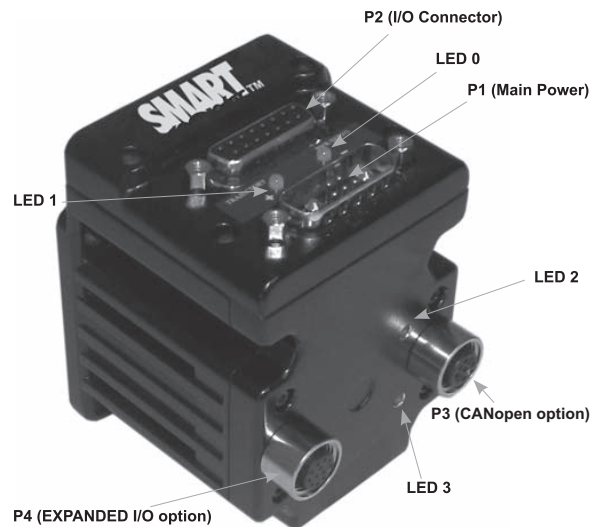
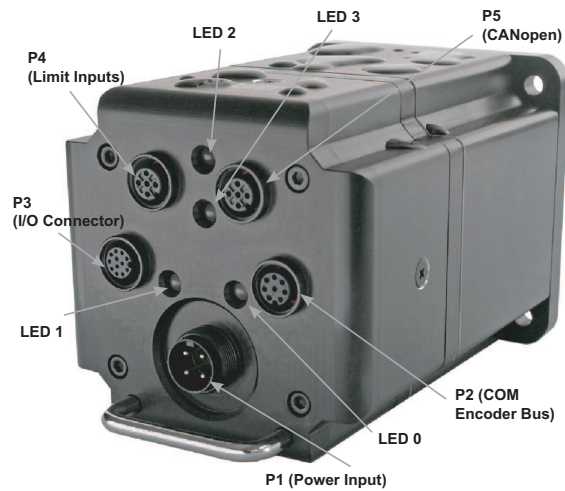
The following table shows the transmission bit rates and corresponding maximum bus lengths. The bus length is the calculated maximum distance of the straight bus from one terminated end to the other terminated end.

Bit rate (bits/second)	Bus length (meters)
1000000	25
800000	50
500000	100
250000	250
125000	500
50000	1000
20000	2500

NOTE: Bus lengths exceeding 200 meters may have additional requirements such as the use of repeaters or optocouplers. For details, see the CiA 301 specifications.

Status LEDs

The Status LEDs provide the same functionality for the D-style and M-style (including IP-sealed) SmartMotors.



LED 0: Drive Status Indicator

Off	No power
Solid green	Drive on
Flashing green	Drive off
Flashing red	Watchdog fault
Solid red	Major fault
Alt. red/green	In boot load; needs firmware

LED 1: Trajectory Status Indicator

Off	Not busy
Solid green	Drive on, trajectory in progress

LED 2: CAN Bus Network Fault (Red LED)

Off	No error
Single Flash	At least one error exceeded limit
Double Flash	Heartbeat or guard error
Solid	Busy off state

LED 3: CAN Bus Network Status (Green LED)

Blinking	Pre-operational state (during boot-up)
Solid	Normal operation
Single	Device is in stopped state

LED Status on Power-up:

- With no program and the travel limit inputs are low:
 - LED 0 will be solid red indicating the motor is in a fault state due to travel limit fault.
 - LED 1 will be off.
- With no program and the travel limit inputs are high:
 - LED 0 will be solid red for 500 milliseconds and then begin flashing green.
 - LED 1 will be off.
- With a program that disables only travel limits and nothing else:
 - LED 0 will be solid red for 500 milliseconds and then begin flashing green.
 - LED 1 will be off.

NOTE: D-style motors with the CDS CAN connector option use LED 1 to indicate a CAN error. Because this LED also indicates the trajectory status, it will alternate red/green colors if a CAN error occurs while a trajectory is in progress.

D-Style Motor CDS Option LED 1 CAN Error Indication:

Condition	Indication
Bt = 0, CAN bus OK	Trajectory LED = Off
Bt = 1, CAN bus OK	Trajectory LED = Green
Bt = 0, CAN bus fault	Trajectory LED = Flashing red
Bt = 1, CAN bus fault	Trajectory LED = Alternating red/green

Bt refers to Busy Trajectory status bit. When the motor is actively pursuing a trajectory, that bit will be set to 1.

Other Communications with the Motor

In addition to communicating with the SmartMotor as a CANopen device, you can also communicate with it directly from a PC or laptop. This is useful if you need a "back door" into the motor, for example, to modify the stored user program or download a new one, or for troubleshooting purposes.

For information on connecting the SmartMotor directly to a PC, see the Getting Started chapter in the *Class 5 SmartMotor™ Installation and Startup Guide*.

Manufacturer-Specific Objects

This chapter provides details on manufacturer-specific objects.

I/O	45
User Variables	45
Calling Subroutines	47
Command Interface (Object 2500h)	48
Command Interface	48
Program Upload/Download	49
Upload from Motor	49
Download to Motor	49

I/O

The CiA 402 motion profile provides limited access to the onboard I/O of the SmartMotor. However, there are other manufacturer-specific objects that provide more I/O control.

As part of the CiA 402 motion profile, objects 60FDh and 60FEh are provided. For details, see Object 60FDh: Digital Inputs on page 224 and Object 60FEh: Digital Outputs on page 226.

For the D-style motor, object 2100h is highly specific to the multiplexed role of the seven I/O pins. This function is not supported on the M-style motor. For more details, see Object 2100h: Port Configuration on page 136.

For general access to individual I/O pins, the Bit I/O object (2101h) offers a more specific way to send commands. This feature works on the M-style and D-style motors. It can be used to disable the limit inputs if desired. For more details, see Object 2101h: Bit IO on page 137.

NOTE: The limit-switch inputs for all SmartMotors must be satisfied before motion is allowed. The inputs must either be physically wired or disabled if not connected. Additionally, M-style motors require the drive-enable input to be true (high) for motion to start.

User Variables

The SmartMotor has an array of user variables that are accessible to user programs and are visible as CANopen objects. This provides a common area where information can be shared between a user program and the CANopen network.

The variables use predefined names: a–z, aa–zz and aaa–zzz, which comprise a total of 78 variables; these are 32-bit signed integers.

Additionally, there is a 204-byte array. It can be accessed as 8, 16 or 32-bit signed values. For more details, see the *SmartMotor™ Developer's Guide*.

There are 12 variables that are available as "mappable" variables. This feature allows a CANopen SmartMotor in slave or master mode to accept PDO mappings to data of size 8, 16, or 32 bits:

- Mappable Variables object (2220h) offers access to 8-bit user variables ab[0], ab[1], ab[2] and ab[3]. For more details, see Object 2220h: 8-Bit Mappable Variables on page 153.
- Mappable Variables object (2221h) offers access to 16-bit user variables aw[32], aw[33], aw[34] and aw[35]. For more details, see Object 2221h: 16-Bit Mappable Variables on page 154.
- Mappable Variables object (2204h) offers access to 32-bit user variables aaa, bbb, ccc and ddd. For more details, see Object 2204h: Mappable 32-bit Variables on page 142.

These mappable variables are available for applications such as general-purpose I/O blocks using PDO communications. Also, note that the "master" does not always need to be the SmartMotor receiving all data.

A wider range of user variables is accessible through the User Variable object (2201h). However, this mechanism does not allow PDO communications — object 2201 is only available through SDO communications. Therefore, it is typically used to pass constants or other configuration data at startup, when a PLC may pass SDO data. During the Operational state, a master may continue to pass data to variables through object 2201h if it is capable of SDO

communication at that time. For more details, see the Object 2201h: User Variable on page 139.

A typical use of user variables in combination with CANopen is to receive information from another motor or sensor device on the network. For example, variable aaa could be mapped to a receive PDO (RxPDO). If that PDO is allocated a COB-ID of a sensor on the network, then that information can be used in a SmartMotor user program.

Another common use of the mapping variables is to report information that does not have a CANopen object. For instance, a user may want to perform a calculation in a user program and report the result back to the master. In this case, the user program would set a variable such as bbb=<expr>. The variable bbb should be mapped to a transmit PDO (TxPDO). Then the master or other nodes on the network can access that information.

It is possible to use the SmartMotor as a bridge by combining the two techniques: receiving data into a user variable and transmitting information from a user variable. This allows interfacing of two devices that need intermediate computation. For example, a temperature sensor could feed into the SmartMotor, and a process control loop in a SmartMotor program could use that information to control a cooling fan through an I/O device. This may be advantageous if there are applications that are easier to program in the SmartMotor instead of the CANopen master.

Often, the mapping variable is used to send or receive a field of bits. When receiving, the bitwise program operators can be used: | (or), & (and), !| (xor). For example, the following IF expression will be true when bit 3 is set:

```
IF (ddd&8) !=0      'Will be true when ddd bit 3 is true.
... do action
ENDIF
```

When transmitting, the following are some simple techniques for setting bitwise values:

```
aaa=aaa|8           'Set bit 3.
aaa=aaa|bbb         'Logical OR all bits from aaa and bbb; save to aaa.
aaa=aaa!|64         'Toggle bit 6 (XOR).
aaa=aaa&-9          'Clear bit 3 and leave other bits alone.
aaa=aaa&(-3&-9)    'Clear bit 1 and 3 at the same time.
aaa=aaa|(2|8)       'Set bit 1 and 3 at the same time.
```

The following table lists the bit numbers and the corresponding decimal values used to set with OR (for 16 bits, only) or clear with AND (for 16 bits, only).

Bit number (0–15)	Decimal value to set bit with OR (for 16 bits, only)	Decimal value to clear bit with AND (for 16 bits, only)
0	1	–2
1	2	–3
2	4	–5
3	8	–9
4	16	–17
5	32	–33
6	64	–65
7	128	–129
8	256	–257
9	512	–513
10	1024	–1025
11	2048	–2049
12	4096	–4097
13	8192	–8193
14	16384	–16385
15	32768	–32769

Calling Subroutines

The functionality of the SmartMotor can be extended by creating and loading a user program into the motor. There are two ways to control the running of this program: a GOSUB call, or a RUN command to run the entire program from the top of the program.

NOTE: A user program will always automatically run from the start when the motor is powered on or reset unless the RUN? command is included at the top of the user program. The RUN command is not the same as the RUN? command. For details on these commands, see the *SmartMotor™ Developer's Guide*.

The GOSUB R2 object (2309h) provides access to the GOSUB, RUN and END commands. It is PDO mappable, and it only reacts to a change of value. For details, see Object 2309h: GOSUB R2 on page 172. This object replaces the functionality of objects 2305h and 2306h.

Bit 8 of the Status Word object (6041h) can be used to determine when the subroutine called with object 2309h has finished. When the bit clears, the subroutine has completed.

Calls to subroutines using object 2309h are automatically blocked if a previous call made through object 2309h is still busy. When that subroutine returns, bit 8 of the Status Word object (6041h) will clear.

NOTE: Unlike GOSUB, there is no CANopen access to the GOTO function.

Command Interface (Object 2500h)

The SmartMotor has many commands that are not mapped to CANopen objects. Many of these commands are obscure or take a complex set of arguments. A mechanism is provided to access these commands by sending a command string to object 2500h.

This section provides details on the object 2500h command interface and use in program upload/download.

Command Interface

This section describes the command interface for the Encapsulated Animatics Command object (2500h). This object provides an interface to the SmartMotor command language. Please note the following:

- The status information must read back from subindex 3 of object 2500h.
- This object is not accessible through PDO.

The following table describes the elements of object 2500h.

Object	Subindex	Description
2500h	0	Number of entries (3).
2500h	1	Command string to motor "VISIBLE STRING" type.
2500h	2	Response from motor "VISIBLE STRING" type.
2500h	3	Status from motor "UNSIGNED 8" type.

The status bits in subindex 3 of object 2500h are:

Bit	Description
0	Command in progress.
1	Command complete/response ready.
2	Overflow.
3-7	Reserved.

The following procedure describes the steps to send a command:

1. Check that the "command in progress" = 0.
2. Write the command to subindex 1 of object 2500h; terminate the command with a null value.
3. Read the status from subindex 3 of object 2500h; check the status of the "command complete" bit.
4. Repeat the previous step if the "command complete" bit is 0.
5. When the "command complete" bit is 1, the command has completed. If it was a report command, there will be a string response to read in subindex 2 of object 2500h; if it was a non-report command, there will be no response. The values are ASCII-encoded decimal format.

Program Upload/Download

The Encapsulated Animatics Command object (2500h) behaves like a string command. Therefore, it can support the upload and download of user programs. The following sections describe the upload and download procedures.

Upload from Motor

The following steps are used to upload a user program from the SmartMotor to the host:

1. The host writes to the motor's subindex 1 of object 2500h with the UPLOAD (or UP) command. Strings need to be null-terminated like most commands.
2. The host checks the "Response ready" and "Command in progress" flags in subindex 3 of object 2500h.
3. When "Response ready" = 1, the host will read a data block of 0–31 bytes plus the null terminator from subindex 2 of object 2500h.
4. The previous step is repeated until the "Command in progress" flag is 0 and the "Response ready" flag is 0. That indicates the process has completed.

NOTE: On the final cycle of the upload, the motor will always set the "Response ready" flag before clearing the "Command in progress" flag. This ensures that the host has a reliable indicator when the final cycle has occurred and will not wait forever. In other words, the host should stop looking for a response as soon as both of those flags are clear.

Download to Motor

The following steps are used to download a user program from the host to the SmartMotor:

1. The host writes to motor's subindex 1 of object 2500h with the LOAD command. Strings need to be null-terminated like most commands.
2. The host waits for the "Command in progress" flag in subindex 3 of object 2500h to return to 0.
3. The host writes the program data to subindex 1 of object 2500h, first 32 bytes, with *no* null terminator. This can include a header and anything after the header. The CAN command manager will consume the header and whatever follows it.
4. The host waits for the "Command in progress" flag in subindex 3 of object 2500h to return to 0. This serves as the ACK (acknowledgment) signal. There is no reading of subindex 2 of object 2500h.

NOTE: Do not attempt to read subindex 2 of object 2500h because that buffer is used for other purposes during this procedure.

5. The host writes more program data to subindex 1 of object 2500h, 32 bytes at a time, with *no* null terminator. Handshaking continues through the "Command in progress" flag. Transmission may be ended at any time by sending 0xFF 0xFF 0x20 in the character stream.

NOTE: This sequence does not need to fall in the same buffer segment. There is no need to pad the buffer.

CiA 402 Drive and Motion Control Profile

The CiA 402 Drive and Motion Control Profile supports the motion control of the SmartMotor. The associated objects comprise a large portion of the object dictionary (see Drive and Motion Control Profile on page 179). This profile is supported by many vendors of industrial controls.

CiA 402 Profile Motion State Machine	51
Control Words, Status Words and the Drive State Machine	51
Status Word (Object 6041h)	52
Control Word (Object 6040h)	53
Motion Profiles	54
Position Mode	54
Absolute Position Mode Summary	55
Absolute Position Mode Example	55
Relative Position Example	57
Velocity Mode	58
Velocity Mode Summary	59
Velocity Mode Example	59
Torque Mode	60
Torque Mode Summary	61
Torque Mode Example	61
Interpolated Position Mode	62
Interpolated Position Mode Summary	63
Example: Short Run on a Single Motor	64
Example: Continuous Run on a Single Motor	65
Example: Resuming Motion in IP Mode	66
Synchronization	66
User Bits	67
Splining	68
Variable-Length Segments	68
Homing Mode	68
Homing Summary	69
Homing Example	69

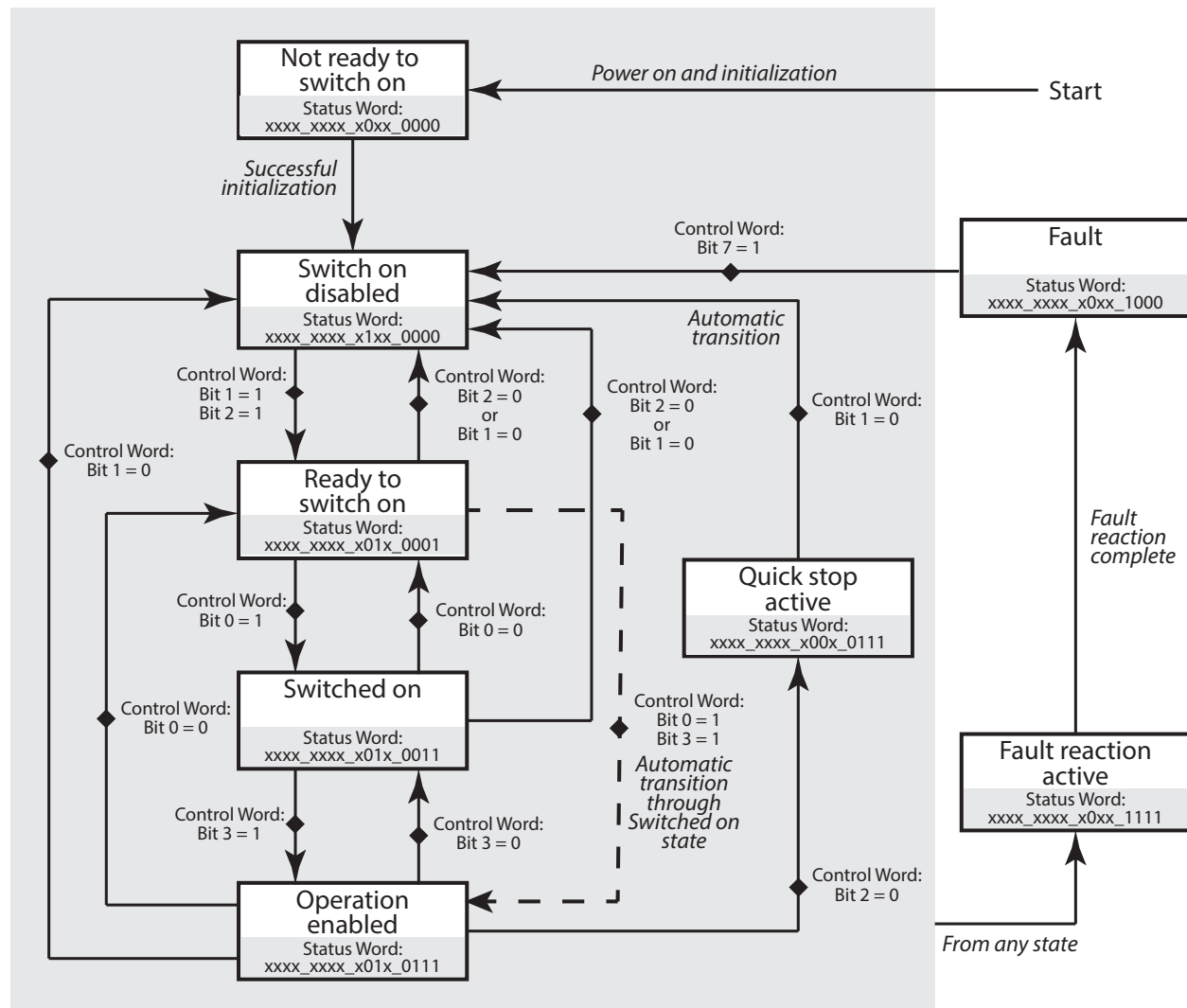
CiA 402 Profile Motion State Machine

Support for the CiA 402 motion profile (DS402) in the SmartMotor includes the Control Word object (6040h) and the Status Word object (6041h). Under all types of motion, the control word starts or stops the drive and the status word reports the state of the drive.

However, the type of motion profile is not controlled with these objects — it is commanded through the Modes of Operation object (6060h) and reported from the Modes of Operation Display object (6061h). For more details, see the examples in Motion Profiles on page 54.

Control Words, Status Words and the Drive State Machine

Refer to the following diagram of the DS402 Drive State Machine. The power drive system finite state automaton (PDS FSA) is described in the DS402 specification. This is the mechanism used to command the motor to begin a new move or turn the drive on/off. The DS402 specification describes several operation states controlled by the Control Word object (6040h) and read back using the Status Word object (6041h).



DS402 Drive State Machine

Status Word (Object 6041h)

The Status Word object (6041h) reports the PDS FSA state machine per the DS402 specification. The following distinct states are defined, where "x" is a bit that could be either a 1 or a 0:

Status Word 6041h (16 bits)	PDS FSA state	Meaning
xxxx xxxx x0xx 0000	Not ready to switch on	Drive is off
xxxx xxxx x1xx 0000	Switch on disabled	Drive is off
xxxx xxxx x01x 0001	Ready to switch on	Drive is off
xxxx xxxx x01x 0011	Switched on	Drive is off
xxxx xxxx x01x 0111	Operation enabled	Drive is enabled
xxxx xxxx x00x 0111	Quick stop active	Drive is enabled
xxxx xxxx x0xx 1111	Fault reaction active	Drive is enabled
xxxx xxxx x0xx 1000	Fault	Drive is off

The state "Operation enabled" is the only one allowing normal operation (motion) of the motor.

The quick stop will automatically transition out of the "Quick stop active" state to the "Switch on disabled" state.

The "Fault reaction active" state will automatically transition to the "Fault" state unless the fault reaction is "slow to a stop" rather than OFF or MTB.

For more details, see Object 6041h: Status Word on page 183.

Control Word (Object 6040h)

The Control Word object (6040h) must be written to command the motor to start motion. Only certain state transitions are allowed. Therefore, the PLC or host writing to the Control Word object (6040h) should read the Status Word object (6041h) to determine the current state.

The following table describes the bits in the Control Word object (6040h). For more details, see Object 6040h: Control Word on page 181.

State to enter	Bits of the Control Word					Allowed from
	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0	
Switch on disabled	0	X	X	0	X	Ready to switch on, Switched on, Operation enabled, Quick stop active (by forcing bit 1 to a 0)
Ready to switch on	0	X	1	1	0	Switch on disabled, Switched on, Operation enabled
Switched on	0	0	1	1	1	Ready to switch on, Operation enabled
Operation Enabled	0	1	1	1	1	Ready to switch on, Switched on
Quick Stop active	0	X	0	1	X	Operation enabled, Ready to switch on, Switched on
Switch on disabled	N/A	N/A	N/A	N/A	N/A	Quick stop active (automatic transition when quick stop completes)
Switch on disabled	0 to 1 transition	X	X	X	X	Fault
Fault	N/A	N/A	N/A	N/A	N/A	Fault reaction active (automatic transition when fault reaction completes)
Fault reaction active	N/A	N/A	N/A	N/A	N/A	Occurrence of a fault will leave current state (automatic transition when fault occurs)
NOTE: Rising edge of bit 7 clears the fault unless a fault condition still exists.						

A typical startup sequence of values to write to the control word is:

1. 0000h — Starting value.
2. 0080h — Clear past faults.
3. 0006h — Enter "Ready to Switch On" state.
4. 000Fh — Enter "Operation Enabled" state; for velocity or torque mode, this starts motion.
5. 001Fh — Start a homing or position move.

Motion Profiles

This section provides example values written to specific objects for various motion profiles.

In these examples, it can be assumed that the writes are made through either PDO or SDO communications. Typically, objects like the Control Word object (6040h) would be written cyclically with PDO communications. However, it is also possible for a single SDO write to set these values. If PDO communications are used, it is assumed that the master is writing values continuously, and the noted sequence indicates when a value should be changed to a new value.



Position Mode

This section describes the process for creating a motion using Absolute Position mode and Relative Position mode.

It is assumed that either the SmartMotor's drive-enable input and hardware limit switch inputs are in the ready state, or the user has issued the appropriate I/O commands to disable the limits. For details, see Object 2100h: Port Configuration on page 136 and Object 2101h: Bit IO on page 137.

Absolute Position Mode Summary

The following table provides a summary of settings for creating a motion using Absolute Position mode. For a different example in step format, see the next section.

Description	SMI Command	Index Object Code	Sub-Index	Data		
				Length	Hex	Dec
Disable positive limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(2)	2101h	03	02	0002	2
Disable negative limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(3)	2101h	03	02	0003	3
Reset status word	ZS	6040h	00	02	0080	<u>128</u> 0000 0000 1000 0000
Set Mode Position	MP	6060h	00	01	01	1
Set profile speed in PP mode	VT=xxxx	6081h	00	04	0000C350	50000
Set target position	PT=0	607Ah	00	04	00000000	0
Set acceleration	AT=xxxx	6083h	00	04	00000064	100
Set deceleration	DT=xxxx	6084h	00	04	00000064	100
Change state: Ready to switch on		6040h	00	02	0006	<u>6</u> 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	<u>7</u> 0000 0000 0000 0111
Enable command, single setpoint (motion not actually started yet)		6040h	00	02	002F	<u>47</u> 0000 0000 0010 1111
Begin motion to target position	G	6040h	00	02	003F	<u>63</u> 0000 0000 0011 1111
Prepare for next command		6040h	00	02	002F	<u>47</u> 0000 0000 0010 1111
Set target position	PT=1000	607Ah	00	04	000003E8	1000
Begin motion to target position	G	6040h	00	02	003F	<u>63</u> 0000 0000 0011 1111

Absolute Position Mode Example

The following procedure shows the steps for creating a motion using Absolute Position mode. For details on Absolute Position mode, see the *SmartMotor™ Developer's Guide*.

NOTE:

Position Units (PU): encoder counts

Acceleration/Deceleration Units (ADU): (encoder counts per (sample²)) * 65536

Velocity Units (VU): encoder counts per sample * 65536

1. Clear the faults by setting the Control Word object (6040h) to the following values:
 - a. 0
 - b. 0080h (128 decimal)
 - c. 0
2. Set the Modes of Operation object (6060h) to the value 1 (decimal).
3. Set the Profile Velocity object (6081h) to the desired speed in VU (for example, the decimal value 100000). This is always a positive value. The target position determines the direction of motion.
4. Set the Profile Acceleration object (6083h) to the desired acceleration in ADU (for example, the decimal value 10).
5. Set the Profile Deceleration object (6084h) to the desired deceleration in ADU (for example, the decimal value 10).
6. Set the Target Position object (607Ah) to the desired absolute position in PU.
7. Initialize and start the motion by setting the Control Word object (6040h) to the values:
 - a. 0006h (6 decimal) — This is required to satisfy the CiA 402 drive state machine. For details, see CiA 402 Profile Motion State Machine on page 51.
 - b. 002Fh (47 decimal) — This configures the single-setpoint positioning mode.
 - c. 003Fh (63 decimal) — The motion begins.
8. Wait for the motion to complete.
9. Set the Target Position object (607Ah) to a new absolute position in PU. Motion will not begin at this time.
10. Initialize, start and stop the motion by setting the Control Word object (6040h) to the following values:
 - a. 002Fh (47 decimal) — Bit 4 must be transitioned for the new setpoint to begin. By writing that value to the Control Word object (6040h), bit 4 will begin in the low state. The next step will write a different value to that object, which will transition bit 4 to a high state.
 - b. 003Fh (63 decimal) — Starts the motion.
 - c. 013Fh (319 decimal) — Stops the motion. The motor will decelerate before reaching the target.
11. Initialize and resume the motion by setting the Control Word object (6040h) to the following values:
 - a. 002Fh (47 decimal) — bit 4 must be transitioned for the motion to resume. By writing that value to the Control Word object (6040h), bit 4 will begin in the low state. The next step will write a different value to that object, which will transition bit 4 to a high state.
 - b. 003Fh (63 decimal) — the motion resumes.
12. Turn off motor by setting the Control Word object (6040h) to the value 0.

Relative Position Example

The following procedure shows the steps for creating a motion using Relative Position mode. For details on Relative Position mode, see the *SmartMotor™ Developer's Guide*.

1. Clear the faults by setting the Control Word object (6040h) to the following values:
 - a. 0
 - b. 0080h (128 decimal)
 - c. 0
2. Set the Modes of Operation object (6060h) to the value 1 (decimal).
3. Set the Profile Velocity object (6081h) to the desired speed in VU (for example, the decimal value 100000). This is always a positive value. The target position determines the direction of motion.
4. Set the Profile Acceleration object (6083h) to the desired acceleration in ADU (for example, the decimal value 10).
5. Set the Profile Deceleration object (6084h) to the desired deceleration in ADU (for example, the decimal value 10).
6. Set a relative target by setting the Target Position object (607Ah) to the desired relative position in PU.
7. Initialize and start the motion by setting the Control Word object (6040h) to the following values:
 - a. 0006h (6 decimal) — This is required to satisfy the 402 drive state machine.
 - b. 006Fh (111 decimal) — This configures the single-setpoint mode of positioning.
 - c. 007Fh (127 decimal) — The motion begins. This sets bit 6 to indicate a relative move.
8. Wait for the motion to complete.

NOTE: If a relative move is commanded while a previous one is in progress, the ending target position for the in-progress move is replaced. The new ending position is calculated by adding the current commanded position (when the command is received) and the relative target (object 607A). The previous ending target position is not a part of this calculation.
9. Set a relative target by setting the Target Position object (607Ah) to the desired relative position in PU. Motion will not begin at this time.
10. Set a new target and start the motion by setting the Control Word object (6040h) to the following values:
 - a. 006Fh (111 decimal) — Bit 4 must be transitioned for the new setpoint to begin. By writing that value to the Control Word object (6040h), bit 4 will begin in the low state. The next step will write a different value to that object, which will transition bit 4 to a high state.
 - b. 007Fh (127 decimal) — The motion begins.
11. Stop the motion by setting the Control Word object (6040h) to the value 017Fh (383 decimal). The motor will decelerate before reaching the target.

12. Initialize and resume the motion by setting the Control Word object (6040h) to the following values:
 - a. 006Fh (111 decimal) — Bit 4 must be transitioned for the motion to resume. By writing that value to the Control Word object (6040h), bit 4 will begin in the low state. The next step will write a different value to that object, which will transition bit 4 to a high state.
 - b. 007Fh (127 decimal) — The motion resumes. It performs a relative move from the current position (not the original position).
13. Turn off motor by setting the Control Word object (6040h) to the value 0.



Velocity Mode

This section describes the process for creating a motion using Velocity mode.

It is assumed that either the SmartMotor's drive-enable input and hardware limit switch inputs are in the ready state, or the user has issued the appropriate I/O commands to disable the limits. For details, see Object 2100h: Port Configuration on page 136 and Object 2101h: Bit IO on page 137.

Velocity Mode Summary

The following table provides a summary of settings for creating a motion using Velocity mode. For a different example in step format, see the next section.

Description	SMI Command	Index Object Code	Sub-Index	Data		
				Length	Hex	Dec
Disable positive limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(2)	2101h	03	02	0002	2
Disable negative limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(3)	2101h	03	02	0003	3
Reset status word	ZS	6040h	00	02	0080	<u>128</u> 0000 0000 1000 0000
Set Mode Velocity	MV	6060h	00	01	03	3
Set velocity in PV mode	VT=xxxx	60FFh	00	04	0000C350	50000
Set acceleration	AT=xxxx	6083h	00	04	00000064	100
Set deceleration	DT=xxxx	6084h	00	04	00000064	100
Change state: Ready to switch on		6040h	00	02	0006	<u>6</u> 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	<u>7</u> 0000 0000 0000 0111
Start command	G	6040h	00	02	000F	<u>15</u> 0000 0000 0000 1111
Update velocity while already running in PV mode	VT=xxxx, G	60FFh	00	04	000186A0	100000
Halt command (set bit 8)	X (default) See object 605Dh	6040h	00	02	010F	<u>271</u> xxxx xxx1 0000 1111
Start command	G	6040h	00	02	000F	<u>15</u> 0000 0000 0000 1111
Quick stop command (bit 2 = 0)	Quick stop then OFF See objects 6085h, 605Ah	6040h	00	02	000B	<u>11</u> xxxx xxxx 0000 1011

Velocity Mode Example

The following procedure shows the steps for creating a motion using Velocity mode. For details on Velocity mode, see the *SmartMotor™ Developer's Guide*.

NOTE:

Position Units (PU): encoder counts

Acceleration/Deceleration Units (ADU): (encoder counts per (sample²)) * 65536

Velocity Units (VU): encoder counts per sample * 65536

1. Clear the faults by setting the Control Word object (6040h) to the following values:
 - a. 0
 - b. 0080h (128 decimal)
 - c. 0
2. Set the Modes of Operation object (6060h) to the value 3 (decimal).
3. Set the Target Velocity object (60FFh) to the desired speed in VU (for example, the decimal value 100000). To reverse the direction of motion, use a negative value.
4. Set the Profile Acceleration object (6083h) to the desired acceleration in ADU (for example, the decimal value 10).
5. Set the Profile Deceleration object (6084h) to the desired deceleration in ADU (for example, the decimal value 10).
6. Set the Control Word object (6040h) to the value 0006h (6 decimal). This is required to satisfy the CiA 402 drive state machine. For details, see CiA 402 Profile Motion State Machine on page 51.
7. Start, stop and resume the motion by setting the Control Word object (6040h) to the following values:
 - a. 000Fh (15 decimal) — Starts the motion
 - b. 010Fh (271 decimal) — Stops the motion
 - c. 000Fh (15 decimal) — Resumes the motion
8. Change the speed by setting the Target Velocity object (60FFh) to the desired speed in VU (for example, the decimal value 200000). The motor will immediately accelerate /decelerate to the new speed. To reverse the direction of motion, use a negative value.
9. Turn off motor by setting the Control Word object (6040h) to the value 0.



Torque Mode

This section describes the process for creating a motion using Torque mode.

It is assumed that either the SmartMotor's drive-enable input and hardware limit switch inputs are in the ready state, or the user has issued the appropriate I/O commands to disable the limits. For details, see Object 2100h: Port Configuration on page 136 and Object 2101h: Bit IO on page 137.

Torque Mode Summary

The following table provides a summary of settings for creating a motion using Torque mode. For a different example in step format, see the next section.

Description	SMI Command	Index Object Code	Sub-Index	Data		
				Length	Hex	Dec
Disable positive limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(2)	2101h	03	02	0002	2
Disable negative limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(3)	2101h	03	02	0003	3
Reset status word	ZS	6040h	00	02	0080	<u>128</u> 0000 0000 1000 0000
Set Mode Torque	MT	6060h	00	01	04	4
Set Torque Slope	TS=xxxx	6087h	00	04	000000C8	200
Set Target Torque	T=xxxx	6071h	00	02	0064	100
Change state: Ready to switch on		6040h	00	02	0006	<u>6</u> 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	<u>7</u> 0000 0000 0000 0111
Start command	G	6040h	00	02	000F	<u>15</u> 0000 0000 0000 1111
Update torque while already running in TQ mode	T=xxxx, G	6071h	00	02	0096	150
Halt command (set bit 8) See object 605Dh	X (default)	6040h	00	02	010F	<u>271</u> xxxx xxx1 0000 1111
Start command	G	6040h	00	02	000F	<u>15</u> 0000 0000 0000 1111
Quick stop command (bit 2 = 0) See object 605Ah	Quick stop then OFF	6040h	00	02	000B	<u>11</u> xxxx xxxx 0000 1011

Torque Mode Example

The following procedure shows the steps for creating a motion using Torque mode. For details on torque mode, see the *SmartMotor™ Developer's Guide*.

NOTE: Units entered for objects 6071h and 6087h are specific to the DS402 profile. In other words, they do not use the units that would be used by the T= or TS= commands. For details, see Object 6071h: Target Torque on page 196. Also, see Object 6087h: Torque Slope on page 208.

1. Clear the faults by setting the Control Word object (6040h) to the following values:
 - a. 0
 - b. 0080h (128 decimal)
 - c. 0
2. Set the Modes of Operation object (6060h) to the value 4 (decimal).
3. Set the Target Torque object (6071h) as desired (for example, the decimal value 100). To reverse the direction of motion, use a negative value.
4. Set the Torque Slope object (6087h) as desired (for example, the decimal value 200). This controls the ramp-up/down rate to the previously-specified Target Torque.
5. Set the Control Word object (6040h) to the value 0006h (6 decimal). This is required to satisfy the CiA 402 drive state machine. For details, see CiA 402 Profile Motion State Machine on page 51.
6. Start, stop and resume the motion by setting the Control Word object (6040h) to the following values:
 - a. 000Fh (15 decimal) — Starts the motion
 - b. 010Fh (271 decimal) — Stops the motion
 - c. 000Fh (15 decimal) — Resumes the motion
7. Change the torque by setting the Target Torque object (6071h) as desired (for example, the decimal value 50). The motor will immediately ramp up/down to the setting. To reverse the direction of motion, use a negative value.
8. Turn off the motor by setting the Control Word object (6040h) to the value 0.

Interpolated Position Mode



Interpolated position (IP) mode allows for buffering and execution of a constant stream of positions. This is useful for host-driven applications with complex motion paths, such as CNC machining.

There are several aspects to this mode of operation that require more effort to configure and operate compared to position, velocity, or torque mode.

- Time synchronization should be used. Because of clock drifts, the individual motors will consume position data at slightly different rates. Over a period of several hours, motors could be significantly out of step (for example, one motor gets several data points ahead of another). With time synchronization, the high-resolution timestamp object is used to coordinate clocks in this process, and the motors will adjust their clocks accordingly.
- Buffer level of data points must be maintained. There are specific objects to monitor, and the host must not allow the buffer of data points to run empty or to overflow. Therefore, the host must be able to accurately monitor and control the flow of data points.
- Data points are entered as absolute positions. However, they are processed in a relative format that depends on the position of the motor at a specific time during the setup and configuration of IP mode.

Interpolated Position Mode Summary

The following table provides a summary of settings for creating a motion using Interpolated Position mode. For a different example in step format, see the next section.

Description	SMI Command	Index Object Code	Sub-Index	Data		
				Length	Hex	Dec
Note: this example works best if motor starts at position 0. See example for position mode to position the motor at a target of 0.						
Disable positive limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(2)	2101h	03	02	0002	2
Disable negative limit switch input  CAUTION: Skip this step if limit switches are in use.	EIGN(3)	2101h	03	02	0003	3
Reset status word	ZS	6040h	00	02	0080	<u>128</u> 0000 0000 1000 0000
Set Mode Interpolation		6060h	00	01	07	7
Change state: Ready to switch on		6040h	00	02	0006	<u>6</u> 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	<u>7</u> 0000 0000 0000 0111
Clear buffer		60C4h	6	01	00	0
Enable buffer		60C4h	6	01	01	1
Set time period to 1 (second)		60C2h	1	01	01	1
Set time period to seconds		60C2h	2	01	00	0
Write data point 1		60C1h	1	04	00000000	0
Write data point 2		60C1h	1	04	000003E8	1000
Write data point 3		60C1h	1	04	00000BB8	3000
Write data point 4		60C1h	1	04	000007D0	2000
Write data point 5		60C1h	1	04	000003E8	1000
Write data point 6		60C1h	1	04	00000000	0
Write zero-length segment		60C2h	1	01	00	0
Write data point		60C1h	1	04	00000000	0
Enable command (motion not actually started yet)		6040h	00	02	000F	<u>15</u> 0000 0000 0000 1111
Begin motion		6040h	00	02	001F	<u>31</u> 0000 0000 0001 1111

Example: Short Run on a Single Motor

This example loads the interpolation buffer with a short set of data and then starts the interpolation. The following procedure is intended for demonstration. Typically, a host will run in IP mode continuously, which is shown in the next example.

1. Use Position mode to place the motor at the starting point for IP mode. For this example, use an absolute move to position 0.

For reasons of initializing the buffer and the starting motor position, it is best to perform a position move or relative-position move (PRT=0) before resetting the interpolation buffer. This will ensure the motor is in the correct state for IP mode.

NOTE: Perform any origin shift *before* the position move — do *not* change the origin (OSH=, O=) after the position move.

2. Set the Control Word object (6040h) to the value 000Fh. Assuming the motor is holding at the starting position, this will leave the drive on.
3. Clear the buffer by setting subindex 6 of the Interpolation Data Configuration object (60C4h) to the value 0.
4. Buffer enable: set subindex 6 of Interpolation Data Configuration object (60C4h) to the value 1.
5. Set the interpolation time:
 - a. Set subindex 2 of the Interpolation Time Period object (60C2h) to the value 0, which designates whole seconds.
 - b. Set subindex 1 of the Interpolation Time Period object (60C2h) to the value 1. When combined with the above setting, this results in one second per data point.
6. Set the Modes of Operation object (6060h) to the value 7.
7. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 0 (the current position)
 - b. 2000
 - c. 6000
 - d. 8000
 - e. 6000
 - f. 3000
 - g. 0 (the final point)
8. Create and write a zero-length segment to end Interpolation mode:
 - a. Write the value 0 to the Interpolation Time Period object (60C2h), subindex 1. This is used to create a zero-length segment to end Interpolation mode.
 - b. Write the value 0 to the Interpolation Data Record object (60C1h), subindex 1. This is the same value as the final point. It writes the final zero-length segment that ends Interpolation mode.

9. Set the Control Word object (6040h) to the value 0006h (6 decimal). This is required to satisfy the CiA 402 drive state machine. For details, see CiA 402 Profile Motion State Machine on page 51.
10. Set the Control Word object (6040h) to the value 000Fh (15 decimal).
11. Start the process by setting the Control Word object (6040h) to the value 1Fh.
12. When the final point has finished, the motor will clear the trajectory bit in the SmartMotor status word. For details, see Object 2304h: Motor Status on page 159.

Also, the Interpolation Mode Status object (2400h) will report if IP mode is running in bit 15. For details, see Object 2400h: Interpolation Mode Status on page 173.

Example: Continuous Run on a Single Motor

This example procedure shows how to continuously operate a host in IP mode.

1. Use Position mode to place the motor at the starting point for IP mode. For this example, use an absolute move to position 0.

For reasons of initializing the buffer and the starting motor position, it is best to perform a position move or relative-position move (PRT=0) before resetting the interpolation buffer. This will ensure the motor is in the correct state for IP mode.

NOTE: Perform any origin shift *before* this position move — do *not* change the origin (OSH=, O=) after this position move.

2. Set the Control Word object (6040h) to the value 000Fh. Assuming the motor is holding at the starting position, this will leave the drive on.
3. Clear the buffer by setting subindex 6 of Interpolation Data Configuration object (60C4h) to the value 0.
4. Enable the buffer by setting subindex 6 of Interpolation Data Configuration object (60C4h) to the value 1.
5. Set the interpolation time:
 - a. Set subindex 2 of the Interpolation Time Period object (60C2h) to the value -3, which designates milliseconds.
 - b. Set subindex 1 of the Interpolation Time Period object (60C2h) to the value 20. Combined with the above setting, this results in 20 milliseconds per data point. Other values are acceptable, of course, depending on the network or host cycle time.
6. Set the Modes of Operation object (6060h) to the value 7.
7. Put data in the buffer:
 - a. Write the value 0 (the current position) to the Interpolation Data Record object (60C1h), subindex 1.
 - b. Write the first data point in units of encoder counts to the Interpolation Data Record object (60C1h), subindex 1.
 - c. Repeat the previous step until a sufficient number of data points are buffered. In other words, enough to keep feeding Interpolation mode if the host has latencies or temporarily becomes unresponsive.
8. Start the process by setting the Control Word object (6040h) to the value 1Fh.

9. Monitor the buffer capacity by using bits 0–6 of the Interpolation Mode Status object (2400h), which will report the number of buffer spaces available. As the number of available spaces approaches 0, the host should wait before sending further data.

NOTE: Bits 0–6 must be masked because the upper bits are used to report other information. For details, see Object 2400h: Interpolation Mode Status on page 173.

10. End Interpolation mode:
 - a. Set subindex 1 of the Interpolation Time Period object (60C2h) to the value 0.
 - b. Repeat the final data point after this time period has been changed, and then write the repeated final data point to Interpolation Data Record object (60C1h), subindex 1. When the motor consumes this point, it will end its trajectory and hold its position. No further data points will be accepted.
11. Set the Control Word object (6040h) to the value 000Fh. This will leave the drive on but holding at the ending position.

Example: Resuming Motion in IP Mode

To resume motion without leaving IP mode:

1. Set the Control Word object (6040h) to the value 000Fh. This will be used later to cause a rising edge on bit 4.
2. Do not clear the buffer. It is not necessary because this example assumes the most recent value written to subindex 6 of the Interpolation Data Configuration object (60C4h) was 1.
3. Set subindex 1 of the Interpolation Time Period object (60C2h) back to the desired value (do not use 0).
4. Add more points to the buffer using subindex 1 of the Interpolation Data Record object (60C1h). Start with the current position.
5. Start the process by setting the Control Word object (6040h) to the value 1Fh. The Interpolation mode will resume. Monitor the buffer capacity and end IP mode as described in the previous examples.

Synchronization

When running multiple motors in Interpolation mode, the rate at which data points are consumed can vary by several parts per million. While this sounds small, over time it will lead to the SmartMotors not reaching a coordinated point simultaneously.

The following brief example is for a network of two motors with the master producing a sync every 10 milliseconds. It is also possible for the time-producer motor to be the sync producer if the CANopen master cannot do so (this method is not shown here).

NOTE: This is an advanced topic that requires an understanding of PDO mapping. For details, see PDO Mapping on page 71.

1. Configure one motor as the time producer:
 - a. Map transmit PDO 4 to object 1013h.
 - b. Set the transmission type to 100 (to transmit at once per second because the sync rate is 100/second). The exact rate is not critical, but it is typically on the order of one second.

2. Configure all other motors as time consumers:

- a. Map receive PDO 4 to object 1013h. Use the same COB-ID that was used to transmit PDO 4 from the time-producer.
- b. Set the transmission type to 254. This will accept the high-resolution timestamp when the time producer transmits it. The most recent sync is the reference point in time where the timestamps from the producer and consumers are compared.

NOTE: The time consumer adjusts itself to match the time producer.

3. Switch to a network operational state.

When the first timestamp is received by the time consumers, they will accept the value without trying to adjust to it. This is considered the starting point, so the consumer clocks are immediately forced to this value instead of adjusting to it. If the synchronization process is interrupted or the motors are switched out of operational network state, then the synchronization process will stop. This means that when it is restarted, it could take significant time for the adjustment process to catch up. Instead, the adjustment process should be forced to reaccept the time as it did at the beginning of the process. This can be accomplished by two different methods:

1. Issue SmartMotor command CANCTL(2,0).
2. Switch to Interpolation mode using object 6060h. The motor must be in a different operating mode, and it must see the transition to value 7 (Interpolation mode) in object 6060h.

User Bits

A special feature is provided by the SmartMotor that allows status bits to be correlated with specific data points in the buffer. The status bit can be used to indicate when a particular segment between two points is achieved. This can be used to call special program routines or to set outputs to control external devices. For example, a laser-engraving tool may need a simple on/off state at certain points in the motion path. This event is correlated with the specific range of positions in the buffer.

To write the bits, write to object 2403h — the range of values is 0 to 3Fh, which represents six bits. These are associated with the next data record written to the Interpolation Data Record object (60C1h), subindex 1. When the associated data point is reached, the new value (bit pattern) will be visible in SmartMotor status word 8 (object 2304h, subindex 9) as bits 8–13.

The following procedure provides an example of the user bits feature. The value of object 2403h is initially 0. The buffer is populated either initially or in a continuous run situation:

1. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 2000
 - b. 3000
2. Set the Interpolation User Bits object (2403h) to the value 1.
3. Put data in the buffer by writing the value 4000 to the Interpolation Data Record object (60C1h), subindex 1.
4. Set the Interpolation User Bits object (2403h) to the value 0.

5. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 5000
 - b. 6000

In the previous example, the user bit indicates when the motor position is between 3000 and 4000. The user bit is accessible in SmartMotor status word 8 (object 2304h, subindex 9). This can be read in a user program with the following code:

```
IF B(8,8)      ' RB(status word 8, bit 8)
OS(0)         ' Set output 0.
ELSE
OR(0)         ' Clear output 0.
ENDIF
```

Splining

By default, object 60C0h is set to 0. This commands the linear form of interpolation. To smooth data points, splined motion can be enabled by setting object 60C0h to the value -3. The change to this mode takes effect with the next data point written through object 60C1h, subindex 1.

NOTE: While it is outside the scope of this manual, it is possible to mix splined and linear interpolation per written data point. This provides interpolation control in cases where spline interpolation does not provide the desired motion path.

Variable-Length Segments

It is possible to vary the length (in time) of the interpolation segment between data points. Object 60C2h, subindex 1 and 2, control the interpolation timer period. There are some cases where it may be beneficial to reduce the required number of points. For example, rounded areas require more points, but straight segments require less points. The application of this technique is outside the scope of this manual. However, note that any change to object 60C2h will be associated with the next position data record written through object 60C1h, subindex 1.

Homing Mode

This section describes the process for activating the SmartMotor homing process.

- For homing modes 1, and 17 there must be a negative limit switch connected and enabled. The positive limit may also be present or not, but it cannot be faulted.
- For homing modes 2, and 18 there must be a positive limit switch connected and enabled. The negative limit may also be present or not, but it cannot be faulted.
- For other homing modes, the limit switches must either be cleared of faults, or they must be disabled.

Homing Summary

The following table provides a summary of settings for activating the homing process. For a different example in step format, see the next section.

Description	SMI Command	Index Object Code	Sub-Index	Data		
				Length	Hex	Dec
Note: Limit switches must be physically connected in this example. The negative limit switch will be used as the home reference.						
Reset status word	ZS	6040h	00	02	0080	128 0000 0000 1000 0000
Set Mode Homing (HM)		6060h	00	01	06	6
Set homing method		6098h	00	01	01	1
Set homing speed 1		6099h	01	04	000186A0	100000
Set homing speed 2		6099h	02	04	00002710	10000
Set homing acceleration		609Ah	00	04	00000064	100
Set homing offset		607Ch	00	04	000003E8	1000
Change state: Ready to switch on		6040h	00	02	0006	6 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	7 0000 0000 0000 0111
Enable operation		6040h	00	02	000F	15 0000 0000 0000 1111
Start command	G	6040h	00	02	001F	31 0000 0000 0001 1111
The homing will begin by heading toward the negative limit.						

Homing Example

The following procedure shows the steps for activating the homing process.

NOTE:

Position Units (PU): encoder counts

Acceleration/Deceleration Units (ADU): (encoder counts per (sample²)) * 65536

Velocity Units (VU): encoder counts per sample * 65536

1. Clear the faults by setting the Control Word object (6040h) to the following values:
 - a. 0
 - b. 0080h (128 decimal)
 - c. 0
2. Set the Modes of Operation object (6060h) to the value 6 (decimal).
3. Set the Homing Method object (6098h) to the method desired. For details, see Object 6098h: Homing Method on page 210.

4. Set subindex 1 of the Homing Speed object (6099h) to the desired speed in VU (for example, the decimal value 100000). This is always a positive value. The Homing mode determines the direction of motion.
5. Set subindex 2 of the Homing Speed object (6099h) to the desired speed in VU (for example, the decimal value 100000). This is always a positive value. The Homing mode determines the direction of motion.
6. Set the Homing Acceleration object (609Ah) to the desired acceleration in ADU (for example, the decimal value 10).
7. (Optional) Set the Home Offset object (607Ch) to the desired homing offset in PU.
8. Initialize and start the motion by setting the Control Word object (6040h) to the following values:
 - a. 0006h (6 decimal) — This is required to satisfy the CiA 402 drive state machine. For details, see CiA 402 Profile Motion State Machine on page 51.
 - b. 000Fh (15 decimal)
 - c. 001Fh (31 decimal) — The motion begins.
9. Wait for the motion to complete. The Status Word object (6041h) will report when the home position has been located. When the motor has come to a stop, then bit 10 = 1 (target reached) and bit 12 = 1 (home position found).

If bit 13 = 1 in the Status Word object (6041h), there was an error and homing was not completed.

PDO Mapping

This chapter provides information on the Process Data Objects (PDOs) and the PDO mapping process. It also describes the low-level steps that must occur at startup between the master and the motor to enable PDO communications.

Overview	72
Mapping and Communication Parameters Objects	73
Communications Parameters Objects	74
Mapping Parameters Objects	75
Mapping Entries	75
Mapping Procedure	76
Time Sync Motors Mapping Procedure	76
Example Start-up Sequence	78

Overview

Process Data Objects (PDOs) are containers that hold one or more data objects. The set of objects in a PDO can be configured through the process of dynamic mapping. In a SmartMotor, this means that data objects such as the Velocity Actual Value object (606Ch) and the Status Word object (6041h) can be placed in the same PDO transmission from the SmartMotor. The same can be done for receive PDOs — the motor will unpack the received PDO according to the mapping configuration and consume the data objects.

A CAN packet contains a maximum payload of 8 bytes. This creates a limit to the amount of data that is mapped into a single PDO. For example, a PDO can contain one INTEGER32 and two INTEGER16 objects. Other combinations are allowed, but the number of bytes must be 8 or less.

A set of objects is available for performing object mapping. These objects are included in the set known as the Communication Profile objects (1000h-1FFFh). This is the standard for any CANopen devices that support dynamic mapping. For details on the Communication Profile objects, see Communication Profile on page 97.

NOTE: Some CANopen masters may have a graphical interface or automated means of performing this mapping.

Mapping and Communication Parameters Objects

The following table lists the overall set of mapping and communication parameters objects. Note that all of these contain sub-objects, which are described in the tables later in this section.

Object		
decimal	hex	Description
5120	1400	Receive PDO1 Communication Parameters
5121	1401	Receive PDO2 Communication Parameters
5122	1402	Receive PDO3 Communication Parameters
5123	1403	Receive PDO4 Communication Parameters
5124	1404	Receive PDO5 Communication Parameters
5632	1600	Receive PDO1 Mapping Parameters
5633	1601	Receive PDO2 Mapping Parameters
5634	1602	Receive PDO3 Mapping Parameters
5635	1603	Receive PDO4 Mapping Parameters
5636	1604	Receive PDO5 Mapping Parameters
6144	1800	Transmit PDO1 Communication Parameters
6145	1801	Transmit PDO2 Communication Parameters
6146	1802	Transmit PDO3 Communication Parameters
6147	1803	Transmit PDO4 Communication Parameters
6148	1804	Transmit PDO5 Communication Parameters
6656	1A00	Transmit PDO1 Mapping Parameters
6657	1A01	Transmit PDO2 Mapping Parameters
6658	1A02	Transmit PDO3 Mapping Parameters
6659	1A03	Transmit PDO4 Mapping Parameters
6660	1A04	Transmit PDO5 Mapping Parameters

Communications Parameters Objects

The following table describes the Communications Parameters objects (receive and transmit), which have sub-objects of the same structure.

Subindex (decimal)	Description
0	Number of Entries: The number of sub-objects in the object; the value is 5 (read only).
1	COB-ID: This PDO will listen for CAN packets with this identifier (Receive PDO) or transmit CAN packets with this identifier (Transmit PDO).
2	<p>Transmission Type:</p> <p>Value 0: N/A</p> <p>Value 1: Transmit on sync packet (Transmit PDO). Accept data on sync packet (Receive PDO). The Transmit PDO is sent when a sync packet is seen.</p> <p>Values 2–240: Same as value 1, except the rate is divided (e.g., the value 2 specifies every other sync packet).</p> <p>Values 241–251: Reserved.</p> <p>Values 252, 253: Not supported.</p> <p>Value 254: Transmit if the self-timer has expired. This mode simply transmits this PDO at the rate of the event timer.</p> <p>Value 255: Transmit if either the event timer period expires or an object mapped in the PDO changes value. The event timer for each PDO resets each time a transmission occurs through either mechanism. Therefore, the event timer is a maximum time between transmissions; the inhibit time is a minimum time between transmissions.</p>
3	Inhibit time: Limits how often a transmission is allowed. This is typically left at the default setting. The units are: value * 100 microseconds (i.e., a value of 1 is 100 microseconds).
4	Compatibility entry: Use the default setting.
5	Event Timer: The maximum time (in milliseconds) between transmissions of this PDO if the transmission type value for a transmit PDO is 254 or 255.

Mapping Parameters Objects

The following table describes the Mapping Parameters objects (receive and transmit), which have sub-objects of the same structure.

Subindex (decimal)	Description
0	Number of Entries: Defines the number of objects that are mapped within this PDO. For instance, if "Mapping Entry 1" and "Mapping Entry 2" have been set up, then write the value 2.
1	Mapping Entry 1: Points to the mapped object. For details, see the following sections.
2	Mapping Entry 2: Points to the mapped object. For details, see the following sections.
3	Mapping Entry 3: Points to the mapped object. For details, see the following sections.
4	Mapping Entry 4: Points to the mapped object. For details, see the following sections.

Mapping Entries

Only four mapping entries are allocated for the SmartMotor. Therefore, a maximum of four objects can be mapped into a PDO. The mapping entries must be filled contiguously starting from mapping entry 1. For example, for three entries, use mapping entry 1, 2 and 3.

All of these mapping entries are UNSIGNED32-bit values. There are three pieces of data packed into each of these fields to represent the object being mapped:

- The object number
- The object subindex (0 if none)
- The object size (in bits)

Therefore, in the form: (hex) nnnniiss

- n: object number
- i: subindex
- s: size

The following example uses the Velocity Actual Value object (606Ch):

(hex) 606c0020



CAUTION: There is a specific procedure defined by the CANopen specification for mapping a variable. This procedure must be followed or an error will occur, which will prevent the change to the mapping.

Mapping Procedure

The following procedure uses the previous Velocity Actual Value object example. Transmit PDO 1 is mapped to contain the Velocity Actual Value object (606Ch) and the Status Word object (6041h).

1. Enter the NMT Pre-Operational state.
2. Set bit 31 of the COB-ID — set subindex 1 of the Transmit PDO Communication Parameter 1 object (1800h) to the value C0000180h. This assumes that subindex 1 of object 1800h has been set to the default value 40000180h.
3. Set the number of entries to 0 in subindex 0 of the Transmit PDO Mapping Parameter 1 object (1A00h).
4. Using the same object (1A00h), set the mapping object. It uses a 32-bit value with the following order: highest 2 bytes: object; next byte: subindex; the last byte: length in bits.
 - a. For the status word, set subindex 1 = 60410010h.
 - b. For the actual velocity, set subindex 2 = 606c0020h.
5. Using the same object (1A00h), set the number of entries back to the number of items created in the previous step — set subindex 0 to the value 2.
6. Clear bit 31 of the COB-ID — set subindex 1 of object 1800h to the value 40000180h. This will specify this PDO to transmit with the COB-ID of 180h.
7. Set the Transmission Type in subindex 2 of object 1800h to "sync" (1-240) or "event timer" (254-255).

If the "event timer" is chosen, then also specify the number of milliseconds between transmissions in subindex 5 of object 1800h.
8. Enter the NMT Operational state.

Time Sync Motors Mapping Procedure

The following procedure maps SmartMotors to synchronize (following) motion based on an external encoder input.

1. External encoder:
 - a. Node-ID: 100
 - b. Transmit PDO 1:
 1. Transmission type 1 (sync)
 2. COB ID: 1e4h
 3. Mapping: object 6004h (32-bit)

2. SmartMotor 1

- a. Node-ID: 1
- b. Receive PDO 1:
 - 1. Transmit type 1 (sync reception)
 - 2. COB ID: 1e4h (encoder's transmit PDO)
 - 3. Mapping: object 2208h, subindex 3 (32 bit)
- c. Receive PDO 2:
 - 1. Transmit type 1 (sync reception)
 - 2. COB ID: 301h
 - 3. Mapping: object 2209h, subindex 0 (16 bit)
- d. Receive PDO 3: (optional if control word desired as PDO – not required to synchronize following mode.)
 - 1. Transmit type 1 (sync reception)
 - 2. COB ID: 401h
 - 3. Mapping: object 6040h, subindex 0 (16 bit)

3. SmartMotor 2

- a. Node-ID: 2
- b. Receive PDO 1:
 - 1. Transmit type 1 (sync reception)
 - 2. COB ID: 1e4h (encoder's transmit PDO.)
 - 3. Mapping: object 2208h, subindex 3 (32 bit)
- c. Receive PDO 2:
 - 1. Transmit type 1 (sync reception)
 - 2. COB ID: 301h (same as motor 1)
 - 3. Mapping: object 2209h, subindex 0 (16 bit)
- d. Receive PDO 3: (optional if control word desired as PDO – not required to synchronize following mode),
 - 1. Transmit type 1 (sync reception)
 - 2. COB ID: 402h
 - 3. Mapping: object 6040h, subindex 0 (16 bit)

4. Use the same process for the remaining motors.

Also, see the following example start-up sequence that is based on this mapping.

Example Start-up Sequence

The following is an example start-up sequence based on an application using the previous mapping.

1. Power up of system
2. PLC/Master: Set encoder PDO parameters, PDO mapping, any other settings.
3. PLC/Master: Set motor 1 PDO mappings, PDO parameters. Repeat for motor 2, 3, etc.
4. PLC/Master: Set motor 1 object 1006h to cycle time in micro seconds. Example: 10 millisecond sync rate: set value to 10000. Repeat for motor 2, 3, etc.
5. PLC/Master: begin sending sync packet (COB-ID 80h) continuously at selected rate.
6. PLC/Master: Set object 6060h to value -11 in each SmartMotor.
7. PLC/Master: Clear limits in each SmartMotor(if no hardware limit switches):
 - a. Write to object 2101h, subindex 3: value = 2
 - b. Write to object 2101h, subindex 3: value = 3
8. PLC/Master: configure object 220Ah – 220Dh in each motor to configure follow ratio, and ramp-up/ramp-down rates in follow mode.
9. PLC/Master: configure object 2207h in each motor to configure the maximum expected value from the encoder. For example if encoder has position range of 0 to 4095 (4096 resolution), then set value in object 2207h to the value: 4095.
10. PLC/Master: Set NMT state to operational (broadcast to encoder and all motors.)
11. PLC/Master: clear faults in each SmartMotor:
 - a. Write 0080h to object 6040h.
 - b. Write 0000h to object 6040h.
12. PLC/Master: enable operation in each SmartMotor:
 - a. Write 0006h to object 6040h.
 - b. Write 0007h to object 6040h.
 - c. Write 000Fh to object 6040h.
13. Motors are now following the encoder.
14. To halt all SmartMotors, send RxPDO 2 (mapped to object 2209h) with bit 0 set to value 1. All motors receive this at the same sync interval. All motors will begin ramp down equivalent to X(2) command.
15. To resume all SmartMotors, send RxPDO 2 (mapped to object 2209h) with bit 0 set to value 0. All motors receive this at the same sync interval. All motors will begin ramp up equivalent to G(2) command.

Also, see the example user program: CAN Bus - Time Sync Follow Encoder in Chapter 3 of the *SmartMotor Developer's Guide* for a similar application that uses a SmartMotor as the "master".

CANopen User Program Commands

This chapter provides details on the CANopen commands used with the SmartMotor and its user program. SmartMotor programming is described in the *SmartMotor™ Developer's Guide*. The SmartMotor user program allows the motor to take on autonomous or distributed control functions needed in an application.

NOTE: The CAN network must have all devices set to the same baud rate for proper operation.

Address and Baud Rate Commands	80
CADDR=frm	80
CBAUD=frm	80
CAN Error Reporting Commands	80
=CAN, RCAN	80
RB(2,4), x=B(2,4)	83
Network Control Commands	83
CANCTL(action, value)	83
NMT(address, command code)	85
SDORD(address, obj index, subindex, bytecount)	86
SDOWR(address, obj index, subindex, bytecount, data)	86
Exceptions to NMT, SDORD and SDOWR Commands	87

Address and Baud Rate Commands

The following are related commands. For more details on these commands, see the *SmartMotor™ Developer's Guide*.

CADDR=frm

Set can address

Where frm is a number from 1 to 127. The value is stored in the SmartMotor's EEPROM. However, the SmartMotor must be powered off and on for it to take effect.

CBAUD=frm

Set CAN baud rate

Where frm may be one of the following bit rates (bits/second): 1000000, 800000, 500000, 250000, 125000, 50000 and 20000. The value is stored in the SmartMotor's EEPROM. However, the SmartMotor must be powered off and on for it to take effect.

The setting of 10000 bits/second is not supported. For details on other unsupported CANopen features, see Not Supported on page 33.

CAN Error Reporting Commands

The following are related commands. For more details on these commands, see the *SmartMotor™ Developer's Guide*.

=CAN, RCAN

Get CAN error

The =CAN and RCAN commands are used to assign/report errors and certain status information for the CAN bus.

- Assigned to a program variable: x=CAN(y)
- As a report: RCAN(y)

Where y is the following:

Assignment	Report	Description																																		
=CAN(0)	RCAN(0)	Gets the CAN bus status bits: (*Indicates an error bit)																																		
		<table><tr><th>Bit</th><th>Description</th></tr><tr><td>0</td><td>CAN power okay (not used by CANopen)</td></tr><tr><td>1*</td><td>DeviceNet COM fault occurred (not used by CANopen)</td></tr><tr><td>2</td><td>DeviceNet Power Ignore option enabled (not used by CANopen)</td></tr><tr><td>3</td><td>Reserved</td></tr><tr><td>4*</td><td>User attempted a Combitronic read from broadcast address</td></tr><tr><td>5*</td><td>Combitronic debug, internal issue.</td></tr><tr><td>6*</td><td>Timeout (Combitronic client)</td></tr><tr><td>7*</td><td>Combitronic server ran out of buffer slots</td></tr><tr><td>8*</td><td>Errors reached warning level</td></tr><tr><td>9*</td><td>Receive Errors reached warning level</td></tr><tr><td>10*</td><td>Transmit Errors reached warning level</td></tr><tr><td>11*</td><td>Receive Passive Error</td></tr><tr><td>12*</td><td>Transmit Passive Error</td></tr><tr><td>13*</td><td>Bus Off Error</td></tr><tr><td>14*</td><td>RX buffer 1 overflowed</td></tr><tr><td>15*</td><td>RX buffer 0 overflowed</td></tr></table>	Bit	Description	0	CAN power okay (not used by CANopen)	1*	DeviceNet COM fault occurred (not used by CANopen)	2	DeviceNet Power Ignore option enabled (not used by CANopen)	3	Reserved	4*	User attempted a Combitronic read from broadcast address	5*	Combitronic debug, internal issue.	6*	Timeout (Combitronic client)	7*	Combitronic server ran out of buffer slots	8*	Errors reached warning level	9*	Receive Errors reached warning level	10*	Transmit Errors reached warning level	11*	Receive Passive Error	12*	Transmit Passive Error	13*	Bus Off Error	14*	RX buffer 1 overflowed	15*	RX buffer 0 overflowed
		Bit	Description																																	
		0	CAN power okay (not used by CANopen)																																	
		1*	DeviceNet COM fault occurred (not used by CANopen)																																	
		2	DeviceNet Power Ignore option enabled (not used by CANopen)																																	
		3	Reserved																																	
		4*	User attempted a Combitronic read from broadcast address																																	
		5*	Combitronic debug, internal issue.																																	
		6*	Timeout (Combitronic client)																																	
		7*	Combitronic server ran out of buffer slots																																	
		8*	Errors reached warning level																																	
		9*	Receive Errors reached warning level																																	
		10*	Transmit Errors reached warning level																																	
		11*	Receive Passive Error																																	
		12*	Transmit Passive Error																																	
		13*	Bus Off Error																																	
14*	RX buffer 1 overflowed																																			
15*	RX buffer 0 overflowed																																			
=CAN(1)	RCAN(1)	Gets the value of the current NMT state: <ul style="list-style-type: none">• Pre-Operational: 127• Operational: 5• Stopped: 4																																		
=CAN(2)	RCAN(2)	Gets the value of the Control Word object (6040h)																																		
=CAN(3)	RCAN(3)	Gets the value of the Status Word object (6041h)																																		

Assignment	Report	Description		
=CAN(4)	RCAN(4)	Gets the result code of the most recent SDO read or write, or NMT command as a master.		
		Code	Description	Type
		0	No error (operation succeeded).	SDO NMT
		>0	Error from remote device as defined by CANopen and/or remote device datasheet. Refer to SDO Response Error Codes.	SDO
		-1	Timeout	SDO
		-2	Multiple commands issued (tried to call an SDO operation while SDO was busy, such as from an interrupt).	SDO
		-3	Master disabled. Use appropriate CANCTL(...) function to activate master mode.	SDO NMT
		-4	Protocol not supported. The remote device attempted to respond with an unsupported method.	SDO
		-5	Transmit fail. Hardware, baud rate, cabling or similar is causing a backlog in the transmit buffer.	SDO NMT
		-6	Wrong size. SDORD command was called with one data size; response from remote device was different size.	SDO read
		-20	Invalid host. The remote device address is out of possible range.	SDO NMT
		-21	Invalid data size (requested an unsupported size).	SDO
		-22	Invalid object index (outside the allowed range). Index must be from 0 to 65535.	SDO
		-23	Invalid object subindex (outside the allowed range). subindex must be from 0 to 255.	SDO
		-24	Invalid NMT command state. Requested NMT state is out of range (this is a gross range check; it doesn't imply all values in the range are valid).	NMT

The =CAN(0) and RCAN(0) commands are used to report a bit map of conditions that could occur over the CAN bus. Not all bits are error bits. Therefore, it cannot be assumed that a nonzero value for RCAN is an error.

RCAN, which is the same as RCAN(0), reports a decimal number that is a combination of the bits shown in the =CAN(0)/RCAN(0) row of the previous table. Use the CAN command, which is the same as =CAN(0), in a program to assign the decimal number to a variable, for example:

x=CAN

A calculator with a binary display function can convert this decimal number to indicate the set of bits shown. Also, the *SmartMotor Developer's Worksheet* can be used for this conversion. It is available from the Moog Animatics website at:

<http://www.animatics.com/support/download-center.html>

NOTE: Object 2304h, subindex 3, bit 4 (CAN error) reports true if any of the error indications above are set. In a user program, this is a simpler test than attempting to filter the result of RCAN for the error conditions.

RB(2,4), x=B(2,4)

Determine if CAN error has occurred

Report/get if an error state has occurred over CAN, CANopen or Combitronic. Further investigation through RCAN(0) will give more details. This can be cleared using the Z(2,4) or ZS command.

For more details, see the *SmartMotor™ Developer's Guide*.

Network Control Commands

The following are related commands. For more details on these commands, see the *SmartMotor™ Developer's Guide*.

CANCTL(action, value)

Control network features

Commands execute based on the action argument, which controls CAN functions.

Action =	Description
0	Reserved; not used in CANopen firmware.
1	Reset the CAN communications controller in the motor and all errors. Resets the CANopen protocol in the motor. The value argument is ignored.
2	Reset the activity of the CANopen clock sync using the high-resolution time stamp. The value argument is ignored.
3	<p>This action uses the following value arguments:</p> <ul style="list-style-type: none"> Value = 0: Reset the CANopen interpolation buffer through user command. Leaves the buffer disabled to prevent new data points. Value = 1: Reset the CANopen interpolation buffer through user command. Sets buffer access to allow new data points.
4	Use of this command is discouraged. It was previously provided to force the motion mode of operation from a user program. However, this functionality is now available through existing SmartMotor commands such as MV, MP, MT, MC, MFR, MSR and MD. For details on these commands, see the <i>SmartMotor™ Developer's Guide</i> .
5	Set timeout for Combitronic. The value argument specifies the time in milliseconds; it defaults to 30 (for 30 milliseconds).
12	<p>This action uses the following value arguments:</p> <ul style="list-style-type: none"> Value = 0: Clears bit 14 in the status word (6041h). This is the default value at power-up of the motor. Value = 1: Sets bit 14 in the status word (6041h).
13	<p>This action uses the following value arguments:</p> <ul style="list-style-type: none"> Value = 0: Disables access to several objects listed below. Clears "remote" bit 9 in the status word (6041h). Value = 1: Enables access to several objects listed below. By default, this is the state at power-up of the motor. Sets "remote" bit 9 in the status word (6041h). <p>The affected objects are:</p> <ul style="list-style-type: none"> 6040h: Control Word 6060h: Modes of Operation 6071h: Target Torque 6081h: Profile Velocity (pp mode) 6083h: Profile Acceleration 6084h: Profile Deceleration 6087h: Torque Slope 60FBh: Subindex 1–8, 10 (PID parameters) 60FFh: Target Velocity
14	<p>Enable/disable Combitronic time sync based on <value>:</p> <p>0 - Disable 1 - Enable as slave (default at power up) 2 - Enable as master</p>

Action =	Description
15	Set timeout in milliseconds, where: <ul style="list-style-type: none"> Value=[time in milliseconds] Default value is 1000 milliseconds <p>If the program encounters a PRINT statement while the buffer is still waiting to be read, then the program pauses either for the specified time or until the buffer is read—the pause lasts only for the shorter condition. After this timeout, the string is replaced by the output from the pending PRINT statement.</p>
16 ^a	Sets the SDO command read/write timeout period. SDO reads or writes initiated by the SmartMotor acting as CANopen master will wait up to this time before declaring a timeout. <value> sets time in msec; range from 10 to 1000. Default value is 500.
17 ^{a,b}	Enables master commands: NMT, SDORD, SDOWR. For future support of master functionality and features, certain number ranges are reserved (see next table).
18	CSP mode sync priority control: 5.0.4.49 / 5.98.0.49 or later 0: Default, disables this special mode 1: Enable sync timing priority suppression (uncommon)
a. Requires firmware 5.x.4.30 or later; these are for CANopen only. b. When not enabled, by default, the commands NMT, SDORD, SDOWR will return an error instead of the intended action/value. Bit 0 of status word 10 will be a 1 (true) when master is enabled.	

Reserved values for CANCTL(17,value):

Value	Description
-1	Disable master commands (default)
0-2	Reserved
3	Enable master commands; "simple mode" (no flying master, no monitoring of nodes, no EMCY support, no LSS support)
4-9999	Reserved

NMT(address, command code)

For command codes and range of addresses, see NMT Control on page 27

NOTE: See CANCTL(17,x) on page 85, which is required to activate this command.

Transmit NMT message to network

The NMT command transmits an NMT message to the network; it can command a specific slave or all slaves to enter the commanded state. The command uses the form:

NMT(target address, desired state)

```
NMT(0,1)    'Tell everyone to go operational.
NMT(2,128)  'Tell motor 2 to go pre-operational.
x=CAN(4)
IF x!=0
    ' NMT command failed.
ENDIF
```

SDORD(address, obj index, subindex, bytecount)

Read value from SDO

The SDORD command gets (reads) the value from the specified SDO on a specified device.

NOTE: See CANCTL(17,x) on page 85, which is required to activate this command.

EXAMPLE: Read an SDO

```
x=SDORD(1, 24592,0,2)  ' Read 2 bytes from address 1,
                        ' object 0x6010, sub-index 0.
e=CAN(4)                ' Get any error information

y=SDORD(1, 24608,0,2)  ' Read 2 bytes from address 1,
                        ' object 0x6020, sub-index 0.
ee=CAN(4)               ' Get any error information

IF (e|ee)==0            ' Confirm the status of both SDO operations.
    ' Success
    b=x                ' Set some example variable according
    c=y                ' to the data received.
    GOSUB(3)            ' Some routine to take action when this data is valid.
ELSE
    GOSUB(8)            ' Go do something to deal with error when read fails.
ENDIF
```

SDOWR(address, obj index, subindex, bytecount, data)

Write value to SDO

The SDOWR command writes a value to the specified SDO on a specified device.

NOTE: See CANCTL(17,x) on page 85, which is required to activate this command.

EXAMPLE: Write an SDO

```
a=1234
SDOWR(1,9029,0,4,a)    ' Write 4 bytes to address 1,
IF CAN(4)==0            ' Confirm the status of the most recent SDO operation.
    ' Success
    GOSUB(4)            ' Some routine to take action when the write succeeds.
ELSE
    GOSUB(9)            ' Go do something to deal with error when write fails.
ENDIF
```

Exceptions to NMT, SDORD and SDOWR Commands

Note the following exceptions when using the NMT, SDORD, SDOWR commands:

- No Combitronic version of these commands, i.e., there is no ":" operator form of the command, for example:
 `x=SDORD(...):3`
is not allowed. Refer to each command's description in Part 2 of this guide.
- No monitoring the heartbeat of other network nodes.
- No special commands for sending or receiving PDOs. PDOs must be mapped to existing objects to send or receive data as a slave device. Even the SmartMotor designated as a master must configure its own PDO mappings.

NOTE: SmartMotors currently have 5 transmit and 5 receive PDOs.

- No capability to read EDS files. The user is responsible for writing a program with the relevant object index, subindex and data type.
- No LSS host behavior is provided from the SmartMotor. Each slave device is expected to have the properly configured address and baud rate. Each device must have a unique address; all devices must use the same baud rate. Any need to set the baud rate or address is not the responsibility of Moog Animatics.
- Only one SmartMotor may fill the master role. No other SmartMotors on the network may issue these commands, because this implementation does not support a multi-CANopen-master functionality.
- No support for master read/write of segmented or block SDO protocol. Only Expedited (32-bit or smaller) data transmission are supported by the master functionality.

For more details and example programs, see the *SmartMotor™ Developer's Guide*.

Troubleshooting

The following table provides troubleshooting information for solving SmartMotor problems that may be encountered when using CANopen. For additional support resources, see the Moog Animatics Support page at:

<http://www.animatics.com/support.html>

Issue	Cause	Solution
CANopen Communication Issues		
Master does not recognize motor.	Motor not powered.	Check Drive Status LED. If LED is not lit, check wiring.
	Disconnected or miswired CAN connector, or broken wiring between motors.	Check that CANopen connector is correctly wired and connected to motor. For details, see Connections, Wiring and Status LEDs on page 34.
	Wrong CAN BAUD rate.	Set CBAUD setting and then reboot motor. For details, see Address and Baud Rate Commands on page 80.
	Wrong CAN node ID (address)	Set CADDR setting and then reboot motor. For details, see Address and Baud Rate Commands on page 80.
	Wrong firmware	Contact Moog Animatics for the correct firmware version.
	Wrong bus topology, or wrong placement of terminators.	The CAN bus should be a linear bus topology. For details, see Connections, Wiring and Status LEDs on page 34.
	Line lengths or drop lengths of CAN bus are too long.	Decrease line and/or drop lengths. For details, see Connections, Wiring and Status LEDs on page 34.
	Network flooded with traffic.	Set master temporarily to the Pre-Operational state. Stop user programs in all motors. For details, see NMT States on page 26.
Red CAN error LED.	A warning or bus off condition has occurred.	Check CAN Bus Network Fault LED — A blinking red LED may indicate occasional issues from any of the causes listed above; a solid red LED indicates that these issues have occurred frequently, which causes the motor to stop communicating (bus off condition). In this case, the SmartMotor must be reset after fixing the cause of the problem.

Issue	Cause	Solution
Communication and Control Issues		
Motor control power light does not illuminate.	Motor is equipped with the DE option.	To energize control power, apply 24-48 VDC to pin 15 and ground to pin 14.
	Motor has routed drive power through drive-enable pins.	Ensure cabling is correct and drive power is not being delivered through the 15-pin connector.
Motor does not communicate with SMI.	Transmit, receive, or ground pins are not connected correctly.	Ensure that transmit, receive and ground are all connected properly to the host PC.
	Motor program is stuck in a continuous loop or is disabling communications.	To prevent the program from running on power up, use the Communications Lockup Wizard located on the SMI software Communications menu.
Motor disconnects from SMI sporadically.	COM port buffer settings are too high.	Adjust the COM port buffer settings to their lowest values.
	Poor connection on serial cable.	Check the serial cable connections and/or replace it.
	Power supply unit (PSU) brownout.	PSU may be too high-precision and/or undersized for the application, which causes it to brown-out during motion. Make moves less aggressive, increase PSU size, or change to a linear unregulated power supply.
Motor stops communicating over serial port after power reset, requires re-detection.	Motor does not have its address set in the user program. NOTE: Serial addresses are lost when motor power is off or reset.	Use the SADDR or ADDR= command within the program to set the motor address.
Red PWR SERVO light illuminated.	Critical fault.	To discover the source of the fault, use the Motor View tool located on the SMI software Tools menu.
Common Faults		
Bus voltage fault.	Bus voltage is either too high or too low for operation.	Check servo bus voltage. If motor uses the DE power option, ensure that both drive and control power are connected.
Overcurrent occurred.	Motor intermittently drew more than its rated level of current. Does not cease motion	Consider making motion less abrupt with softer tuning parameters or acceleration profiles.
Excessive temperature fault.	Motor has exceeded temperature limit of 85°C. Motor will remain unresponsive until it cools down below 80°C.	Motor may be undersized or ambient temperature is too high. Consider adding heat sinks or forced air cooling to the system.
Excessive position error.	The motor's commanded position and actual position differ by more than the user-supplied error limit.	Increase error limit, decrease load, or make movement less aggressive.

Issue	Cause	Solution
Historical positive/negative hardware limit faults.	A limit switch was tripped in the past.	Clear errors with the ZS command.
	Motor does not have limit switches attached.	Configure the motor to be used without limit switches by setting their inputs as general use.
Programming and SMI Issues		
Several commands not recognized during compiling.	Compiler default firmware version set incorrectly.	Use the "Compiler default firmware version option" in the SMI software Compile menu to select the default firmware version closest to the motor firmware version. In the SMI software, view the motor firmware version by right-clicking the motor and selecting Properties.

SDO Response Error Codes

The following table shows the list of possible errors (abort codes) from a remote device as defined by CANopen and/or remote device datasheet.

NOTE: Unlisted codes are reserved.

Code		Description
Hex	Dec	
0503 0000h	84082688	Toggle bit not alternated.
0504 0000h	84148224	SDO protocol timed out. Note that the SmartMotor uses the RCAN(4) -1 value to indicate a timeout. Refer to Troubleshooting on page 88.
0504 0001h	84148225	Client/server command specifier not valid or unknown.
0504 0002h	84148226	Invalid block size (block mode only).
0504 0003h	84148227	Invalid sequence number (block mode only).
0504 0004h	84148228	CRC error (block mode only).
0504 0005h	84148229	Out of memory.
0601 0000h	100728832	Unsupported access to an object.
0601 0001h	100728833	Attempt to read a write only object.
0601 0002h	100728834	Attempt to write a read only object.
0602 0000h	100794368	Object does not exist in the object dictionary.
0604 0041h	100925505	Object cannot be mapped to the PDO.
0604 0042h	100925506	Number and length of objects to be mapped would exceed PDO length.
0604 0043h	100925507	General parameter incompatibility reason.
0604 0047h	100925511	General internal incompatibility in the device.
0606 0000h	101056512	Access failed due to a hardware error.

Code		Description
Hex	Dec	
0607 0010h	101122064	Data type does not match—length of service parameter does not match.
0607 0012h	101122066	Data type does not match—length of service parameter too high.
0607 0013h	101122067	Data type does not match—length of service parameter too low.
0609 0011h	101253137	Subindex does not exist.
0609 0030h	101253168	Value range of parameter exceeded (only for write access).
0609 0031h	101253169	Value of parameter written too high.
0609 0032h	101253170	Value of parameter written too low.
0609 0036h	101253174	Maximum value is less than minimum value.
0800 0000h	134217728	General error.
0800 0020h	134217760	Data cannot be transferred or stored to the application.
0800 0021h	134217761	Data cannot be transferred or stored to the application because of local control.
0800 0022h	134217762	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	134217763	Object dictionary dynamic generation fails or no object dictionary is present (e.g., object dictionary is generated from file and generation fails because of a file error).

Object Reference

This chapter provides details on the CANopen objects used with the Moog Animatics SmartMotor. The following TOC groups the objects by category.

Object Categories	96
Communication Profile	97
Object 1000h: Device Type	99
Object 1001h: Error Register	100
Object 1005h: COB-ID SYNC	101
Object 1006h: Communication Cycle Period	103
Object 1008h: Manufacturer Device Name	105
Object 1009h: Manufacturer Hardware Version	106
Object 100Ah: Manufacturer Software Version	107
Object 1013h: High-Resolution Timestamp	108
Object 1017h: Producer Heartbeat Time	109
Object 1018h: Identity Object	110
Object 1200h: Server SDO Parameter 1	111
Object 1400h: Receive PDO Communication Parameter 1	112
Object 1401h: Receive PDO Communication Parameter 2	113
Object 1402h: Receive PDO Communication Parameter 3	114
Object 1403h: Receive PDO Communication Parameter 4	115
Object 1404h: Receive PDO Communication Parameter 5	116
Object 1600h: Receive PDO Mapping Parameter 1	117
Object 1601h: Receive PDO Mapping Parameter 2	118
Object 1602h: Receive PDO Mapping Parameter 3	119
Object 1603h: Receive PDO Mapping Parameter 4	120
Object 1604h: Receive PDO Mapping Parameter 5	121
Object 1800h: Transmit PDO Communication Parameter 1	122
Object 1801h: Transmit PDO Communication Parameter 2	123
Object 1802h: Transmit PDO Communication Parameter 3	124
Object 1803h: Transmit PDO Communication Parameter 4	125
Object 1804h: Transmit PDO Communication Parameter 5	126
Object 1A00h: Transmit PDO Mapping Parameter 1	127
Object 1A01h: Transmit PDO Mapping Parameter 2	128

Object 1A02h: Transmit PDO Mapping Parameter 3	129
Object 1A03h: Transmit PDO Mapping Parameter 4	130
Object 1A04h: Transmit PDO Mapping Parameter 5	131
Manufacturer-Specific Profile	132
Object 2000h: Node Id	134
Object 2001h: Bit Rate Index	135
Object 2100h: Port Configuration	136
Object 2101h: Bit IO	137
Object 2200h: User EEPROM	138
Object 2201h: User Variable	139
Object 2202h: Set Position Origin	140
Object 2203h: Shift Position Origin	141
Object 2204h: Mappable 32-bit Variables	142
Object 2205h Negative Software Position Limit	143
Object 2206h Positive Software Position Limit	144
Object 2207h Encoder Modulo Limit	145
Object 2208h Encoder Follow Data	146
Object 2209h Encoder Follow Control	147
Start/Stop Capability	147
Object 220Ah MFMUL	149
Object 220Bh MFDIV	150
Object 220Ch MFA	151
Object 220Dh MFD	152
Object 2220h: 8-Bit Mappable Variables	153
Object 2221h: 16-Bit Mappable Variables	154
Object 2300h: Bus Voltage	155
Object 2301h: RMS Current	156
Object 2302h: Internal Temperature	157
Object 2303h: Internal Clock	158
Object 2304h: Motor Status	159
Object 2305h: Motor Control	168
Object 2306h: Motor Subroutine Index	169
Object 2307h: Sample Period	170
Object 2308h: Microsecond Clock	171
Object 2309h: GOSUB R2	172

Object 2400h: Interpolation Mode Status	173
Object 2401h: Buffer Control	174
Object 2402h: Buffer Setpoint	175
Object 2403h: Interpolation User Bits	176
Object 2404h: Interpolation Sample Clock	177
Object 2500h: Encapsulated SmartMotor Command	178
Drive and Motion Control Profile	179
Object 6040h: Control Word	181
Object 6041h: Status Word	183
Object 605Ah: Quick Stop Option Code	184
Object 605Dh: Halt Option Code	185
Object 605Eh: Fault Reaction Option Code	186
Object 6060h: Modes of Operation	187
Object 6061h: Modes of Operation Display	189
Object 6062h: Position Demand Value	190
Object 6063h: Position Actual Internal Value	191
Object 6064h: Position Actual Value	192
Object 6065h: Following Error Window	193
Object 606Bh: Velocity Demand Value	194
Object 606Ch: Velocity Actual Value	195
Object 6071h: Target Torque	196
Object 6074h: Torque Demand Value	197
Object 6077h: Torque Actual	198
Object 6079h: DC Link Circuit Voltage	199
Object 607Ah: Target Position	200
Object 607Ch: Home Offset	201
Object 6080h: Max Motor Speed	203
Object 6081h: Profile Velocity in PP Mode	204
Object 6083h: Profile Acceleration	205
Object 6084h: Profile Deceleration	206
Object 6085h: Quick Stop Deceleration	207
Object 6087h: Torque Slope	208
Object 608Fh: Position Encoder Resolution	209
Object 6098h: Homing Method	210
Object 6099h: Homing Speeds	213

Object 609Ah: Homing Acceleration	214
Object 60C0h: Interpolation Sub-Mode Select	215
Object 60C1h: Interpolation Data Record	216
Object 60C2h: Interpolation Time Period	217
Object 60C4h: Interpolation Data Configuration	219
Object 60F4h: Following Error Actual Value	220
Object 60FBh: Position Control Parameter Set	221
Object 60FCh: Position Demand Internal Value	223
Object Description	223
Entry Description	223
Object 60FDh: Digital Inputs	224
Object 60FEh: Digital Outputs	226
Object 60FFh: Target Velocity	228
Object 6402h: Motor Type	229
Object 6502h: Supported Drive Modes	230
Object 67FFh: Single Device Type	231

Object Categories

The object descriptions are grouped by the following categories.

- Communication Profile on page 97

This set of objects in the range 1000h to 1FFFh implement the 301 specification for general CANopen communications. This configures CANopen services and PDO behavior.

- Manufacturer-Specific Profile on page 132

This set of objects in the range 2000h to 5FFFh implement manufacturer-specific objects, which do not follow a common standard. They provide access to SmartMotor commands and data.

- Drive and Motion Control Profile on page 179

This set of objects in the range 6000h to 67FFh implement the CiA 402 motion profile. This provides access to common commands for controlling the motor.

Communication Profile

This section describes the objects in the Communication Profile. This set of objects in the range 1000h to 1FFFh implement the 301 specification for general CANopen communications. This configures CANopen services and PDO behavior.

Object 1000h: Device Type	99
Object 1001h: Error Register	100
Object 1005h: COB-ID SYNC	101
Object 1006h: Communication Cycle Period	103
Object 1008h: Manufacturer Device Name	105
Object 1009h: Manufacturer Hardware Version	106
Object 100Ah: Manufacturer Software Version	107
Object 1013h: High-Resolution Timestamp	108
Object 1017h: Producer Heartbeat Time	109
Object 1018h: Identity Object	110
Object 1200h: Server SDO Parameter 1	111
Object 1400h: Receive PDO Communication Parameter 1	112
Object 1401h: Receive PDO Communication Parameter 2	113
Object 1402h: Receive PDO Communication Parameter 3	114
Object 1403h: Receive PDO Communication Parameter 4	115
Object 1404h: Receive PDO Communication Parameter 5	116
Object 1600h: Receive PDO Mapping Parameter 1	117
Object 1601h: Receive PDO Mapping Parameter 2	118
Object 1602h: Receive PDO Mapping Parameter 3	119
Object 1603h: Receive PDO Mapping Parameter 4	120
Object 1604h: Receive PDO Mapping Parameter 5	121
Object 1800h: Transmit PDO Communication Parameter 1	122
Object 1801h: Transmit PDO Communication Parameter 2	123
Object 1802h: Transmit PDO Communication Parameter 3	124
Object 1803h: Transmit PDO Communication Parameter 4	125
Object 1804h: Transmit PDO Communication Parameter 5	126

Object 1A00h: Transmit PDO Mapping Parameter 1	127
Object 1A01h: Transmit PDO Mapping Parameter 2	128
Object 1A02h: Transmit PDO Mapping Parameter 3	129
Object 1A03h: Transmit PDO Mapping Parameter 4	130
Object 1A04h: Transmit PDO Mapping Parameter 5	131

Object 1000h: Device Type

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1000h	000	Device Type	00000000h	FFFFFFFFh	00020192h	No	Unsigned 32-bit	Read Only

This object is required by CANopen to provide information about this device. The value of this object does not change.

Bit	Meaning
0–15 (16 bits)	Device profile: 402 (192 hex)
16–23 (8 bits)	Device type: 02 hex, to indicate a single instance of a servo drive
24–31 (8 bits)	Device mode: 0 (manufacturer-specific / reserved)

Also, refer to Object 67FFh: Single Device Type on page 231.

Object 1001h: Error Register

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1001h	000	Error Register	00h	FFh		No	Unsigned 8-bit	Read Only

The value read from this object contains a bit field with the following meaning:

Bit	Function
0	General error Includes any of the following: <ul style="list-style-type: none"> • motion fault • drive not ready • CAN communication errors • program command error • program checksum error • serial communication error
1-7	Reserved

Object 1005h: COB-ID SYNC

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1005h	000	COB-ID SYNC	00000001h	FFFFFFFFh	00000080h	No	Unsigned 32-bit	Read Write

This object specifies the COB-ID used for the Synchronization object (transmit or receive).

Bit	Setting
0–10	COB-ID of the Synchronization object.
11–28	Set to 0.
29	Set to 0 for typical 11-bit identifiers.
30	Set to 0 to be a sync consumer (receive). Set to a 1 to be a sync producer (transmit). Sync message is produced in any operation mode: Stopped, Operational and Pre-Operational. NOTE: The Communication Cycle Period object (1006h) must be set before setting this bit to 1; otherwise, an SDO abort error will be issued.
31	Set to 0 (not used).

For example, the motor is a:

- Sync *consumer* with the default sync COB-ID of 80h: 00000080h
- Sync *producer* with the default sync COB-ID of 80h: 40000080h

EXAMPLE: (for the complete program, see the example "CAN Bus - Time Sync Follow Encoder" in Chapter 3 of the *SmartMotor Developer's Guide*)

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x1005    4101 ' Object 1005h: COB-ID Sync
#define x1006    4102 ' Object 1006h: Communication Cycle Period
. . .
mmm=1           ' network master's address
fff=mmm         ' following motor's address. In this demo, it is the network
                 ' master. But you can make it a 3rd-party
eee=2           ' encoders address
. . .
NMT(0,128)      GOSUB10 ' Network broadcast to go pre-operational state.
' Setup the sync producer/consumers and set timebase. Provides time sync so
' motor clocks keep in step, and data is transmitted/accepted on sync also.
SDOWR(mmm,x1006,0,4,10000) GOSUB10 ' define Cycle period object 0x1006:0,
                                   ' size 4, 10ms
SDOWR(eee,x1006,0,4,10000) GOSUB10 ' define Cycle period object 0x1006:0,
                                   ' size 4, 10ms
IF mmm!=fff
' If follow motor is not the master.
```

```
SDOWR(fff,x1006,0,4,10000)  GOSUB10  ' define Cycle period  object
                                ' 0x1006:0, size 4, 10ms

ENDIF
SDOWR(mmm,x1005,0,4,128)    GOSUB10  ' define Cycle ID x0000 0080 (required
                                ' to avoid error in next line.)
SDOWR(mmm,x1005,0,4,1073741952) GOSUB10  ' define Cycle ID, producer
                                ' x4000 0080
SDOWR(eee,x1005,0,4,128)    GOSUB10  ' define Cycle ID, consumer x0000 0080
IF mmm!=fff
    ' If follow motor is not the master.
    SDOWR(fff,x1005,0,4,128)  GOSUB10  ' define Cycle ID, consumer x0000 0080
ENDIF
. . .
C10
    ' Code to check for CAN error and display it
. . .
RETURN
```

Object 1006h: Communication Cycle Period

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1006h	000	Communication Cycle Period	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object defines the communication cycle period in microseconds for transmission of the sync message. Set to 0 to disable the sync message transmission. Also, refer to Object 1005h: COB-ID SYNC on page 101.

For certain applications, this object can be used to provide the following features:

- Network encoder following: When receiving encoder data, it will arrive at a rate of several milliseconds between samples. For smooth motion, the SmartMotor must interpolate this data internally at a faster rate to take smaller steps per PID cycle. The motor will know the time interval based on object 1006h, cycle period.
- Synchronization: When the SmartMotors should all have a common timebase, this allows data to be produced and consumed one-for-one. The arrival time of sync packets from the master and the value set as the cycle period by object 1006h will coordinate this behavior.

EXAMPLE: (for the complete program, see the example "CAN Bus - Time Sync Follow Encoder" in Chapter 3 of the *SmartMotor Developer's Guide*)

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x1005    4101 ' Object 1005h: COB-ID Sync
#define x1006    4102 ' Object 1006h: Communication Cycle Period
. . .
mmm=1           ' network master's address
fff=mmm         ' following motor's address.  In this demo, it is the network
                ' master.  But you can make it a 3rd-party
eee=2           ' encoders address
. . .
NMT(0,128)      GOSUB10 ' Network broadcast to go pre-operational state.
' Setup the sync producer/consumers and set timebase. Provides time sync so
' motor clocks keep in step, and data is transmitted/accepted on sync also.
SDWR(mmm,x1006,0,4,10000) GOSUB10 ' define Cycle period  object 0x1006:0,
                                ' size 4, 10ms
SDWR(eee,x1006,0,4,10000) GOSUB10 ' define Cycle period  object 0x1006:0,
                                ' size 4, 10ms

IF mmm!=fff
' If follow motor is not the master.
SDWR(fff,x1006,0,4,10000) GOSUB10 ' define Cycle period  object
                                ' 0x1006:0, size 4, 10ms
ENDIF
SDWR(mmm,x1005,0,4,128) GOSUB10 ' define Cycle ID x0000 0080 (required
                                ' to avoid error in next line.)
SDWR(mmm,x1005,0,4,1073741952) GOSUB10 ' define Cycle ID, producer
                                ' x4000 0080
SDWR(eee,x1005,0,4,128) GOSUB10 ' define Cycle ID, consumer x0000 0080
```

```
IF mmm!=fff
  ' If follow motor is not the master.
  SDWR(fff,x1005,0,4,128)  GOSUB10  ' define Cycle ID, consumer x0000 0080
ENDIF
. . .
C10
  ' Code to check for CAN error and display it
  . . .
RETURN
```


Object 1008h: Manufacturer Device Name

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1008h	000	Manufacturer Device Name			SMClass5	No	String	Read Only

This object contains the manufacturer device name. This value does not change and reports as:

SMClass5

Object 1009h: Manufacturer Hardware Version

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1009h	000	Manufacturer Hardware Version			01.00	No	String	Read Only

This object contains the device hardware version. This value does not change and reports as:

01.00

Object 100Ah: Manufacturer Software Version

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
100Ah	000	Manufacturer Software Version				No	String	Read Only

This object contains the firmware version of the motor. It reports a string in the format:

5.x.y.z

Where x can be:

- 0 for CANopen D-style motor
- 98 for CANopen M-style motor

The y and z positions represent the major and minor software release version, respectively.

The string is 16 bytes long; it is padded with null characters at the end.

Similar SmartMotor Commands: RFW, RSP (firmware) info

Object 1013h: High-Resolution Timestamp

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1013h	000	High-Resolution Timestamp	00000000h	FFFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Write

This object contains a timestamp with a resolution of 1 microsecond. It can be mapped into a PDO in order to define a high-resolution timestamp.

Typically, one motor is configured to transmit its object 1013, and one or more other motors receive this value for the purpose of synchronization.

When this object is read, it is the captured value of the high-resolution timer at the most recent sync; therefore, it is not the current value.

NOTE: The captured value *is not* the current value of the high-resolution timer.

When this object is written, it is used to skew the motor's internal timing to stay synchronized with other motors.

For more details, see Synchronization on page 66.

Object 1017h: Producer Heartbeat Time

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1017h	000	Producer Heartbeat Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object defines the cycle time of the heartbeat transmission from the motor in milliseconds. Transmission begins as soon as the value is set. If the value is 0, nothing is transmitted.

The heartbeat contains information that tells the master (or other devices) that the heartbeat came from this device and what network state it is in (Operational, Pre-Operational, Stopped).

Object 1018h: Identity Object

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1018h	000	Number of Entries	01h	04h	04h	No	Unsigned 8-bit	Read Only
1018h	001	Vendor ID	00000000h	FFFFFFFFh	00000226h	No	Unsigned 32-bit	Read Only
1018h	002	Product Code	00000000h	FFFFFFFFh	00000003h	No	Unsigned 32-bit	Read Only
1018h	003	Revision Number	00000000h	FFFFFFFFh	Revision number	No	Unsigned 32-bit	Read Only
1018h	004	Serial Number	00000000h	FFFFFFFFh	Motor serial number	No	Unsigned 32-bit	Read Only

This object contains general information about the device. These values are constant and do not change.

- Subindex 1 contains the Vendor ID number assigned to Moog Animatics: 226h.
- Subindex 2 contains the manufacturer-specific product code (varies by product):

Product	Code
Class 5	3
Class 6 EtherCAT	1
Class 6 SL17	5

- Subindex 3 contains the revision number:
 - Bit 31–16 is the major revision number
 - Bit 15–0 is the minor revision number
- Subindex 4 contains the unique serial number of this SmartMotor. This number is the same as the serial number printed on the SmartMotor label, except that the leading alpha character is dropped. Only the 24-bit numeric digits are reported.

Object 1200h: Server SDO Parameter 1

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1200h	000	Number of Entries	02h	02h	02h	No	Unsigned 8-bit	Read Only
1200h	001	COB-ID Client to Server	00000600h	BFFFFFFFh	00000600 + node ID	No	Unsigned 32-bit	Read Only
1200h	002	COB-ID Server to Client	00000580h	BFFFFFFFh	00000580 + node ID	No	Unsigned 32-bit	Read Only

These are the COB-ID values used for SDO communications from the CANopen master to the SmartMotor. The value is automatically updated based on the node ID (motor address) according to the default connection set. This information cannot be changed; it is provided for informative purposes only.

Object 1400h: Receive PDO Communication Parameter 1

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1400h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1400h	001	COB-ID	00000001h	FFFFFFFFh	00000200h + node ID	No	Unsigned 32-bit	Read Write
1400h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1400h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1400h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1400h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of receive PDO 1.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1401h: Receive PDO Communication Parameter 2

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1401h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1401h	001	COB-ID	00000001h	FFFFFFFFh	80000300h + node ID	No	Unsigned 32-bit	Read Write
1401h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1401h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1401h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1401h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of receive PDO 2.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1402h: Receive PDO Communication Parameter 3

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1402h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1402h	001	COB-ID	00000001h	FFFFFFFFh	80000400h + node ID	No	Unsigned 32-bit	Read Write
1402h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1402h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1402h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1402h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of receive PDO 3.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1403h: Receive PDO Communication Parameter 4

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1403h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1403h	001	COB-ID	00000001h	FFFFFFFFh	80000500h + node ID	No	Unsigned 32-bit	Read Write
1403h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1403h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1403h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1403h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of receive PDO 4.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1404h: Receive PDO Communication Parameter 5

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1404h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1404h	001	COB-ID	00000001h	FFFFFFFFh	80000000h	No	Unsigned 32-bit	Read Write
1404h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1404h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1404h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1404h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of receive PDO 5.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1600h: Receive PDO Mapping Parameter 1

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1600h	000	Number of Entries	00h	04h	01h	No	Unsigned 8-bit	Read Write
1600h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60400010h	No	Unsigned 32-bit	Read Write
1600h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1600h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1600h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into receive PDO 1.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1601h: Receive PDO Mapping Parameter 2

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1601h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1601h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60400010h	No	Unsigned 32-bit	Read Write
1601h	002	Mapping Entry 2	00000000h	FFFFFFFFh	60600008h	No	Unsigned 32-bit	Read Write
1601h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1601h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into receive PDO 2.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1602h: Receive PDO Mapping Parameter 3

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1602h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1602h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60400010h	No	Unsigned 32-bit	Read Write
1602h	002	Mapping Entry 2	00000000h	FFFFFFFFh	607A0020h	No	Unsigned 32-bit	Read Write
1602h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1602h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into receive PDO 3.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1603h: Receive PDO Mapping Parameter 4

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1603h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1603h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60400010h	No	Unsigned 32-bit	Read Write
1603h	002	Mapping Entry 2	00000000h	FFFFFFFFh	60FF0020h	No	Unsigned 32-bit	Read Write
1603h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1603h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into receive PDO 4.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1604h: Receive PDO Mapping Parameter 5

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1604h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1604h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60400010h	No	Unsigned 32-bit	Read Write
1604h	002	Mapping Entry 2	00000000h	FFFFFFFFh	60710010h	No	Unsigned 32-bit	Read Write
1604h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1604h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into receive PDO 5.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1800h: Transmit PDO Communication Parameter 1

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1800h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1800h	001	COB-ID	00000001h	FFFFFFFFh	40000180h + node ID	No	Unsigned 32-bit	Read Write
1800h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1800h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1800h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1800h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of transmit PDO 1.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1801h: Transmit PDO Communication Parameter 2

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1801h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1801h	001	COB-ID	00000001h	FFFFFFFFh	C0000280h + node ID	No	Unsigned 32-bit	Read Write
1801h	002	Transmission Type	00h	FFh	FFh	No	Unsigned 8-bit	Read Write
1801h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1801h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1801h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of transmit PDO 2.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1802h: Transmit PDO Communication Parameter 3

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1802h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1802h	001	COB-ID	00000001h	FFFFFFFFh	C0000380h + node ID	No	Unsigned 32-bit	Read Write
1802h	002	Transmission Type	00h	FFh	01h	No	Unsigned 8-bit	Read Write
1802h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1802h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1802h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of transmit PDO 3.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1803h: Transmit PDO Communication Parameter 4

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1803h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1803h	001	COB-ID	00000001h	FFFFFFFFh	C0000480h + node ID	No	Unsigned 32-bit	Read Write
1803h	002	Transmission Type	00h	FFh	01h	No	Unsigned 8-bit	Read Write
1803h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1803h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1803h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of transmit PDO 4.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1804h: Transmit PDO Communication Parameter 5

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1804h	000	Number of Entries	02h	05h	05h	No	Unsigned 8-bit	Read Only
1804h	001	COB-ID	00000001h	FFFFFFFFh	C0000000h	No	Unsigned 32-bit	Read Write
1804h	002	Transmission Type	00h	FFh	01h	No	Unsigned 8-bit	Read Write
1804h	003	Inhibit Time	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
1804h	004	Compatibility Entry	00h	FFh	00h	No	Unsigned 8-bit	Read Write
1804h	005	Event Timer	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of transmit PDO 5.

For the following items, refer to the table Communications Parameters Objects on page 74:

- Sub-index 0: Number of subindex objects in this object (5)
- Sub-index 1: COB-ID used for this PDO; when set, bit 31 is used to disable the PDO
- Sub-index 2: Transmission type
- Sub-index 3: Inhibit time in units of 100 microseconds
- Sub-index 4: Not used; use the default setting
- Sub-index 5: Event time, in milliseconds, for transmit PDOs type 254 and 255

Object 1A00h: Transmit PDO Mapping Parameter 1

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1A00h	000	Number of Entries	00h	04h	01h	No	Unsigned 8-bit	Read Write
1A00h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60410010h	No	Unsigned 32-bit	Read Write
1A00h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1A00h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1A00h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into transmit PDO 1.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1A01h: Transmit PDO Mapping Parameter 2

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1A01h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1A01h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60410010h	No	Unsigned 32-bit	Read Write
1A01h	002	Mapping Entry 2	00000000h	FFFFFFFFh	60610008h	No	Unsigned 32-bit	Read Write
1A01h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1A01h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into transmit PDO 2.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1A02h: Transmit PDO Mapping Parameter 3

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1A02h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1A02h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60410010h	No	Unsigned 32-bit	Read Write
1A02h	002	Mapping Entry 2	00000000h	FFFFFFFFh	60640020h	No	Unsigned 32-bit	Read Write
1A02h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1A02h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into transmit PDO 3.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1A03h: Transmit PDO Mapping Parameter 4

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1A03h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1A03h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60410010h	No	Unsigned 32-bit	Read Write
1A03h	002	Mapping Entry 2	00000000h	FFFFFFFFh	606C0020h	No	Unsigned 32-bit	Read Write
1A03h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1A03h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into transmit PDO 4.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Object 1A04h: Transmit PDO Mapping Parameter 5

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1A04h	000	Number of Entries	00h	04h	02h	No	Unsigned 8-bit	Read Write
1A04h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60410010h	No	Unsigned 32-bit	Read Write
1A04h	002	Mapping Entry 2	00000000h	FFFFFFFFh	60770010h	No	Unsigned 32-bit	Read Write
1A04h	003	Mapping Entry 3	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write
1A04h	004	Mapping Entry 4	00000000h	FFFFFFFFh	00000000h	No	Unsigned 32-bit	Read Write

This object controls which objects are mapped into transmit PDO 5.

For the following items, refer to Mapping Parameters Objects on page 75:

- Subindex 0: Number of valid subindex objects in this object. This is set according to the filled mapping entries starting from subindex 1.
- Subindexes 1–4: These provide information about the object mapped in this PDO. They contain the indexes, the subindexes and the lengths of the mapped object. Fill these starting from subindex 1. The structure is:

Bit	Meaning
Bits 16–31 (16 bit)	Index of the object to map
Bits 8–15 (8 bit)	Subindex of the object to map
Bits 0–7 (8 bit)	Length of the object (in bits)

Manufacturer-Specific Profile

This section describes the objects in the Manufacturer-Specific Profile. This set of objects in the range 2000h to 5FFFh implement manufacturer-specific objects, which do not follow a common standard. They provide access to SmartMotor commands and data.

Object 2000h: Node Id	134
Object 2001h: Bit Rate Index	135
Object 2100h: Port Configuration	136
Object 2101h: Bit IO	137
Object 2200h: User EEPROM	138
Object 2201h: User Variable	139
Object 2202h: Set Position Origin	140
Object 2203h: Shift Position Origin	141
Object 2204h: Mappable 32-bit Variables	142
Object 2205h Negative Software Position Limit	143
Object 2206h Positive Software Position Limit	144
Object 2207h Encoder Modulo Limit	145
Object 2208h Encoder Follow Data	146
Object 2209h Encoder Follow Control	147
Object 220Ah MFMUL	149
Object 220Bh MFDIV	150
Object 220Ch MFA	151
Object 220Dh MFD	152
Object 2220h: 8-Bit Mappable Variables	153
Object 2221h: 16-Bit Mappable Variables	154
Object 2300h: Bus Voltage	155
Object 2301h: RMS Current	156
Object 2302h: Internal Temperature	157
Object 2303h: Internal Clock	158
Object 2304h: Motor Status	159
Object 2305h: Motor Control	168

Object 2306h: Motor Subroutine Index	169
Object 2307h: Sample Period	170
Object 2308h: Microsecond Clock	171
Object 2309h: GOSUB R2	172
Object 2400h: Interpolation Mode Status	173
Object 2401h: Buffer Control	174
Object 2402h: Buffer Setpoint	175
Object 2403h: Interpolation User Bits	176
Object 2404h: Interpolation Sample Clock	177
Object 2500h: Encapsulated SmartMotor Command	178

Object 2000h: Node Id

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2000h	000	Node Id	00h	FFh	Loaded from EEPROM at boot-up	Yes	Unsigned 8-bit	Read Only

This object contains the active CANopen ID.

Object 2001h: Bit Rate Index

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2001h	000	Bit Rate Index	00h	08h	Loaded from EEPROM at boot-up	Yes	Unsigned 8-bit	Read Only

This object reports the current CAN bit rate setting. The value is reported as an index representing the bit rate. Refer to the following table:

Index	Bit rate (kilobits/sec)
0	1000
1	800
2	500
3	250
4	125
5	N/A
6	50
7	20
8	N/A

Object 2100h: Port Configuration

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2100h	000	Port Configuration	00000000h	7FFFFFFFh	000000A0h	Yes	Unsigned 32-bit	Read Write

This object controls the configuration of I/O ports 0 through 6 (formerly named A through G) on a D-style motor. Due to constraints, some of the inputs are grouped together. For example, ports 4 and 5 (formerly named E and F) can be configured together for RS-485. For more details, see I/O on page 45.

This object is not supported in M-style firmware (5.98.x.x). For details, see I/O on page 45.

Object 2100			Port	Port
Bits	Binary Bits	Value	Effect	Effect
			0 (A)	1 (B)
0–3 (4 bits)	xxxx xxxx xxxx 0000	0	input	input
	xxxx xxxx xxxx 0001	1	output	input
	xxxx xxxx xxxx 0010	2	input	output
	xxxx xxxx xxxx 0011	3	output	output
			2 (C)	
4–5 (2 bits)	xxxx xxxx xx00 xxxx	0	input	
	xxxx xxxx xx01 xxxx	1	output	
	xxxx xxxx xx10 xxxx	2	positive limit	
			3 (D)	
6–7 (2 bits)	xxxx xxxx 00xx xxxx	0	input	
	xxxx xxxx 01xx xxxx	1	output	
	xxxx xxxx 10xx xxxx	2	negative limit	
			4 (E)	5 (F)
8–10 (3 bits)	xxxx x000 xxxx xxxx	0	input	input
	xxxx x001 xxxx xxxx	1	output	input
	xxxx x010 xxxx xxxx	2	input	output
	xxxx x011 xxxx xxxx	3	output	output
	xxxx x100 xxxx xxxx	4	I ² C	
	xxxx x101 xxxx xxxx	5	RS-485	
			6 (G)	
11–12 (2 bits)	xxx0 0xxx xxxx xxxx	0	input	
	xxx0 1xxx xxxx xxxx	1	output	
	xxx1 0xxx xxxx xxxx	2	go	
13–15 (3 bits)	N/A	N/A	Reserved	

Object 2101h: Bit IO

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2101h	000	Number of Entries	03h	03h	03h	No	Unsigned 8-bit	Read Only
2101h	001	Set Output	0000h	7FFFh*	0000h	Yes	Unsigned 16-bit	Read Write
2101h	002	Clear Output	0000h	7FFFh*	0000h	Yes	Unsigned 16-bit	Read Write
2101h	003	Make Input	0000h	7FFFh*	0000h	Yes	Unsigned 16-bit	Read Write
*Class 5 firmware doesn't necessarily return an error specific to the IO ports actually present; anything unsupported may be silently ignored.								

This object allows individual control of each I/O point. It is designed for SDO-type communications at startup. It is not intended for cyclic PDO communications.

The value written is the identifier of the I/O port to be controlled. The action to take on that port is a function of the specified subindex object:

- subindex 1: Drive the specified I/O high.
- subindex 2: The action depends on I/O type:
 - For D-style motor ports 0–6, drive the specified I/O low
 - For D-style motor ports 16–25, turn off the specified I/O
 - For M-style motor ports 0–10, turn off the specified I/O
- subindex 3: Turn off the specified I/O and disable certain special function such as a limit input. The specified I/O point will simply become a generic input.

For example, to make I/O port 2 (formerly named port C) a generic input, write the value 2 to subindex 3.

For more I/O details, see I/O on page 45.

Object 2200h: User EEPROM

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2200h	000	Number of Entries	00h	FFh	03h	No	Unsigned 8-bit	Read Only
2200h	001	EEPROM index	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
2200h	002	EEPROM number of bytes	00h	FFh	00h	No	Unsigned 8-bit	Read Write
2200h	003	EEPROM value				No	String: 8 bytes.	Read Write

This object provides access to user non-volatile EEPROM memory. Through SDO commands, a value can be written to the user EEPROM. To do this:

1. Set the EEPROM index (subindex 1) to the EEPROM location where the new value will be written. Typical values are 0 to 32339.
2. Set subindex 2 to the number of bytes that will be written.
3. Write the binary data to the EEPROM in subindex 3. Up to 8 bytes may be written at a time.

Object 2201h: User Variable

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2201h	000	Number of Entries	00h	FFh	03h	No	Unsigned 8-bit	Read Only
2201h	001	Index	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Write
2201h	002	Data Type	80h	7Fh	00h	No	Signed 8-bit	Read Write
2201h	003	Value	80000000h	7FFFFFFFh	00000000h	No	Signed 32-bit	Read Write

This object provides access to user variables through SDO commands. To do this:

1. Set the index (subindex 1) to the user variable that a value will be written to or read from. Refer to the following table to determine the correct index.
2. Set subindex 2 according to the table for the desired variable-type access.
3. Read or write the data using subindex 3.

Only one variable is written at a time. If the data type is ab[] or aw[], a single byte or word is written, respectively.

Data type (subindex 2)	Index (subindex 1)	Variable's data type	Variables accessed
0	0–25	long (32-bit)	a–z
0	26–51	long (32-bit)	aa–zz
0	52–77	long (32-bit)	aaa–zzz
1	0–50	long (32-bit)	al[Index]
2	0–101	word (16-bit)	aw[Index]
3	0–203	byte (8-bit)	ab[Index]

The variable arrays: al[index], aw[index] and ab[index] overlap the same physical memory of 204 bytes. This allows different access to common memory based on data size. For instance, al[0] is the same region as ab[0] through ab[3]. The byte order is little-endian, such that ab[0] is the lowest byte of al[0].

For more details, see User Variables on page 45.

Object 2202h: Set Position Origin

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2202h	000	Set Position Origin	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write

The value written to this object becomes the new position value. Both the commanded position (RPC) and actual position (RPA) are shifted by this value minus the current command value. The value read from this object is the most recent value written to this object — it is *not* an indication of the motor's current state.

Similar SmartMotor Commands: O=

Object 2203h: Shift Position Origin

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2203h	000	Shift Position Origin	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write

This object shifts the absolute position (RPA) and the commanded position (RPC) by the specified value. Each time this value is written, the position is shifted by that amount. The value read from this object is the most recent value written to this object — it is *not* an indication of the motor's current state.

Similar SmartMotor Commands: OSH=

Object 2204h: Mappable 32-bit Variables

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2204h	000	Number of Entries	04h	04h	04h	No	Unsigned 8-bit	Read Only
2204h	001	aaa	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write
2204h	002	bbb	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write
2204h	003	ccc	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write
2204h	004	ddd	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write

This object provides direct read or write access to user variables aaa–ddd. This object is provided to fill the need for PDO access to user variables. SDO access is also allowed. Also, see Object 2220h: 8-Bit Mappable Variables on page 153 and Object 2221h: 16-Bit Mappable Variables on page 154.

For more details, see User Variables on page 45.

Object 2205h Negative Software Position Limit

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2205h	000	Negative Software Position Limit	80000000h	7FFFFFFFh	80000000h	Yes	Signed 32-bit	Read Write

This object defines the negative software position limit in units of encoder counts. If the software position limits are enabled and the actual position is out of range, then a software-limit fault occurs.

The term "negative" does not imply the value must be negative. Positive values are permitted; however, they should be a lower value than the positive software position limit.

Similar SmartMotor Commands: SLN=, RSLN

Object 2206h Positive Software Position Limit

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2206h	000	Positive Software Position Limit	80000000h	7FFFFFFFh	7FFFFFFFh	Yes	Signed 32-bit	Read Write

This object defines the positive software position limit in units of encoder counts. If the software limits are enabled and the actual position is out of range, then a software-limit fault occurs.

The term "positive" does not imply the value must be positive. Negative values are permitted; however, they should be a higher value than the negative software position limit.

Similar SmartMotor Commands: SLP=, RSLP

Object 2207h Encoder Modulo Limit

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2207h	000	Encoder Modulo Limit	0	FFFFFFFFh	FFFFFFFFh	No	unsigned 32-bit	Read Write

This object defines the encoder modulo limit in units of encoder counts.

An encoder will have some maximum value (modulo limit) before a roll-over of values. The modulo limit must be known by the motor to correctly interpret the incoming encoder data. Object 2207h supports this. The number is unsigned and based at 0. For example, an encoder with a resolution of 4096 will have this register configured with the value 4095, because that is the largest possible value (i.e., the value range is 0-4095 inclusive).

EXAMPLE:

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x2207      8711      'Object 2207h: External encoder follow max value
                             '(where encoder rolls over) i.e., 10 bit encoder
                             'would be 1023

. . .
fff=mmm                ' following motor's address.  In this demo, it is the network
                        ' master.  But you can make it a 3rd-party

. . .
' Set other objects in follow motor relating to follow mode.
SDOWR(fff,x2207,0,4,xffffffff)  GOSUB10  'Set encoder modulo limit

. . .
C10
  ' Code to check for CAN error and display it

. . .
RETURN
```

Object 2208h Encoder Follow Data

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2208h	000	Number of objects	3	3	3	No	unsigned 8-bit	Read Only
2208h	001	Encoder input value 8-bit signed or unsigned	0	FFh	0	Yes	unsigned 8-bit	Read Write
2208h	002	Encoder input value 16-bit signed or unsigned	0	FFFFh	0	Yes	unsigned 16-bit	Read Write
2208h	003	Encoder input value 32-bit signed or unsigned.	0	FFFFFFFFh	0	Yes	unsigned 32-bit	Read Write

This object is provided to accept data from a network (CANopen) based encoder. Three different data sizes are provided to handle PDO mapping to data sources of 8, 16, and 32 bits. Also, see object 2207h for configuring the resolution of this external encoder so that the SmartMotor knows when the encoder has rolled-over its number space.

- Subindex 0: Returns the number of subindex objects in this object
- Subindex 1: Encoder input value, 8-bit signed or unsigned
- Subindex 2: Encoder input value, 16-bit signed or unsigned
- Subindex 3: Encoder input value, 32-bit signed or unsigned

EXAMPLE:

Refer to the example program "CAN Bus - Time Sync Follow Encoder" in Chapter 3 of the *SmartMotor Developer's Guide*.

Object 2209h Encoder Follow Control

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2209h	000	Encoder follow control	0	FFFFh	0000h	Yes	unsigned 16-bit	Read Write

This object controls the behavior for the mode of following a network encoder and behavior of Object 220Ch MFA and Object 220Dh MFD. Refer to the following table.

Bit	Meaning
Bit 0	<p>Halt CANopen encoder follow/cam mode.</p> <p>0: Run/resume normally with the next increment, NOTE: Cam mode will restart from MCW command setting depending on bit 1.</p> <p>1: Mask the encoder increments. Setting to 1 will bring motors to a stop on the Sync packet event.</p> <p>See start/stop capability details below this table.</p>
Bit 1	<p>Control the resume of cam relative to bit 0.</p> <p>Firmware 5.0.4.16 or higher and 5.98.4.16 or higher:</p> <p>0: Cam is restarted on resume (bit 0).</p> <p>1: Resume of Cam mode from existing master location instead of restart.</p> <p>NOTE: Previous firmware does not support this and always restarts cam on resume (bit 0).</p>
Bit 2	<p>Ramp-up command MFA master or slave units. Object Object 220Ch MFA (not the serial command MFA)</p> <p>0: master</p> <p>1: slave</p>
Bit 3	<p>Ramp-down command MFD master or slave units. Object Object 220Dh MFD (not the serial command MFD)</p> <p>0: master</p> <p>1: slave</p>
Bit 4-15	Reserved. Write as 0.

Start/Stop Capability

For certain applications, object 2209h provides a start/stop capability used to maintain relative position offset between motors:

- **Start:** When starting in this mode with the G(2) command (object 2209h), the next encoder value received after this command will be the first master position. After that, the following encoder value received will be used to compute a difference from the first, and so on. This avoids a sudden jump in position when restarting after a stop (i.e., the firmware must ignore the master encoder while the motors are stopped).

- Stop: To ensure multiple following motors stop while remembering position offset relative to each other, the encoder data should be received through a single synchronous PDO to all motors. The control command object 2209h should be configured as a single synchronous PDO from the master that all motors receive at the same time using the same COB-ID. This allows the motors to receive the encoder data and control commands in a uniform way, relative to each other, when the sync packet arrives to all motors.

EXAMPLE:

Refer to the example program "CAN Bus - Time Sync Follow Encoder" in Chapter 3 of the *SmartMotor Developer's Guide*.

Object 220Ah MFMUL

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
220Ah	000	MFMUL (Mode Follow Multiplier)	-32767	32767	1	No	Signed 16-bit	Read Write

This object specifies the multiplier for external encoder mode follow with ratio MFMUL/MFDIV.

Both MFMUL and MFDIV may be positive or negative; this controls the resulting direction of shaft rotation.

For more details on MFMUL, see the *SmartMotor Developer's Guide*.

EXAMPLE:

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x220A    8714 ' Object 220Ah: External encoder follow MFMUL
. . .
fff=mmm        ' following motor's address. In this demo, it is the network
                ' master. But you can make it a 3rd-party
. . .
' Set other objects in follow motor relating to follow mode.
. . .
SDWR(fff,x220A,0,2,100)    GOSUB10 ' set MFMUL
. . .
C10
' Code to check for CAN error and display it
. . .
RETURN
```

Similar SmartMotor Commands: MFMUL=, RMFMUL, MFDIV

Object 220Bh MFDIV

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
220Bh	000	MFDIV (Mode Follow Divisor)	-32767 *	32767 *	1	No	Signed 16-bit	Read Write
* The value 0 is not accepted because a divide by 0 is not possible.								

This object specifies the divisor for external encoder mode follow with ratio MFMUL/MFDIV.

Both MFMUL and MFDIV may be positive or negative; this controls the resulting direction of shaft rotation.

For more details on MFDIV, see the *SmartMotor Developer's Guide*.

EXAMPLE:

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x220B    8715 ' Object 220Bh: External encoder follow MFDIV
. . .
fff=mmm          ' following motor's address.  In this demo, it is the network
                  ' master.  But you can make it a 3rd-party

. . .
' Set other objects in follow motor relating to follow mode.
. . .
SDOWR(fff,x220B,0,2,100)    GOSUB10 ' set MFDIV
. . .
C10
' Code to check for CAN error and display it
. . .
RETURN
```

Similar SmartMotor Commands: MFDIV=, RMFDIV, MFMUL

Object 220Ch MFA

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
220Ch	000	MFA (Mode Follow Ascend)	0	7FFFFFFh	0 (disabled)	No	Signed 32-bit	Read Write

This object sets the ascend ramp to the specified sync ratio from a ratio of zero.

For more details on MFA, see the *SmartMotor Developer's Guide*.

EXAMPLE:

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x220C    8716 ' Object 220Ch: External encoder follow MFA
. . .
fff=mmm        ' following motor's address.  In this demo, it is the network
                ' master.  But you can make it a 3rd-party

. . .
' Set other objects in follow motor relating to follow mode.
. . .
SDOWR(fff,x220C,0,4,20000)  GOSUB10 ' set MFA  control word x2209
                                ' determines if master or slave units.

. . .
C10
' Code to check for CAN error and display it
. . .
RETURN
```

Similar SmartMotor Commands: MFA, MFD

Object 220Dh MFD

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
220Ch	000	MFA (Mode Follow Descend)	0	7FFFFFFh	0 (disabled)	No	Signed 32-bit	Read Write

This object sets the descend ramp from the specified sync ratio to a ratio of zero.

For more details on MFD, see the *SmartMotor Developer's Guide*.

EXAMPLE:

```
'++++ HEX Coded Objects for CAN +++++
. . .
#define x220D 8717 ' Object 220Ch: External encoder follow MFD
. . .
' Set other objects in follow motor relating to follow mode.
. . .
fff=mmm          ' following motor's address. In this demo, it is the network
                  ' master. But you can make it a 3rd-party
. . .
SDOWR(fff,x220D,0,4,10000)  GOSUB10 ' set MFD control word x2209
                              ' determines if master or slave units.
. . .
C10
' Code to check for CAN error and display it
. . .
RETURN
```

Similar SmartMotor Commands: MFD, MFA

Object 2220h: 8-Bit Mappable Variables

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2220h	000	Number of Entries	04h	04h	04h	No	Unsigned 8-bit	Read Only
2220h	001	ab[0]	80h	7Fh	00h	Yes	Signed 8-bit	Read Write
2220h	002	ab[1]	80h	7Fh	00h	Yes	Signed 8-bit	Read Write
2220h	003	ab[2]	80h	7Fh	00h	Yes	Signed 8-bit	Read Write
2220h	004	ab[3]	80h	7Fh	00h	Yes	Signed 8-bit	Read Write

This object provides direct read or write access to user variables ab[0]–ab[3]. This object is provided to fill the need for PDO access to user variables. SDO access is also allowed. Also, see Object 2221h: 16-Bit Mappable Variables on page 154 and Object 2204h: Mappable 32-bit Variables on page 142.

For more details, see User Variables on page 45.

Object 2221h: 16-Bit Mappable Variables

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2221h	000	Number of Entries	04h	04h	04h	No	Unsigned 8-bit	Read Only
2221h	001	aw[32]	8000h	7FFFh	0000h	Yes	Signed 16-bit	Read Write
2221h	002	aw[33]	8000h	7FFFh	0000h	Yes	Signed 16-bit	Read Write
2221h	003	aw[34]	8000h	7FFFh	0000h	Yes	Signed 16-bit	Read Write
2221h	004	aw[35]	8000h	7FFFh	0000h	Yes	Signed 16-bit	Read Write

This object provides direct read or write access to user variables aw[32]–aw[35]. This object is provided to fill the need for PDO access to user variables. SDO access is also allowed. Also, see Object 2220h: 8-Bit Mappable Variables on page 153 and Object 2204h: Mappable 32-bit Variables on page 142.

For more details, see User Variables on page 45.

Object 2300h: Bus Voltage

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2300h	000	Bus Voltage	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only

This object reports the bus voltage (in millivolts) supplied to the motor drive stage.

Object 2301h: RMS Current

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2301h	000	RMS Current	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only

This object reports the RMS current (in milliamperes) of the motor windings.

Similar SmartMotor Commands: RUIA

Object 2302h: Internal Temperature

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2302h	000	Internal Temperature	00h	FFh		Yes	Unsigned 8-bit	Read Only

This object reports the SmartMotor's internal temperature in degrees C; the resolution is ± 1 degree C.

Similar SmartMotor Commands: RTEMP

Object 2303h: Internal Clock

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2303h	000	Internal Clock	00000000h	FFFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Write

This object represents the SmartMotor's internal clock in milliseconds. The value can be set as desired. This object is equivalent to the RCLK, =CLK, or CLK= commands (read or write), and it uses the same internal clock.

NOTE: This object is not the same as Object 2308h, which uses special clock-synchronization features that are only accessible through CANopen or serial interpolation. For details, see Object 2308h: Microsecond Clock on page 171.

Similar SmartMotor Commands: CLK=, RCLK

Object 2304h: Motor Status

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2304h	000	Number of Entries	00h	FFh	12h (18 dec)	No	Unsigned 8-bit	Read Only
2304h	001	Status Word 0	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	002	Status Word 1	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	003	Status Word 2	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	004	Status Word 3	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	005	Status Word 4	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	006	Status Word 5	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	007	Status Word 6	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	008	Status Word 7	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	009	Status Word 8	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	010	Status Word 9	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	011	Status Word 10	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	012	Status Word 11	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	013	Status Word 12	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	014	Status Word 13	0000h	FFFFh		Yes	Unsigned 16-bit	Read Write
2304h	015	Status Word 14	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	016	Status Word 15	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	017	Status Word 16	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only
2304h	018	Status Word 17	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only

This object reports the SmartMotor status words, which are equivalent to the RW(index) command. There is a special case where user status bits in status word 13 are writable through this object. This allows a host to cause user interrupts in a motor.

- Subindex 0 reports the number of status words (18)
- Subindex 1 reports SmartMotor status word 0
- Subindex 2 reports SmartMotor status word 1

- Subindex 3 reports SmartMotor status word 2
- Subindex 4 reports SmartMotor status word 3
- Subindex 5 reports SmartMotor status word 4
- Subindex 6 reports SmartMotor status word 5
- Subindex 7 reports SmartMotor status word 6
- Subindex 8 reports SmartMotor status word 7
- Subindex 9 reports SmartMotor status word 8
- Subindex 10 reserved
- Subindex 11 reports SmartMotor status word 10
- Subindex 12 reserved
- Subindex 13 reports SmartMotor status word 12
- Subindex 14 reports SmartMotor status word 13
- Subindexes 15–16 reserved
- Subindex 17 reports SmartMotor status word 16
- Subindex 18 reports SmartMotor status word 17

Status Word 0	Motion and motor health
0	Drive ready
1	Motor is off
2	Trajectory in progress
3	Bus voltage fault
4	Overcurrent occurred
5	Excessive temperature fault
6	Excessive position error fault
7	Velocity limit fault
8	Real-time temperature limit
9	Position error derivative fault
10	Right (+) limit enabled
11	Left (–) limit enabled
12	Historical right (+) limit
13	Historical left (–) limit
14	Right (+) limit asserted
15	Left (–) limit asserted

Status Word 1	Index registration and soft limits
0	Arming bit for rise capture of encoder 0
1	Arming bit for fall capture of encoder 0
2	Rising edge captured on encoder 0
3	Falling edge captured on encoder 0
4	Arming bit for rise capture of encoder 1
5	Arming bit for fall capture of encoder 1
6	Rising edge captured on encoder 1
7	Falling edge captured on encoder 1
8	Capture input state 0
9	Capture input state 1
10	Soft limits enabled
11	Soft limits behavior mode
12	Historical right soft limit
13	Historical left soft limit
14	Right soft limit
15	Left soft limit

Status Word 2	Communication state and program state
0	Com 0 error
1	Com 1 error
2	Reserved
3	Reserved
4	CAN error
5	Reserved
6	Ethernet error
7	IIC communications active
8	Reserved
9	Datablock checksum is bad (fault)
10	User program is running
11	Trace in progress
12	User EEPROM write buffer overflow
13	User EEPROM busy
14	Command error
15	Program checksum error

Status Word 3	PID, brake, move generation
0	Position error has exceeded soft threshold
1	Torque saturation
2	Voltage saturation
3	Wraparound occurred
4	KG enabled
5	Shaft direction
6	Torque direction
7	IO fault latch
8	Trajectory 1 relative position move
9	Reserved
10	Reserved
11	Modulo counter rollover
12	Brake asserted
13	Brake OK
14	Go on external input
15	Velocity reached or target ratio reached

Status Word 4	Timer status
0	Timer 0 running
1	Timer 1 running
2	Timer 2 running
3	Timer 3 running
4	Reserved
5	Reserved
6	Reserved
7	Reserved
8	Reserved
9	Reserved
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Reserved

Status Word 5	Interrupt enable status
0	Event 0 enabled
1	Event 1 enabled
2	Event 2 enabled
3	Event 3 enabled
4	Event 4 enabled
5	Event 5 enabled
6	Event 6 enabled
7	Event 7 enabled
8	Reserved
9	Reserved
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Events enabled

Status Word 6	Commutation status
0	Trapezoidal commutation
1	Enhanced trapezoidal commutation
2	Sinusoidal commutation
3	Current mode commutation
4	Reserved
5	Reserved
6	Reserved
7	Drive enable input fault
8	Electrical angle valid
9	TOB enabled (Torque overrun braking)
10	Invert direction enabled
11	MTB active
12	Encoder fault
13	Low bus voltage
14	High bus voltage
15	Reserved

Status Word 7	Multiple Trajectories
0	TG1 in progress
1	TG1 Accel/Ascend
2	TG1 Slewing
3	TG1 Decel/Descend
4	TG1 Reserved/Dwell
5	Reserved
6	Reserved
7	Reserved
8	TG2 in progress
9	TG2 Accel/Ascend
10	TG2 Slewing
11	TG2 Decel/Descend
12	TG2 Dwell (higher)
13	TG2 Traverse state
14	TG2 Lower dwell
15	TS Wait

Status Word 8	Cam/IP Mode user segment bits
0	Cam user bit 0
1	Cam user bit 1
2	Cam user bit 2
3	Cam user bit 3
4	Cam user bit 4
5	Cam user bit 5
6	Cam mode 0
7	Cam mode 1
8	IP user bit 0
9	IP user bit 1
10	IP user bit 2
11	IP user bit 3
12	IP user bit 4
13	IP user bit 5
14	IP mode 0
15	IP mode 1

Status Words 9	Reserved
-----------------------	-----------------

Status Word 10	RxPDO Arrival Notification
0	Master enabled
1	Rx PDO 1 arrived
2	Rx PDO 2 arrived
3	Rx PDO 3 arrived
4	Rx PDO 4 arrived
5	Rx PDO 5 arrived
6	Reserved
7	Reserved
8	Reserved
9	Reserved
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Reserved
<p>The user program should clear these status bits with a Z(10,bit) command, where bit is values 1–5, after the event handler part of the user program is executed. Bit 0 cannot be cleared—it is an indication of the master status, see Network Control Commands on page 83.</p> <p>NOTE: The ZS command will have no effect on these bits.</p>	

Status Word 11	Reserved
-----------------------	-----------------

Status Word 12	User-settable status bits (Read-only from this object)
0	User-settable bit 0
1	User-settable bit 1
2	User-settable bit 2
3	User-settable bit 3
4	User-settable bit 4
5	User-settable bit 5
6	User-settable bit 6

Status Word 12	User-settable status bits (Read-only from this object)
7	User-settable bit 7
8	User-settable bit 8
9	User-settable bit 9
10	User-settable bit 10
11	User-settable bit 11
12	User-settable bit 12
13	User-settable bit 13
14	User-settable bit 14
15	User-settable bit 15

Status Word 13	User-settable status bits (Writable from this object)
0	User-settable bit 16
1	User-settable bit 17
2	User-settable bit 18
3	User-settable bit 19
4	User-settable bit 20
5	User-settable bit 21
6	User-settable bit 22
7	User-settable bit 23
8	User-settable bit 24
9	User-settable bit 25
10	User-settable bit 26
11	User-settable bit 27
12	User-settable bit 28
13	User-settable bit 29
14	User-settable bit 30
15	User-settable bit 31

Status Words 14 and 15	Reserved
-----------------------------------	-----------------

Status Word 16	I/O: D-style: 0-6 (7 is a virtual bit) I/O: M-style: 0-10
-----------------------	--

Status Word 17	I/O: D-style only: AD1 I/O optional
-----------------------	--

Object 2305h: Motor Control

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2305h	000	Motor Control	0000h	FFFFh	0000h	Yes	Unsigned 16-bit	Read Write

This object provides access to certain SmartMotor commands. The value written to the object is a bit field; the corresponding functions are called when the value of a bit is changed. The function is not repeated if the bit value stays the same. The value read from this object is the most recent value written to this object — it is *not* an indication of the motor's current state.

NOTE: This object may be difficult to use; consider using object 2309h instead. This command may be removed in future versions.

Bit	Function
0	Software limit enable: <ul style="list-style-type: none"> • Transition 0 to 1: SLE command (enable software limits) • Transition 1 to 0: SLD command (disable software limits)
1	Program control: <ul style="list-style-type: none"> • Transition 0 to 1: RUN command. • Transition 1 to 0: END command.
2–15	Reserved (set to 0).

Object 2306h: Motor Subroutine Index

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2306h	000	Motor Subroutine Index	-1	999	-1	Yes	Signed 16-bit	Read Write

Each time this object is written, it calls the specified subroutine. Therefore, care must be taken to ensure the subroutine has completed before calling it again. The value read from this object is the most recent value written to this object — it is *not* an indication of the motor's current state.

For more details, see Calling Subroutines on page 47.

Value written	Function
-1	No operation
0-999	GOSUB(value)

Object 2306h has the following limitations:

- Each time it is accessed by a repeated PDO, it will call a subroutine. Therefore, this object is not useful for PLCs or other hosts that send repeated data.
- There is no mechanism provided to indicate that the subroutine has completed. Therefore, the progress of the subroutine cannot be monitored to know when it is finished and ready to call again.

Object 2309h fixes these two limitations and provides additional features. For details, see Object 2309h: GOSUB R2 on page 172.

Object 2307h: Sample Period

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2307h	000	Sample Period	0000h	FFFFh	12500	Yes	Unsigned 16-bit	Read Only

This object reports the SmartMotor sample period in microseconds*100. This is the time period for the PID cycle and trajectory update.

PID mode	Reported from object 2307	Time (microseconds)
1	6250	62.5
2	12500	125.0
4	25000	250.0
8	50000	500.0

Similar SmartMotor Commands: RSP (PID rate info), RSAMP

Object 2308h: Microsecond Clock

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2308h	000	Microsecond Clock	00000000h	FFFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Write

This object represents an internal sync clock in microseconds. Writing to this object can interfere with the time synchronization process used with Interpolation mode. Reading this object provides a value that is only current with the most recent PID cycle.

NOTE: This object is tied to special clock-synchronization features that are only accessible through CANopen or serial interpolation. There is no SmartMotor command equivalent — it is *not* associated with the SmartMotor CLK-type commands, which use a different physical clock that operates independently. Therefore, this object is not the same as Object 2303h. For details, see Object 2303h: Internal Clock on page 158.

Object 2309h: GOSUB R2

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2309h	000	GOSUB R2	-9	+999	-1	Yes	Signed 16-bit	Read Write

This version of GOSUB will only take action when the value written is different from previous values written to this object.

This GOSUB will not nest subroutine calls through this object (other sources of GOSUB may still nest) If there is already an active subroutine that was called through this object, further calls are ignored without buffering.

The following table describes the possible values:

Value	Description
0-999	Corresponds to GOSUB(0) through GOSUB(999). An SDO error is issued if a previous GOSUB called from this object is still busy.
-1	Do nothing. This is useful as a null value since a transition must be made for a new GOSUB call.
-2	END
-3	RUN
-4	EILP
-5	EILN
-6	SLE
-7	SLD
-8	SLM(0)
-9	SLM(1)
-10	Freewheel when the drive is turned off. However, the configured fault reaction will be in effect and will take priority if a fault is present.

Similar SmartMotor Commands: GOSUB, END, RUN, EILP, EILN, SLE, SLD, SLM()

Object 2400h: Interpolation Mode Status

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2400h	000	Interpolation Mode Status	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only

This object provides additional information relevant to Interpolation mode.

Bit	Function
0–5	Number of free record buffer locations
6	Position error tolerance exceeded
7	PLL locked to sync status (firmware 5.0.4.49 / 5.98.0.49 or later)
8	IP mode pending
9	IP mode ready
10	Invalid time units error
11	Invalid position increment error
12	Drive ready
13	FIFO overflow
14	FIFO underflow
15	IP mode running

Object 2401h: Buffer Control

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2401h	000	Buffer Control	0000h	FFFFh	0000h	Yes	Unsigned 16-bit	Read Write

This object provides a special way of controlling the interpolation buffer level when the host cannot monitor the buffer level and/or time synchronization is not possible. The value written is a proportional response to how far the interpolation is from the target buffer level. That level is set using the Buffer Setpoint object (2402h). For details, see Object 2402h: Buffer Setpoint on page 175.

As the buffer empties, the interpolation rate slightly decreases; as the buffer fills, the interpolation rate slightly increases. A typical value to write is 10000.

Note that this is not an ideal way to control the buffer level for the following reasons:

- The buffers of different motors will not perfectly align, so the motion will not be perfectly synchronized.
- The host must send the data to the motor at an even time spacing. However, some hosts may fill the buffer in bursts of activity — that will not work with the SmartMotor.

Object 2402h: Buffer Setpoint

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2402h	000	Buffer Setpoint	00h	FFh	14h	No	Unsigned 8-bit	Read Write

This object specifies the target buffer level. It is used in conjunction with the Buffer Control object (2401h) to maintain the buffer at that level. For details, see Object 2401h: Buffer Control on page 174.

Object 2403h: Interpolation User Bits

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2403h	000	Interpolation User Bits	00h	3Fh	00h	Yes	Unsigned 8-bit	Read Write

These bits are captured from this register when a new interpolation record is written. When the interpolation data is consumed by Interpolation mode, these bits will be reported in the status word (object 2304h, subindex 9) along with the corresponding data record. Those user bits will be displayed in the segment between the previous point and the current point.

In the following example, the user bit will be visible in the status word (object 2304h, subindex 9) between points 3000 and 4000.

1. Set the Interpolation User Bits object (2403h) to the value 0.
2. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 2000
 - b. 3000
3. Set the Interpolation User Bits object (2403h) to the value 1.
4. Put data in buffer by writing the value 4000 to subindex 1 of the Interpolation Data Record object (60C1h).
5. Set the Interpolation User Bits object (2403h) to the value 0.
6. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. Write the value 5000 to object 60C1h, subindex 1.
 - b. Write the value 6000 to object 60C1h, subindex 1.

Object 2404h: Interpolation Sample Clock

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2404h	000	Interpolation Sample Clock	00000000h	FFFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Only

This object reports the 32-bit unsigned count of motor PID sample periods since the start of interpolation.

Object 2500h: Encapsulated SmartMotor Command

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2500h	000	Number of Entries	03h	03h	03h	No	Unsigned 8-bit	Read Only
2500h	001	Command String				No	String: 32 bytes	Read Write
2500h	002	Command Response				No	String: 32 bytes	Read Only
2500h	003	Command Status	00h	FFh	00h	No	Unsigned 8-bit	Read Only

This object provides an interface to the SmartMotor command language. There is a 32-character limit for the command string and for the response string. For details, see Command Interface (Object 2500h) on page 48.

Drive and Motion Control Profile

This section describes the objects in the Drive and Motion Control Profile. This set of objects in the range 6000h to 67FFh implement the CiA 402 motion profile. This provides access to common commands for controlling the motor.

Object 6040h: Control Word	181
Object 6041h: Status Word	183
Object 605Ah: Quick Stop Option Code	184
Object 605Dh: Halt Option Code	185
Object 605Eh: Fault Reaction Option Code	186
Object 6060h: Modes of Operation	187
Object 6061h: Modes of Operation Display	189
Object 6062h: Position Demand Value	190
Object 6063h: Position Actual Internal Value	191
Object 6064h: Position Actual Value	192
Object 6065h: Following Error Window	193
Object 606Bh: Velocity Demand Value	194
Object 606Ch: Velocity Actual Value	195
Object 6071h: Target Torque	196
Object 6074h: Torque Demand Value	197
Object 6077h: Torque Actual	198
Object 6079h: DC Link Circuit Voltage	199
Object 607Ah: Target Position	200
Object 607Ch: Home Offset	201
Object 6080h: Max Motor Speed	203
Object 6081h: Profile Velocity in PP Mode	204
Object 6083h: Profile Acceleration	205
Object 6084h: Profile Deceleration	206
Object 6085h: Quick Stop Deceleration	207
Object 6087h: Torque Slope	208
Object 608Fh: Position Encoder Resolution	209

Object 6098h: Homing Method	210
Object 6099h: Homing Speeds	213
Object 609Ah: Homing Acceleration	214
Object 60C0h: Interpolation Sub-Mode Select	215
Object 60C1h: Interpolation Data Record	216
Object 60C2h: Interpolation Time Period	217
Object 60C4h: Interpolation Data Configuration	219
Object 60F4h: Following Error Actual Value	220
Object 60FBh: Position Control Parameter Set	221
Object 60FCh: Position Demand Internal Value	223
Object Description	223
Entry Description	223
Object 60FDh: Digital Inputs	224
Object 60FEh: Digital Outputs	226
Object 60FFh: Target Velocity	228
Object 6402h: Motor Type	229
Object 6502h: Supported Drive Modes	230
Object 67FFh: Single Device Type	231

Object 6040h: Control Word

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6040h	000	Control Word	0000h	FFFFh	0000h	Yes	Unsigned 16-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

The control word is the primary method of commanding motion in the SmartMotor. The following features are accessed with this object:

- Enable or disable the motor drive
- Quick stop function
- Halt function
- New position setpoint in Profile Position mode (PP)
- Start motion: Profile Position (PP), Profile Velocity (PV), Torque (TQ), Interpolation (IP), and Homing (HM)

For more details, see Control Words, Status Words and the Drive State Machine on page 51.

The SmartMotor =CAN and RCAN commands can be used to assign/report the value of the NMT state, control word (object 6040h) and status word (object 6041h). For details, see =CAN, RCAN on page 80.

The following table provides a listing of the available bits, their names and descriptions.

Bit	Name	Description
0	Switch on	These bits control the CiA 402 profile drive state machine. For more details, see CiA 402 Profile Motion State Machine on page 51.
1	Enable voltage	
2	Quick stop	
3	Enable operation	
4	Operation mode specific: "New setpoint"	Used by PP, HM, and IP modes. In PP mode: all positions must be set with a rising transition of this bit. In IP mode: rising edge of this bit is used to initially start operation but not required at each data point.
5	Operation mode specific: "Change set immediately"	Used in PP mode; other modes can leave as 0.
6	Operation mode specific: "Relative"	In PP mode, this sets a position relative target (PRT=) instead of a position target (PT=) type of move.
7	Fault reset	Rising transition resets fault in all modes of operation. If the fault condition still exists (status word object 6041h), then the cause has not been cleared.
8	Halt	If this bit is set, then the motor will stop from any mode of operation. The action taken is set in advance by the halt option code.
9	Operation mode specific	Used in PP mode; other modes can leave as 0.
10	Reserved	Reserved by the CiA 402 specification.
11	Manufacturer-specific: Reserved for user application	Reserved for the user's application. This bit is visible in a program through RCAN(2).
12	Manufacturer-specific	Do not use; leave at 0.
13	Manufacturer-specific	Do not use; leave at 0.
14	Manufacturer-specific	Do not use; leave at 0.
15	Manufacturer-specific: Reset interpolation buffer	Used to reset the IP mode buffer.

Object 6041h: Status Word

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6041h	000	Status Word	0000h	FFFFh		Yes	Unsigned 16-bit	Read Only

This object indicates the current state of the drive. For more details, see Control Words, Status Words and the Drive State Machine on page 51. The SmartMotor =CAN and RCAN commands can be used to assign/report the value of the NMT state, control word (object 6040h) and status word (object 6041h). For details, see =CAN, RCAN on page 80.

Bit	Name	Description
0	Ready to switch on	The bits 0–3, 5 and 6 represent the state of the CiA 402 profile drive state machine. For more details, see Control Words, Status Words and the Drive State Machine on page 51.
1	Switched on	
2	Operation enabled	
3	Fault	
4	Voltage enabled	Sufficient voltage is present to operate the motor.
5	Quick stop	The bits 0–3, 5 and 6 represent the state of the CiA 402 profile drive state machine. For more details, see Control Words, Status Words and the Drive State Machine on page 51.
6	Switch on disabled	
7	Warning	Not used (reports as 0).
8	Manufacturer-specific	Used by the GOSUB R2 object (2309h) to indicate the sub-routine is busy.
9	Remote	Controlled through CANCTL(13,x). This bit indicates if the motor is accepting commands from the CANopen network. Default is 1, which indicates the motor is accepting commands.
10	Target reached	"Target reached" — this is operation-mode specific. It indicates the speed, position, or torque profile was achieved. In Homing (HM) mode, the motor has come to rest after finding the home position. However, the motor is not specifically at the home position because a deceleration distance was required after finding the position.
11	Internal limit active	"Limit" — set if a position limit is currently showing a fault.
12	Operation mode specific	"Setpoint acknowledgment" — this is operation-mode specific to PP, IP and PV modes. It indicates a new setpoint was received. In Homing (HM) mode, the homing process has found the home position, and the "position actual" has been adjusted to the new home position and home offset.
13	Operation mode specific	"Move error" — set if a position error occurred.
14	Manufacturer-specific	User-controlled bit through CANCTL(12,x).
15	Manufacturer-specific	Not used (reports as 0).

Object 605Ah: Quick Stop Option Code

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
605Ah	000	Quick Stop Option Code	-1	2	2	No	Signed 16-bit	Read Write

This object determines what action should be taken if the quick stop function is active. That function is activated by bit 2 of the Control Word object (6040h). For details, see Object 6040h: Control Word on page 181.

In Profile Torque (TQ) mode, quick stop option code values 1 and 2 will reduce the torque according to the torque slope rate because this is not a servo mode that can follow the deceleration or quick-stop deceleration rates.

Value	Function
-1	MTB (drive turned off, resists rotation)
0	Disable drive (drive turned off, free to rotate)
1	Decelerate on the profile deceleration ramp (see Object 6084h: Profile Deceleration on page 206); drive will automatically leave the quick stop state.
2	Decelerate on the quick stop ramp (see Object 6085h: Quick Stop Deceleration on page 207); drive will automatically leave the quick stop state
3-8	Not supported
9-32767	Reserved

If using Follow or Cam mode, be aware that these decelerations are not applied. The Animatics MFD() command controls the deceleration in those cases.

Object 605Dh: Halt Option Code

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
605Dh	000	Halt Option Code	1	2	1	No	Signed 16-bit	Read Write

This object determines what action should be taken if the halt bit (bit 8) is set in Control Word object (6040h). For details, see Object 6040h: Control Word on page 181.

In Profile Torque (TQ) mode, halt option code values 1 and 2 will reduce the torque according to the torque slope rate because this is not a servo mode that can follow the deceleration or quick-stop deceleration rates.

Value	Function
0	Reserved
1	(Default) Decelerate on the profile deceleration ramp (see Object 6084h: Profile Deceleration on page 206)
2	Slow down on quick-stop ramp
3-4	Not supported
5-32767	Reserved

If using Follow or Cam mode, be aware that these decelerations are not applied. The Animatics MFD() command controls the deceleration in those cases.

Object 605Eh: Fault Reaction Option Code

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
605Eh	000	Fault Reaction Option Code	-1	1*	-1	Yes	Signed 16-bit	Read Write

This object determines what action should be taken if a fault occurs in the motor. Causes of a fault include: limit switches, software limits, overtemperature, excessive position error, etc.

In Profile Torque (TQ) mode, fault reaction option code value 1 will reduce the torque according to the torque slope rate because this is not a servo mode that can follow the deceleration rate.

Value	Function
-1	(Default) MTB (drive turned off, resists rotation)
0	Disable drive (drive turned off, free to rotate)
1	Decelerate on the profile deceleration ramp (see Object 6084h: Profile Deceleration on page 206)
2	Not supported
3	Not supported
4	Not supported
5-32767	Reserved

If using Follow or Cam mode, be aware that these decelerations are not applied. The Animatics MFD() command controls the deceleration in those cases.

Similar SmartMotor Commands: FSA()

Object 6060h: Modes of Operation

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6060h	000	Modes of Operation	-12**	8**	0	Yes	Signed 8-bit	Read Write*

*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.

**The value 0 is allowed and will not return error (but it does not enter a mode of motion.) . Class 5 firmware doesn't necessarily return a range error. Anything not supported in the table below may be silently ignored. Class 5 supported mode 8 (CSP) as of firmware version 5.0.4.46 / 5.98.4.46 or later.

The type of motion control is selected by setting this object to one of the values shown in the following table. The new setting will take effect immediately. When transitioning to Interpolated Position (IP) mode or Profile Position (PP) mode, the motor will stop, there must be a rising transition on bit 4 of the control word and then motion will begin in the new mode.

The value read back from this object does not indicate the current mode of operation; it is only an indication of what was written previously and not an indication of the motor's current state. Use the Modes of Operation Display object (6061h) to see the currently active mode. For details, see Object 6061h: Modes of Operation Display on page 189.

Value -11 or -12 should be used as the mode of operation where follow or cam mode is accepting data from the CANopen data object *and* position profile mode is active (dual trajectory).

Value	Motion Control Mode
-12	Cam from CANopen encoder + position mode ^{1,2}
-11	Follow CANopen encoder + position mode
-10 to -4	Reserved / not supported
-3	Step and direction input
-2	Follow quadrature encoder input
-1	Reserved / not supported
0	Null (not an error, but not a mode of motion either.)
1	Profile Position (PP) mode
2	Reserved / not supported
3	Profile Velocity (PV) mode
4	Torque Profile (TQ) mode
5	Reserved / not supported
6	Homing (HM) mode
7	Interpolated Position (IP) mode ³
8	Cyclic Sync Position (CSP) mode (as of firmware version 5.0.4.46 / 5.98.4.46 or later)
9 to 10	Reserved / not supported

Value	Motion Control Mode
11 to 127	Reserved / not supported
<ol style="list-style-type: none">1. Cam mode operation requires user program to configure cam.2. Cam mode still uses MFA, MFD, MFMUL, MFDIV to control encoder input into cam master. See the <i>SmartMotor Developer's Guide</i> for details.3. This mode is <u>not</u> supported in the standard release; consult Moog Animatics for further information.	

Similar SmartMotor Commands: MV, MP, MT, MFR, MSR

Object 6061h: Modes of Operation Display

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6061h	000	Modes of Operation Display	80h	7Fh	00h	Yes	Signed 8-bit	Read Only

Displays the current mode of motion control; refer to Object 6060h: Modes of Operation on page 187.

Similar SmartMotor Commands: RMODE

Object 6062h: Position Demand Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6062h	000	Position Demand Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the position calculated by the motion profile; it takes into account the acceleration and velocity targets. Because user units are not supported, the value is in units of encoder counts, which are the same units as those for object 60FCh. For details, see Object 60FCh: Position Demand Internal Value on page 223.

When the motor drive is inactive or in torque mode, the value reported is simply the current position.

Similar SmartMotor Commands: RPC

Object 6063h: Position Actual Internal Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6063h	000	Position Actual Internal Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the current position of the motor shaft in units of encoder counts.

Similar SmartMotor Commands: RPA

Object 6064h: Position Actual Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6064h	000	Position Actual Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the current position of the motor shaft in units of encoder counts. Because user units are not supported, the value is in units of encoder counts, which are the same units as those for object 6063h. For details, see Object 6063h: Position Actual Internal Value on page 191.

Similar SmartMotor Commands: RPA

Object 6065h: Following Error Window

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6065h	000	Following Error Window	00000000h	0003FFFFh*	000003E8h	Yes	Unsigned 32-bit	Read Write
*The value -1 is allowed. Class 5 firmware doesn't necessarily return a range error, the high limit shown is what will be accepted by the firmware. Anything higher may be silently ignored.								

This object defines the range of tolerated deviation for the actual position relative to the calculated demand position. If the actual position is out of range, a following-error fault occurs and the drive will react according to the fault reaction. The units of this object are in encoder counts.

Similar SmartMotor Commands: EL=, REL

Object 606Bh: Velocity Demand Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
606Bh	000	Velocity Demand Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the velocity calculated by the motion profile; it takes into account acceleration and velocity targets. The units are: (encoder counts per sample period) * 65536.

Similar SmartMotor Commands: RVC

Object 606Ch: Velocity Actual Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
606Ch	000	Velocity Actual Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the actual velocity of the motor shaft. The units are: (encoder counts per sample period) * 65536.

Similar SmartMotor Commands: RVA

Object 6071h: Target Torque

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6071h	000	Target Torque	-1000	1000	0000h	Yes	Signed 16-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object is the target value for the motor when operating in Profile Torque (TQ) mode. The value written will be reached at a rate specified by the Torque Slope object (6087h). When the Control Word object (6040h) has enabled motion, the value written here will be accepted immediately. The units of this value are per thousand of the motor's rated torque.

A value of 1000 in this object is equivalent to $T=32767$ in the corresponding SmartMotor command. In other words, DS402 considers 1000 to be full-scale torque, whereas 32767 is considered to be full-scale torque for the SmartMotor serial commands.

Similar SmartMotor Commands: T=, RT

Object 6074h: Torque Demand Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6074h	000	Torque Demand Value	8000h	7FFFh	0000h	Yes	Signed 16-bit	Read Only

This object provides the motor's demand torque from the PID when in Position (PP), Velocity (PV) or interpolation (IP) mode, or the torque profile when in Torque (TQ) mode. The units of this value are per thousand of the motor's rated torque.

NOTE: This object represents the requested value from the Torque profile (in TQ mode) or the PID (in all other closed-loop servo modes). However, due to current limits, torque profile, etc., the motor may not be able to deliver the requested torque.

A value of 1000 in this object is equivalent to T=32767 in the corresponding SmartMotor command. In other words, DS402 considers 1000 to be full-scale torque, whereas 32767 is considered to be full-scale torque for the SmartMotor serial commands.

Object 6077h: Torque Actual

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6077h	000	Torque Actual	8000h	7FFFh	0000h	Yes	Signed 16-bit	Read Only

This object reports the actual torque based on measured current. The value is reported in units per thousand of rated torque.

NOTE: This object attempts to report the actual measured torque based on the current in the motor windings. However, not all SmartMotor modes of commutation can successfully measure current-producing torque. Therefore, this command is only valid in the M-style motor while in MDC commutation mode. Other modes will report the same data as object 6074h. For details, see Object 6074h: Torque Demand Value on page 197.

A value of 1000 in this object is equivalent to T=32767 in the corresponding SmartMotor command. In other words, DS402 considers 1000 to be full-scale torque, whereas 32767 is considered to be full-scale torque for the SmartMotor serial commands.

Object 6079h: DC Link Circuit Voltage

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6079h	000	DC Link Circuit Voltage	00000000h	FFFFFFFFh		Yes	Unsigned 32-bit	Read Only

This object describes the supplied voltage, in millivolts, measured at the motor's power inverter.

Similar SmartMotor Commands: RUJA

Object 607Ah: Target Position

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
607Ah	000	Target Position	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write

This object specifies the target position that the motor should move to in Profile Position (PP) mode. The units of this object are in encoder counts. When the "relative" bit (bit 6) of the Control Word object (6040h) is set, the value written is added to the position currently demanded.

The target position will be approached according to the Profile Acceleration object (6083h), Profile Deceleration object (6084h), and Profile Velocity object (6081h).

This object is not immediately accepted when written. It is only accepted when the "New setpoint" bit (bit 4) of the Control Word object (6040h) has a rising transition.

Similar SmartMotor Commands: PT=, PRT=, RPT, RPRT

Object 607Ch: Home Offset

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
607Ch	000	Home Offset	80000000h	7FFFFFFFh	0	Yes	Signed 32-bit	Read Write

This object shifts the origin of the actual position when the Homing (HM) mode is executed. When HM mode is commanded to begin, the home position is first discovered. The home position is the physical location of the switch or index per the specific homing method. Once found, that physical location is assigned the negative of the home offset value:

$$\text{Home position} = -\text{Home offset}$$

The home position is assigned with -home offset. See the following example.

Homing offset object 607Ch = +600

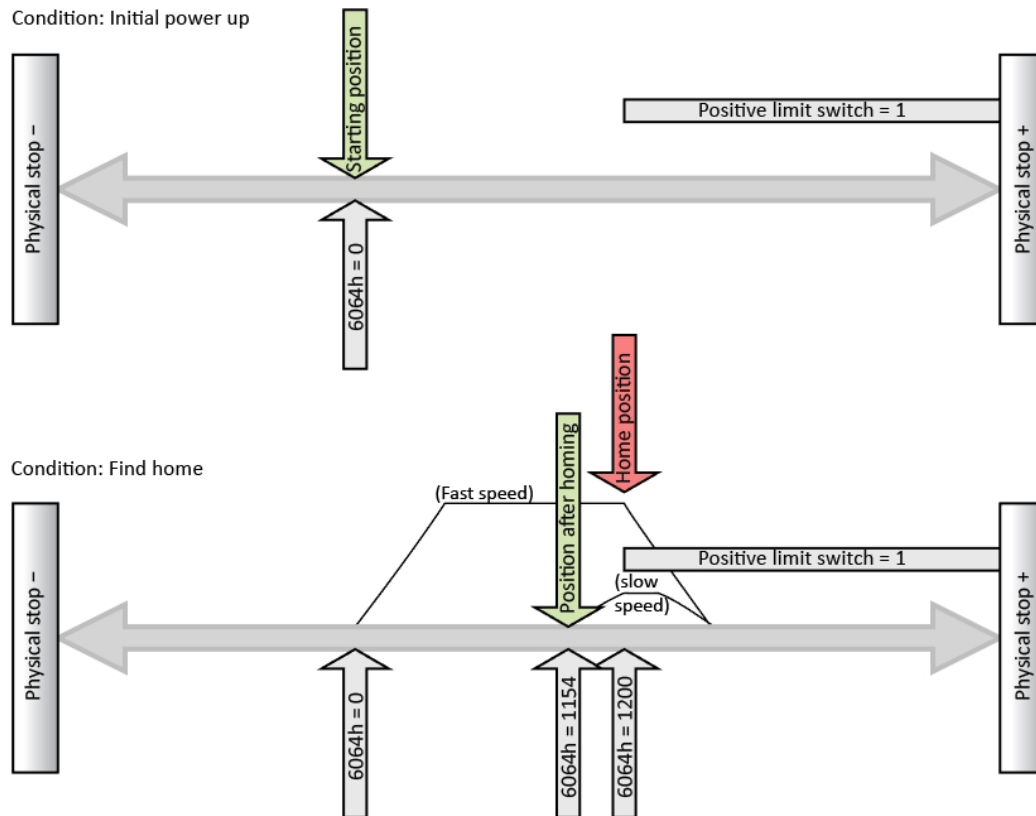
Green: machine physical position

Red: Home Position — where the sensors say it is

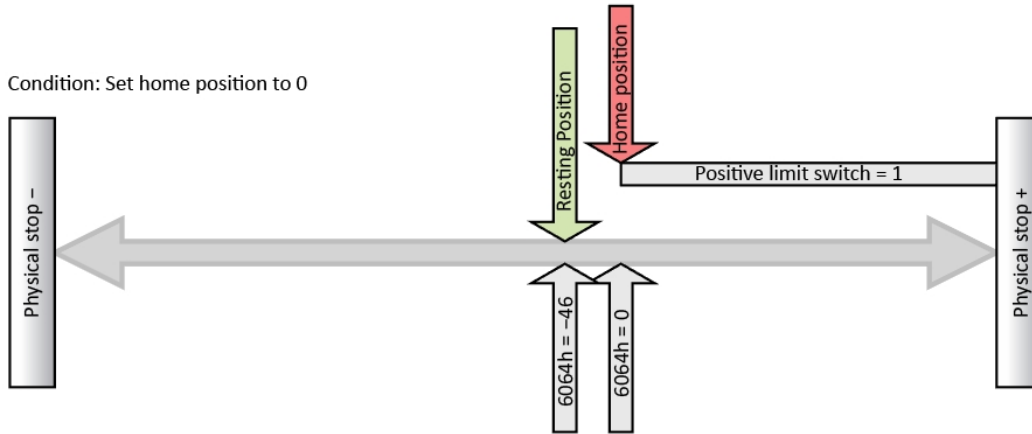
Blue: Zero Position — after homing completes, where the machine reports 6064h = 0

Homing method = 18

Incremental encoder (powers up at value = 0)



Condition: Set home position to 0

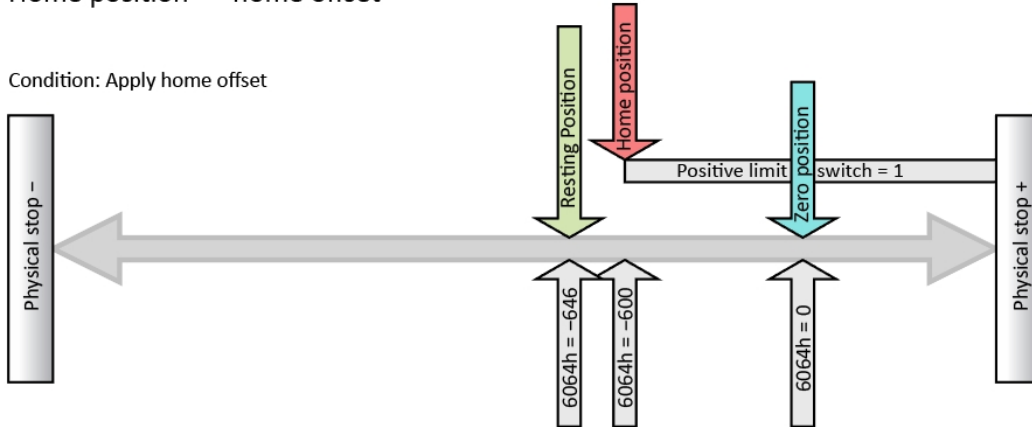


CiA 402 and ETG guidelines state: "Zero position = home position + home offset"

$0 = \text{Home position} + \text{home offset}$

$\text{Home position} = -\text{home offset}$

Condition: Apply home offset



Object 6080h: Max Motor Speed

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6080h	000	Max Motor Speed	00000000h	FFFFFFFFh	Set according to factory settings	Yes	Unsigned 32-bit	Read Write

This object specifies the speed limit for the motor in either direction. The units are in revolutions per minute (rpm). If this value is exceeded, the motor will enter a fault condition.

The value is specific to each SmartMotor model. For details, see the *Moog Animatics Product Catalog*, which is available on the Moog Animatics website.

Similar SmartMotor Commands: VL=, RVL

Object 6081h: Profile Velocity in PP Mode

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6081h	000	Profile Velocity in PP Mode	00000000h	7FFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object only applies to Profile Position (PP) mode. The position profile will accelerate to this speed and remain at this speed until deceleration begins for approach of the position target. The units are: (encoder counts per sample period) * 65536.

Also, refer to Object 60FFh: Target Velocity on page 228.

Similar SmartMotor Commands: VT= (NOTE: The value written to 6081h does not appear when reading back VT.)

Object 6083h: Profile Acceleration

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6083h	000	Profile Acceleration	00000000h	7FFFFFFFh	00000004h	Yes	Unsigned 32-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object is the acceleration in the Profile Velocity (PV) mode and the Profile Position (PP) mode. The units are: (encoder counts per (sample²)) * 65536.

Similar SmartMotor Commands: AT=, ADT=, RAT

Object 6084h: Profile Deceleration

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6084h	000	Profile Deceleration	00000000h	7FFFFFFFh	00000004h	Yes	Unsigned 32-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object is the deceleration in the Profile Velocity (PV) mode and the Profile Position (PP) mode. The units are: (encoder counts per (sample²)) * 65536.

Similar SmartMotor Commands: DT=, ADT=, RDT

Object 6085h: Quick Stop Deceleration

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6085h	000	Quick Stop Deceleration	00000000h	7FFFFFFFh	7FFFFFFFh	Yes	Unsigned 32-bit	Read Write

This object is used to stop the drive with the quick stop function, which is commanded from bit 2 of the Control Word object (6040h). The value is the deceleration used to stop the motor if the quick stop command is given and the Quick Stop Option Code object (605Ah) is set to 2. The units are: (encoder counts per (sample²)) * 65536.

For additional details, see Object 6040h: Control Word on page 181 and Object 605Ah: Quick Stop Option Code on page 184.

Object 6087h: Torque Slope

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6087h	000	Torque Slope	00000000h	FFFFFFFFh	007A12F4h	Yes	Unsigned 32-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object is the torque mode acceleration/deceleration slope. The units are in torque units per second. To put this into context, a value of 1000 in this object can ramp the SmartMotor to full torque in one second.

In SmartMotor commands, the corresponding command is TS=, where the units are different. In the TS= command, the units are: ("T=" per sample)*65536. Therefore, a value of 1000 in this object is equivalent to TS=268427, assuming the default PID rate of 8000 Hz.

For related information, see Object 6071h: Target Torque on page 196.

Similar SmartMotor Commands: TS=, RTS

Object 608Fh: Position Encoder Resolution

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
608Fh	000	Number of Entries	02h	02h	02h	No	Unsigned 8-bit	Read Only
608Fh	001	Encoder Counts	00000000h	FFFFFFFFh	Encoder resolution.	Yes	Unsigned 32-bit	Read Only
608Fh	002	Motor Revolutions	00000000h	FFFFFFFFh	00000001h	Yes	Unsigned 32-bit	Read Only

This object defines the resolution of the encoder. There are two subindex objects that describe the encoder resolution — subindex 001: Encoder Counts and subindex 002: Motor Revolutions. To determine the encoder resolution (number of encoder counts per motor revolution), divide the value of subindex 1 by the value of subindex 2. The units are in encoder counts.

Object 6098h: Homing Method

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6098h	000	Homing Method	80h	7Fh	0	Yes	Signed 8-bit	Read Write

This object selects the method used in Homing (HM) mode. This must be set before starting a homing process, and it should not be changed while HM mode is actively seeking home.

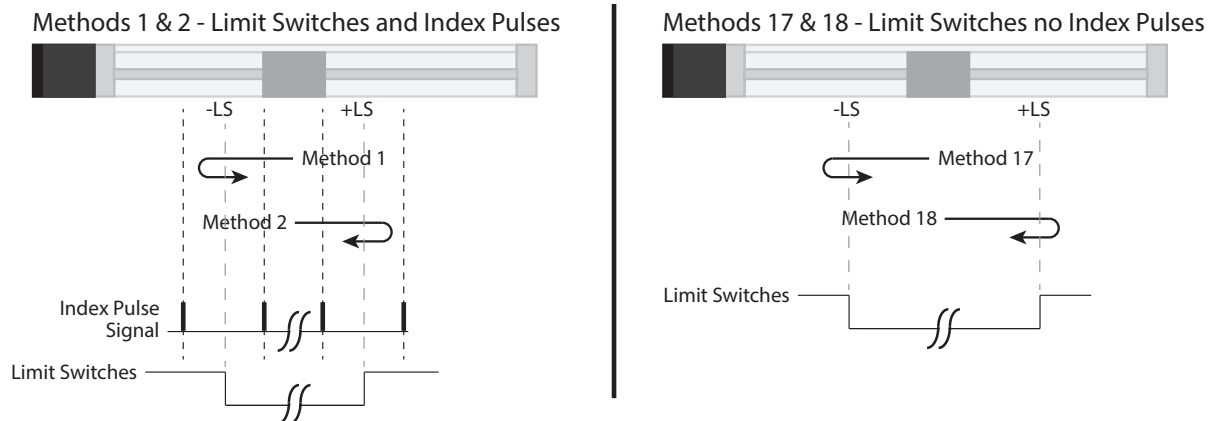
NOTE: The homing input is I/O 6. For more details on I/O, consult the *Class 5 SmartMotor™ Installation and Startup Guide* and the *SmartMotor™ Developer's Guide*.

Homing Method Value	Description
1, 2	Home position is the first index in the positive direction from the negative limit switch (1), or in the negative direction from the positive limit switch (2) (requires that limit switches are enabled).
17, 18	Home position is at the negative limit switch (17) or at the positive limit switch (18) (requires that limit switches are enabled).
33, 34	Home position is the location of the first index in the negative direction (33) or positive direction (34) from the current position.
35	Accept the current position as the home position. (current position = -home offset)

NOTE: Methods 1, 2, 33 and 34 make use of the index of the internal encoder, which provides a precise location (switches may have some position uncertainty). The construction of the machine should consider the proximity of the index mark to the switch threshold. The index location should be at 180 degrees rotation of the encoder (RRES/2) from the switch threshold. This will ensure that the index mark does not fall within the uncertainty of the switch transition.

NOTE: Methods 1, 2, 17 and 18 make use of the limit switches. Limit switches must be enabled and physically wired to the motor. Under these methods, the homing process will not start if the relevant limit has been disabled.

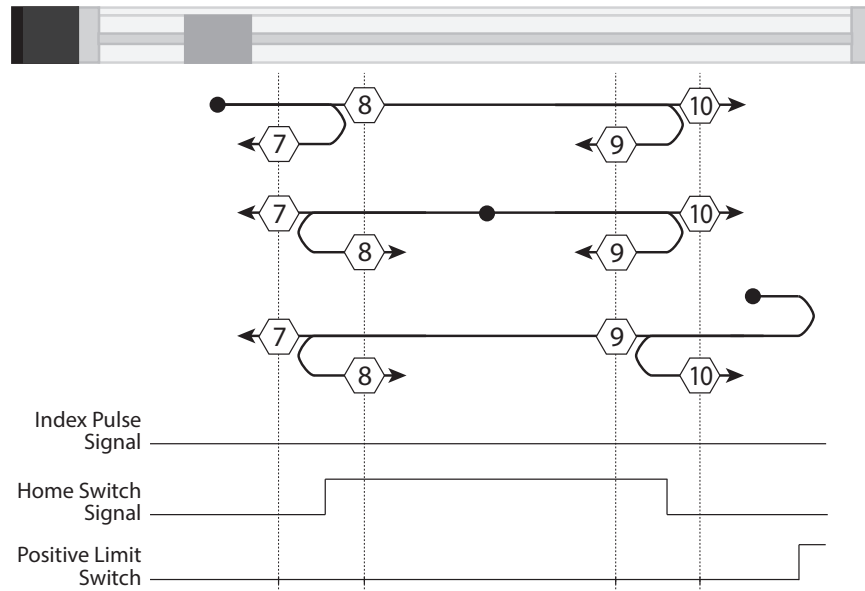
The following figures illustrate the differences between the methods that use an index pulse and those that do not. For example, methods 1 and 2 use an index pulse signal, while methods 17 and 18 do not.



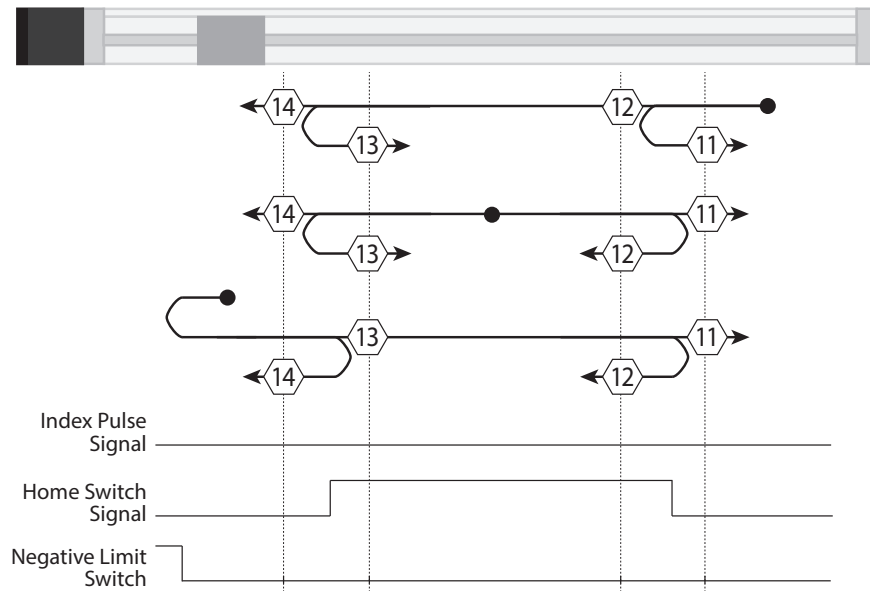
The following figures illustrate homing methods 7-14. Note the following:

- the number in the hexagon is the selected homing mode
- the solid circle is the location of the motor when homing mode started, each possibility is shown

Methods 7-10: Positive Initial Motion



Methods 11-14: Negative Initial Motion



Object 6099h: Homing Speeds

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6099h	000	Number of Entries	02h	02h	02h	No	Unsigned 8-bit	Read Only
6099h	001	Speed during search for switch	00000000h	7FFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Write
6099h	002	Speed during search for zero	00000000h	7FFFFFFFh	00000000h	Yes	Unsigned 32-bit	Read Write

This object only applies to Homing (HM) mode. The homing profile will accelerate to these speeds depending on the segment of the homing routine that is in use.

In general, the "speed during search for switch" segment is expected to be faster than the "speed during search for zero" segment. The "speed during search for zero" segment is selected when the homing mode expects to find the home position with the move it is currently starting. If the homing mode expects an intermediate switch event before the home position, then the "speed during search for switch" segment is selected (for example, a limit switch is tripped before changing direction to find the home index).

The units are: (encoder counts per sample period) * 65536.

Object 609Ah: Homing Acceleration

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
609Ah	000	Homing Acceleration	00000000h	7FFFFFFFh	00000004h	Yes	Unsigned 32-bit	Read Write

This object is the acceleration and deceleration in Homing (HM) mode. The units are: (encoder counts per (sample²)) * 65536.

Object 60C0h: Interpolation Sub-Mode Select

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60C0h	000	Interpolation Sub-Mode Select	8000h	0000h	0000h	Yes	Signed 16-bit	Read Write

Interpolation (IP) mode uses the position data object (60C1h) and the interpolation time period object (60C2h) in one of the following ways:

- Linear interpolation (default): follows a linear set of positions in the times between the data points. The velocity during each segment between points is constant. The disadvantage is that the velocity changes abruptly at the data points; the advantage is that the actual path taken between points is very predictable.
- Spline interpolation: uses the current point, the next point, and the previous point to generate curvature of the path over time. This results in a more continuous velocity. Also, following of curved shapes is typically more accurate between points. However, the disadvantage can be certain cases where an overshoot of position can occur. While this is generally avoided in the algorithm, extreme cases will overshoot.

The following table shows the possible sub-mode functions. The sub-mode data is read from the buffer along with the associated data point; the sub-mode applies to the segment between that point and the previous point.

Value	Function
-3	Spline Interpolation
0	Linear Interpolation
1-32767	Reserved

In the following example, the sub-mode will use Spline Interpolation between points 3000 and 4000.

1. Set the Interpolation Sub-Mode Select object (60C0h) to the value 0.
2. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 2000
 - b. 3000
3. Set the Interpolation Sub-Mode Select object (60C0h) to the value -3.
4. Put data in buffer by writing the value 4000 to subindex 1 of the Interpolation Data Record object (60C1h).
5. Set the Interpolation Sub-Mode Select object (60C0h) to the value 0.
6. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 5000
 - b. 6000

Object 60C1h: Interpolation Data Record

Object	subindex	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60C1h	000	Number of Entries	01h	02h	02h	No	Unsigned 8-bit	Read Only
60C1h	001	Data Record 1	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write
60C1h	002	Data Record 2 (not supported)	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	N/A

This object is used to enter the position data required in Interpolation (IP) mode. Only subindex 1 is used; subindex 2 is not used.

When data is written to subindex 1, it is entered into the buffer. Also, the current values of the Interpolation User Bits object (2403h), Interpolation Sub-Mode object (60C0h) and the Interpolation Time object (60C2h) are captured and entered into the buffer with the same record as the position data.

The value read from this object is the most recent value written to this object — it is *not* an indication of the motor's current state.

NOTE: Object 60C1h, subindex 1, "Data Record 1" can only be written if the "buffer clear" property (object 60C4h, subindex 6) is set to a 1. By default, writing to a data record will produce an error until this action is taken.

NOTE: Object 60C1h, subindex 2, "Data Record 2" is not to be used.

Object 60C2h: Interpolation Time Period

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60C2h	000	Number of Elements	00h	FFh	02h	No	Unsigned 8-bit	Read Only
60C2h	001	Interpolation time units	00h	FFh	01h	Yes	Unsigned 8-bit	Read Write
60C2h	002	Interpolation time index	80h	3Fh	FDh (-3)	Yes	Signed 8-bit	Read Write

This object is used for Interpolated Position (IP) mode. The time written is captured when a data record is written using subindex 1 of the Interpolation Data Record object (60C1h). The time data is read from the buffer along with the associated data point. The time period applies to the segment between that point and the previous point. After it is started, the interpolation process reads data points out of the interpolation buffer once per the time period.

The default time index is -3, which gives the time units in milliseconds.

Interpolation Time Index	Value
-128 to -4	Not allowed (returns SDO error)
-3	0.001 seconds (default)
-2	0.01 seconds
-1	0.1 seconds
0	1 second
1 to 127	Not recommended

The representation of the time is a combination of a value (time units) and a decimal shift (time index):

$$\text{Time} = (\text{time units}) * 10^{(\text{time index})} \text{ seconds}$$

Desired time range	Resolution	Suggested Time Index	Suggested Time Units
1 to 255 milliseconds	0.001 seconds	-3	1 to 255
10 milliseconds to 2.55 seconds	0.010 seconds	-2	1 to 255
100 milliseconds to 4 seconds	0.100 seconds	-1	1 to 40
1 second to 4 seconds	1.000 seconds	0	1 to 4

In the following example, the time segment will be the longer time of 2 seconds between point 3000 and point 4000.

1. Set subindex 1 of the Interpolation Time Period object (60C2h) to the value 1.
2. Set subindex 2 of the Interpolation Time Period object (60C2h) to the value 0, which represents seconds.
3. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 2000
 - b. 3000
4. Set subindex 1 of the Interpolation Time Period object (60C2h) to the value 2.
5. Put data in buffer by writing the value 4000 to subindex 1 of the Interpolation Data Record object (60C1h).
6. Set subindex 1 of the Interpolation Time Period object (60C2h) to the value 1.
7. Put data in the buffer by writing the following values to subindex 1 of the Interpolation Data Record object (60C1h):
 - a. 5000
 - b. 6000

Object 60C4h: Interpolation Data Configuration

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60C4h	000	Number of Entries	00h	FFh	06h	No	Unsigned 8-bit	Read Only
60C4h	001	Maximum buffer size	00000000h	FFFFFFFFh	0000002dh	Yes	Unsigned 32-bit	Read Only
60C4h	002	Actual buffer size	00000000h	FFFFFFFFh	0000002dh	Yes	Unsigned 32-bit	Read Only
60C4h	003	Buffer organization	00h	FFh	00h	Yes	Unsigned 8-bit	Read Only
60C4h	004	Buffer position	0000h	FFFFh	0000h	Yes	Unsigned 16-bit	Read Only
60C4h	005	Size of data record	04h	04h	04h	Yes	Unsigned 8-bit	Read Only
60C4h	006	Buffer clear	00h	01h	00h	Yes	Unsigned 8-bit	Write Only

This object controls some miscellaneous aspects of the Interpolation mode buffer.

The subindex objects have the following functions:

- Subindex 1: Cannot be changed because the SmartMotor buffer cannot be resized. This object can be ignored.
- Subindex 2: Cannot be changed because the buffer cannot be resized. The value is 2Dh or 45 (decimal); this is the number of data records that can be held in the buffer. Each record contains information about the position, time, user bits and Interpolation mode for that segment.
- Subindex 3: Cannot be set. It reports the value 0, which indicates that the buffer is a FIFO type — data records are written into one end of the buffer and the motor firmware reads data out of the other end.
- Subindex 4: Reports the number of occupied buffer slots.
- Subindex 5: Not implemented.
- Subindex 6: Cannot be read. To control buffer access, write one of the values from the following table.

Subindex 6	Function
0	Clear input buffer, access disabled (will not accept writes to object 60C1h), clear all IP data records
1	Enable write access to the buffer (object 60C1h)
2–255	Reserved

Object 60F4h: Following Error Actual Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60F4h	000	Following Error Actual Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the actual value of the following error. This is the difference between the demand position and the actual position:

Following Error Actual Value object (60F4h) = Position Demand Value object (6062h) – Position Actual Value object (6064h)

Similar SmartMotor Commands: REA

Object 60FBh: Position Control Parameter Set

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60FBh	000	Number of Entries	00h	FFh	0Ah (10)	No	Unsigned 8-bit	Read Only
60FBh	001	KP, Proportional Gain	0000h	FFFFh	From EEPROM	Yes	Unsigned 16-bit	Read Write*
60FBh	002	KI, Integral Gain	0000h	7FFFh	From EEPROM	Yes	Unsigned 16-bit	Read Write*
60FBh	003	KL, Integral Limit	0000h	7FFFh	7FFFh	Yes	Unsigned 16-bit	Read Write*
60FBh	004	KD, Derivative Gain	0000h	FFFFh	From EEPROM	Yes	Unsigned 16-bit	Read Write*
60FBh	005	KS, Derivative Damping Sample Rate	00h	03h	01h	Yes	Unsigned 8-bit	Read Write*
60FBh	006	KV, Velocity Feedforward Gain	0000h	FFFFh	From EEPROM	Yes	Unsigned 16-bit	Read Write*
60FBh	007	KA, Acceleration Feed-forward Gain	0000h	FFFFh	From EEPROM	Yes	Unsigned 16-bit	Read Write*
60FBh	008	KG, Gravitational Offset	FF000000h	00FFFFFFh	From EEPROM	Yes	Signed 32-bit	Read Write*
60FBh	009	N/A	0000h	FFFFh	0000h	No	Unsigned 16-bit	Read Only
60FBh	010	Position Loop Control	00h	FFh	00h	Yes	Unsigned 8-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object contains manufacturer-specific parameters for the drive controller. For the SmartMotor, this is primarily used to set the PID parameters (see the following table).

NOTE: The PID parameters do not take effect until subindex 10 is written.

For more details on these PID parameters, see the *SmartMotor™ Developer's Guide*.

Similar SmartMotor Commands: KP=, RKP, KI=, RKI, KL=, RKL, KD=, RKD, KS=, RKS, KV=, RKV, KA=, RKA, KG=, RKG, F

Sub-index	SMI Command	PID Parameter	Function
1	RKP, KP=	KP	Proportional coefficient
2	RKI, KI=	KI	Integral coefficient
3	RKL, KL=	KL	Integral limit
4	RKD, KD=	KD	Derivative coefficient
5	RKS, KS=	KS	Velocity filter option for KD (value is 0, 1, 2 or 3; larger numbers specify longer filter times)
6	RKV, KV=	KV	Velocity feed-forward gain
7	RKA, KA=	KA	Acceleration feed-forward gain
8	RKG, KG=	KG	Gravitational offset
9			Reserved
10	F (no equal sign)		Position loop control (set bit 0 to the value 1 to make the PID parameters take effect)

Object 60FCh: Position Demand Internal Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60FCh	000	Position Demand Internal Value	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Only

This object reports the position calculated by the motion profile; it takes into account the acceleration and velocity targets. The value is in units of encoder counts.

When the motor is inactive or in torque mode, the value reported is simply the current position.

Similar SmartMotor Commands: RPC

Object Description

INDEX	60FC
Name	Position demand internal value
Object Code	Variable
Data Type	INTEGER32
Category	Optional

Entry Description

Access	RO
PDO Mapping	Yes
Default Value	00000000h
Lower Limit	80000000h
Upper Limit	7FFFFFFFh
Unit	-

Object 60FDh: Digital Inputs

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60FDh	000	Digital Inputs	00000000h	FFFFFFFFh		Yes	Unsigned 32-bit	Read Only

This object reports the current state of the digital input signals from the I/O connector(s).

For details, see the table corresponding to your motor model:

Class 5 D-style motor	
Bit	Function
0	Negative limit (if enabled)
1	Positive limit (if enabled)
2	Not supported
3	Not supported
4–15	Reserved
16	General purpose input 0
17	General purpose input 1
18	General purpose input 2
19	General purpose input 3
20	General purpose input 4
21	General purpose input 5
22	General purpose input 6
23–31	Reserved

Class 5 M-style motor	
Bit	Function
0	Negative limit (if enabled)
1	Positive limit (if enabled)
2	Not supported
3	Not supported
4–15	Reserved
16	General purpose input 0
17	General purpose input 1
18	General purpose input 2
19	General purpose input 3
20	General purpose input 4
21	General purpose input 5
22	General purpose input 6
23	General purpose input 7
24	General purpose input 8
25	General purpose input 9
26	General purpose input 10
27	Not fault state
28	Drive enable input
29–31	Reserved

Object 60FEh: Digital Outputs

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60FEh	000	Number of Entries	01h	02h	01h	No	Unsigned 8-bit	Read Only
60FEh	001	Physical Outputs	00000000h	FFFFFFFFh		Yes	Unsigned 32-bit	Read Write

This object allows the digital outputs to the I/O connector(s) to be set or cleared.

NOTE: There is no support for subindex 2.

For details, see the table corresponding to your motor model:

Class 5 D-style motor	
Bit	Function
0	Brake Set - Not Supported
1-15	Reserved
16	General purpose output 0
17	General purpose output 1
18	General purpose output 2
19	General purpose output 3
20	General purpose output 4
21	General purpose output 5
22	General purpose output 6
23	Unconnected bit; remembers value
24-31	Reserved

Class 5 M-style motor	
Bit	Function
0	Brake Set - Not Supported
1-15	Reserved
16	General purpose output 0
17	General purpose output 1
18	General purpose output 2
19	General purpose output 3
20	General purpose output 4
21	General purpose output 5
22	General purpose output 6
23	General purpose output 7
24	General purpose output 8
25	General purpose output 9
26	General purpose output 10
27-31	Reserved

Object 60FFh: Target Velocity

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60FFh	000	Target Velocity	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the CANCTL(13,0) command.								

This object only applies to Profile Velocity (PV) mode. The velocity profile will accelerate to the specified speed and remain at that speed until a stop is commanded or a new speed is specified.

Writing this value takes effect immediately in PV mode, assuming the motor is already in the operation enabled state through Control Word object (6040h). The units are: (encoder counts per sample period) * 65536.

Also, refer to Object 6081h: Profile Velocity in PP Mode on page 204.

Similar SmartMotor Commands: VT=, RVT

Object 6402h: Motor Type

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6402h	000	Motor Type	0000h	FFFFh	000Ah	No	Unsigned 16-bit	Read Only

This object reports the type of motor connected to the controller. The value of this object does not change. It always reports 000Ah (10 decimal), which represents a "Sinusoidal PM BL motor" (per the CiA 402 specification).

Object 6502h: Supported Drive Modes

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6502h	000	Supported Drive Modes	00000000h	FFFFFFFFh	0000006Dh	No	Unsigned 32-bit	Read Only

This object reports a value that corresponds to a bit field indicating the operational modes supported by the drive. The value reports as the default value listed above and does not change.

Bit	Bit relating to default value	Mode
0	1 Supported	Profile Position (PP)
1	0 Not supported	Velocity (VL)
2	1 Supported	Profile Velocity (PV)
3	1 Supported	Torque (TQ)
4	0 Not supported	Reserved
5	1 Supported	Homing (HM)
6	1 Supported	Interpolation (IP)
7	0 Not supported	Cyclic Synchronous Profile (CSP)
8	0 Not supported	Cyclic Synchronous Torque (CSV)
9	0 Not supported	Cyclic Synchronous Torque (CST)
10–15	00 0000	Reserved
16–31	0000 0000 0000 0000	Manufacturer specific

Object 67FFh: Single Device Type

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
67FFh	000	Single Device Type	00000000h	FFFFFFFFh	00020192h	No	Unsigned 32-bit	Read Only

This object specifies the type of device (profile) for objects in the range 6000h to 67FFh. Refer to the following table the possible values and their corresponding functions.

Bit	Value	Function
0–15	0192h (402 decimal)	DS402 device
16–23	02h (2 decimal)	Servo drive
24–31	0	Reserved (manufacturer specific)

Also, refer to Object 1000h: Device Type on page 99.

Reference Documents

The following CiA documents were referenced for this guide:

- CiA 402 CANopen - Drives and motion control device profile:

This specification is now comprised of the following IEC specifications:

- IEC 61800-7-1 (An abstracted view of motion control over a variety of protocols)
 - IEC 61800-7-201 (Describes the implementation of the 402 specification)
 - IEC 61800-7-301 (Describes the default settings of certain objects in the 402 specification)
- CiA 301 CANopen - Application layer and communication profile

The CiA documents are maintained by CAN in Automation (CiA):

<http://www.can-cia.org/>

The IEC documents are maintained by the International Electrotechnical Commission (IEC):

<http://www.iec.ch/>

PN: SC80100001-001
Rev. H