ETHERCAT® IMPLEMENTATION FOR

FULLY INTEGRATED SERVO MOTORS

CLASS 6 SMARTMOTOR™ TECHNOLOGY

Rev. L, February 2026

DESCRIBES THE CLASS 6 SMARTMOTOR™ SUPPORT FOR THE ETHERCAT® PROTOCOL

www.animatics.com

MOOG ANIMATICS

Copyright Notice

Moog Animatics Class 6 SmartMotor™ EtherCAT Guide, Rev. L, PN:SC80100002-001.

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 "EtherCAT Guide" in the subject line and be sent by e-mail to: animatics_sales@moog.com. Thank you in advance for your contribution.

Contact Us:

Moog Animatics
1995 NC Hwy 141
Murphy, NC 28906
USA

Website: www.animatics.com

Email: animatics_sales@moog.com

Table of Contents

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

This manual explains the Moog AnimaticsClass 6 SmartMotor™ support for the EtherCAT® protocol. It describes the major concepts that must be understood to integrate a SmartMotor follower with a PLC or other EtherCATcontroller[1]. However, it does not cover all the low-level details of the EtherCAT protocol.

NOTE: The feature set described in this version of the manual refers to motor firmware 6.0.2.21 (Class 6 M) / 6.4.2.54 (Class 6 D) or later.

This manual is intended for programmers or system developers who have read and understand the EtherCAT Technology Group (ETG) and CiA 402 specifications. Therefore, this manual is not a tutorial on that specification or the EtherCAT 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® over EtherCAT (CoE) to operate the SmartMotor.

The Object Reference chapter of this manual includes details about the specific objects available in the SmartMotor through EtherCAT. The objects include those required by EtherCAT, the CiA 402 motion profile, and manufacturer-specific objects added by Moog Animatics. For details, see Object Reference on page 72.

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 controller 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*.

[1]Moog Animatics has replaced the terms "master" and "slave" with "controller" and "follower", respectively.

Abbreviations

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

Abbreviation	Description
ACK	Acknowledgment
ADU	Acceleration/Deceleration Units
BOOT	Bootstrap (state)
CiA	CAN in Automation
COB	Communication Object
COB-ID	Communication Object Identification
CoE	CANopen over Ethernet
CSP	Cyclic Synchronous Position (mode)
CST	Cyclic Synchronous Torque (mode)
CSV	Cyclic Synchronous Velocity (mode)
DC	Direct Current NOTE: "DC" is also used with DC-Sync; in this case, it means "Distributed Clock"
DC-Sync	Distributed Clock Synchronization
ESC	EtherCAT Follower Controller
ESI	EtherCAT Follower Information (specification)
ESM	EtherCAT State Machine
EtherCAT	Ethernet for Control Automation Technology
FMMU	Fieldbus Memory Management Unit
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
SAFEOP	Safe Operation (state)

Abbreviation	Description
SDO	Service Data Object
SM	Sync Manager
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 these safety symbols:

WARNING: This symbol indicates a potentially nonlethal mechanical hazard, where failure to comply with 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 comply with 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, this 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 site where the machine is being installed and operated. For more details, see Machine Safety on page 14.

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 site 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 next 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 site 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 topic "Controls - Notes and Cautions" located at:

https://www.animatics.com/support/downloads/knowledgebase/controls---notes-and-cautions.html

OSHA standards information can be found at:

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

UL standards information can be found at:

https://www.ulse.org/insight/news-introduction-standards-what-standard-and-why-do-they-matter/

ISO standards information can be found at:

https://www.iso.org/standards.html

EU standards information can be found at:

https://www.europa.eu/youreurope/business/product-requirements/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 these lists.

Related Guides

- *Class 6 D-Style SmartMotor™ Installation and Startup Guide*

  https://www.animatics.com/wp-content/uploads/Moog-Animatics-SmartMotor-Class6DStyle-InstallationandStartupGuide-Manual-en.pdf

- *Class 6 M-Style SmartMotor™ Installation and Startup Guide*

  https://www.animatics.com/wp-content/uploads/2023/09/Moog-Animatics-SmartMotor-Class6MStyleInstallationandStartupGuide-Manual-en.pdf

- *SmartMotor™ Developer's Guide*

  https://www.animatics.com/wp-content/uploads/2023/09/Moog-Animatics-SmartMotor-DevelopersGuide-Manual-en.pdf

- *SmartMotor™ Homing Procedures and Methods Manual*

  https://www.animatics.com/wp-content/uploads/2023/09/Moog-Animatics-SmartMotor-HomingProceduresandMethods.pdf

In addition to the documents listed above, guides for fieldbus protocols and more can be found on the website: https://www.animatics.com/support/downloads.manuals.html

Other Documents

- SmartMotor™ Certifications

  https://www.animatics.com/support/downloads/certifications/

- *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)

  https://www.animatics.com/wp-content/uploads/2023/10/Class-5-Developers-Worksheet.xlsm

Additional Resources

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

- General company information:

  https://www.animatics.com

- Product information:

  https://www.animatics.com/products

- Product support (Downloads, How-to Videos, Forums and more):

  https://www.animatics.com/support

- Contact information, distributor locator tool, inquiries:

  https://www.animatics.com/contact-us

- Applications (Application Notes and Case Studies):

  https://www.animatics.com/applications

CANopen Resources

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

- CAN in Automation website:

  https://www.can-cia.org

EtherCAT Resources

EtherCAT is a common standard maintained by EtherCAT Technology Group (ETG):

- EtherCAT Technology Group website:

  https://www.ethercat.org/default.htm

- EtherCAT Technology Group website – EtherCAT description:

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

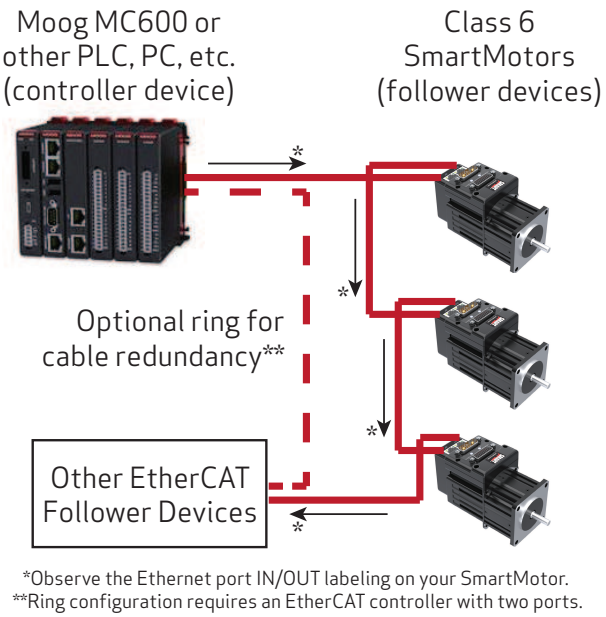
EtherCAT Overview

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

SmartMotor EtherCAT Overview

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

Ethernet for Control Automation Technology (EtherCAT) is a high-performance, Ethernet-based fieldbus system. EtherCAT uses full-duplex Ethernet physical layers. In short, EtherCAT has been designed to be an industrial-communication network typically between a controller and many different follower devices.



EtherCAT Example System Diagram

NOTE: Unlike other fieldbus protocols, EtherCAT does not require terminators at each end of the network bus.

Many network configurations are possible, such as line, tree or star. Requirements for specific configurations depend on the capabilities of the EtherCAT controller device, the follower devices, and use of other networking equipment. For more details, refer to the EtherCAT Technology Group website at:

https://www.ethercat.org

The SmartMotor uses a special communications protocol (language) between the controller and SmartMotor follower devices. This protocol is called CANopen over EtherCAT (CoE), which gives the controller device a mechanism to address and transport data structures to and from the SmartMotor. For details, see CANopen over EtherCAT (CoE) Description on page 20.

CANopen over EtherCAT (CoE) Description

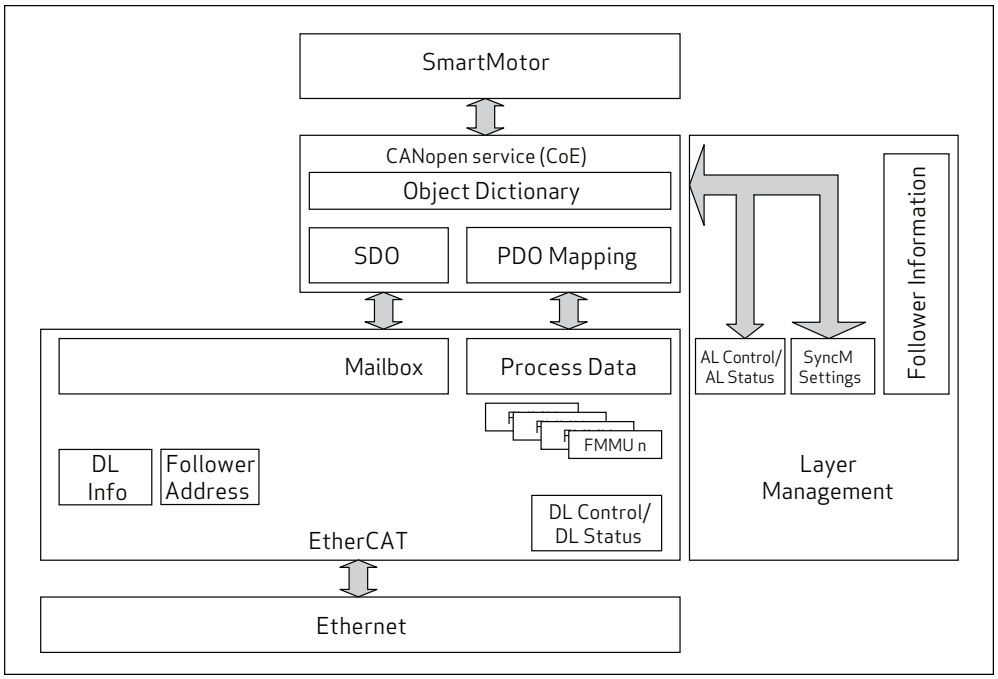
CANopen is a common standard that allows communication between various devices. It is a well-defined protocol that has been implemented over different physical bus structures. The SmartMotors support CANopen over EtherCAT (CoE). The protocol is very similar to the SmartMotor CANopen implementation over Control Area Network (CAN) bus. However, there are slight differences in communication objects in the range 1000h to 1FFFh to support EtherCAT.

CoE is the common EtherCAT communication protocol. It offers two different transport layers:

- SDO for acyclic communication using EtherCAT Mailbox.
- PDO for cyclic communication using EtherCAT Process Data.

For details on these transport layers, see PDO and SDO Communication on page 21.

CoE provides mechanisms to configure PDOs for effective and efficient cyclic data exchange, which is essential and primarily used for controls. See the next figure.



SmartMotor CoE Diagram

Object Dictionary

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 controller to obtain some basic information, such as range limits and descriptions, directly from the device.

EtherCAT Follower Information (ESI) files provide information, in XML file format, to PLCs and system integrators that describes the organization of the object dictionary information. The SmartMotor ESI XML file is provided on the Moog Animatics website Support > Downloads > Software > SmartMotor Fieldbus Configurator (EDS) Files at:

https://www.animatics.com/products/smartmotors/smartmotor-fieldbus-configurator-files/

After opening that page, click Fieldbus Configurator Files > EtherCAT.

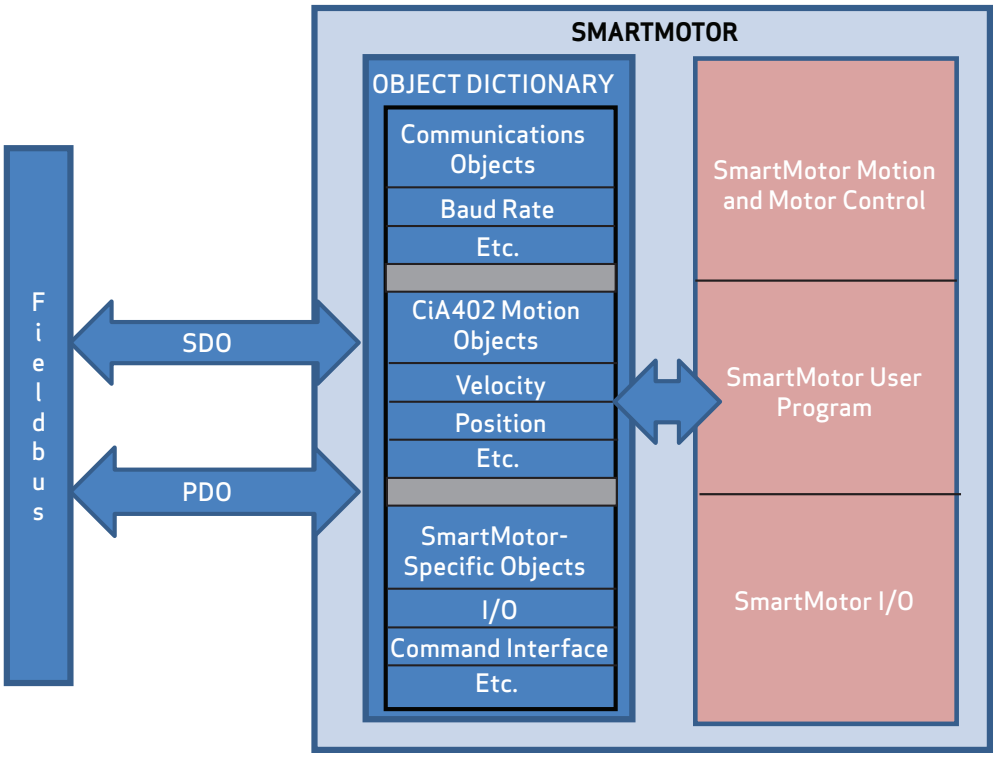
The XML file defines:

- The structures that are used for describing basic data types.
- The profiles or group of objects that are used for control of a SmartMotor. In the case of the SmartMotor, this means that features common to motor control are defined, and specific data objects are assigned to specific object numbers.

The SmartMotor supports the well-defined CANopen CiA 402 Drives and Motion Control Device Profile, which has been adapted to work on EtherCAT, also. That profile has also been adopted by the International Electrotechnical Commission (IEC) under the "Adjustable Speed Electrical Power Drive Systems" specification (IEC 61800-7-200).

PDO and SDO Communication

In CANopen over EtherCAT, 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 72.



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 EtherCAT controllers 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 ESM 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 because:

- 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 follower back to the controller. Both read and write operations confirm by either sending the requested data (read) or confirming that a command was received (write).

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 controller and the follower before entering the Operational state. For further information, see Dynamic PDO Mapping Using CoE on page 55.

This 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 not intended for cyclic updates and routinely accessed by the controller.
- 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).

EtherCAT State Machine (ESM)

The SmartMotor is required to run an EtherCAT State Machine (ESM) to indicate the available network functions. The state machine offers a mechanism to configure the follower device for individual applications. This occurs during the EtherCAT network initialization (INIT) phase and must complete before the follower device is considered fully operational.

The ESM states are:

- INIT (Initialization state): In this state, there is no communication to the individual follower objects. The controller accesses the DL-information registers, which are the very low-level registers at the Data-Link level of EtherCAT.
- PREOP (Pre-Operational state): In this state, there is mailbox (SDO) communication to the follower object dictionary. However, there are no PDO transfers taking place. This state does permit follower Process Data Object (PDO) configuration by the controller. Typically, the controller is allowed to configure the follower behavior before the follower enters subsequent network states.
- SAFEOP (Safe Operation state): In this state, there is mailbox (SDO) communication to the follower device and PDO input data communication to the controller. However, the follower does not accept Receive PDOs (RxPDOs) and its Transmit PDOs (TxPDOs) are maintained in a safe state. The follower is allowed to perform synchronization to the network DC-Sync during this phase, holding off transition to the subsequent Operational state.

- OP (Operational state): In this state, Process Data input and outputs are permitted. Therefore, the follower accepts RxPDO information and transmits TxPDO information.
- BOOT (Bootstrap state): This state is optional. Therefore, it is not supported by the SmartMotor.

During the EtherCAT network startup, the controller and the SmartMotor (follower device) handle the transitions between states using the AL-Control and AL-Status registers across the EtherCAT network (see ESM Transition Diagram on page 24). This information is provided for advanced network use and EtherCAT controller application development:

Summary of Actions: INIT to PREOP

The controller reads the Vendor ID, Product Code and Revision Number from the EEPROM and configures:

- DL registers (0x10:0x11).
- Sync Manager registers (registers 0x800+) for mailbox communication.
- Initialization for DC-Sync (if supported).

Finally, the controller requests PREOP state by writing 0x2 to the AL-Control register (register 0x120); it waits for status confirmation through the AL-Status register (register 0x130).

Summary of Actions: PREOP to SAFEOP

The controller configures follower parameters using mailbox communication:

- Process Data mapping of the SmartMotor. The SmartMotor validates the Process Data mapping and configuration requested by the controller. If the Process Data configuration is unacceptable, the SmartMotor returns an error.
- Registers for process data Sync Managers, configuring the dynamic payload size.
- Fieldbus Memory Management Unit (FMMU) registers (0x600 and greater).

Finally, the controller requests the SAFEOP state (0x4 to AL-Control register) and waits for confirmation through the AL-Status register.

Summary of Actions: SAFEOP to OP

The controller sends valid outputs and requests the OP state (0x8 to AL-Control register). The state is confirmed in the AL-Status register.

The SmartMotor evaluates the timing requirements due to Process Data payload size. The timing information is updated in CoE objects 1C32h and 1C33h – that information will be used by the controller application.

Incorrect EtherCAT Follower Controller (ESC) register configuration (DC-Sync, FMMU, Sync Manager, etc.) can cause INIT, PREOP and SAFEOP errors. The AL-Status register (register 0x134) indicates the error reasons.

AL-Control Register

The bits used for ESM state change requests are shown in the next table.

Bits 0:3 Value	Description
1h	Initialization (INIT) state of the network
2h	Request for Pre-Operational (PREOP) state
3h	Request for Bootstrap (BOOTSTRAP) state

Bits 0:3 Value	Description
4h	Request for Safe Operational (SAFEOP) state
8h	Request for Operational (OP) state

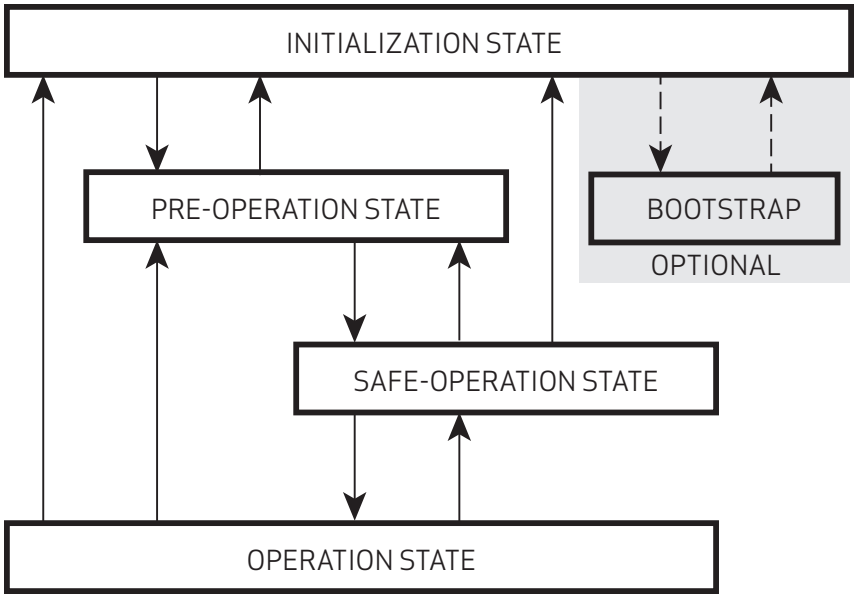
AL-Status Codes

The set of status codes used by the follower is extensive. For the purpose of this document, the next table provides a subset of the complete set of codes.

Code Value	Description
0x0000	No error
0x0001	Unspecified error
0x0002	Invalid request for state change
0x0003	Request for a unknown or unsupported state
0x0013	BOOTSTRAP state is not supported
0x0014	No valid firmware
0x0015	Invalid mailbox configuration in BOOTSTRAP state
0x0016	Invalid mailbox configuration in PREOP state
0x0017	Invalid Sync Manager configuration
0x0018	No valid inputs
0x0019	No valid outputs
0x001A	Synchronization error
0x001B	Sync Manager watchdog
0x001D	Invalid output configuration for PDO; check dynamic mapping
0x001E	Invalid input configuration for PDO; check dynamic mapping
0x0020	SmartMotor needs cold start or cycle power
0x0021	SmartMotor needs INIT
0x0022	SmartMotor needs PREOP
0x0023	SmartMotor needs SAFEOP
0x8000	SmartMotor is not ready
0x8001	PDO data sizes are not configured
0x8002	SmartMotor to Network Processor watchdog triggered
0x8003	DC-Sync configuration is invalid
0x8004	Firmware is booting
0x8005	Reboot of Network Processor requested
0x8006	EtherCAT network channel INIT requested
0x8007	Network Processor configuration is cleared

ESM Transition Diagram

The next diagram shows the relationship and interaction between the possible ESM states.



EtherCAT State Machine

For more details on the EtherCAT network management, see the EtherCAT Technology Group (ETG) website at:

https://www.ethercat.org/default.htm

PDO Communications over EtherCAT

Process Data Objects (PDO) are the cyclic data communications across the EtherCAT network between the controller and follower device. PDO-transfers using CoE across EtherCAT are facilitated with two different sync managers: Sync-Manager 2 (SM2) for Receive PDO (RxPDO) transfers from controller to follower, and Sync-Manager 3 (SM3) Transmit PDO (TxPDO) transfers from follower to controller.

The size of the either the RxPDO or TxPDO is limited to 78 bytes in the SmartMotor, where a byte is defined as a total of 8 bits. Therefore, during the process of dynamic mapping, the combined image of the SmartMotor assignment and PDO mapping needs to be considered. For more details, see Dynamic PDO Mapping Using CoE on page 55.

The EtherCAT controller and SmartMotor communicate the arrangement of PDO data during network initialization so that any arrangement of data can be handled. Therefore, objects of the CoE SmartMotor dictionary can be dynamically added by the application programmer.

Receive PDO Example

When operating the SmartMotor in CiA 402 Profile Velocity Mode, a typical RxPDO can be dynamically mapped and configured to contain these objects from the CoE dictionary:

Control Word object (6040h) – Unsigned 16 bits

Target Velocity object (60FFh) – Signed 32 bits

To further support the application, other objects are easily mappable to the RxPDO as needed. For example, the Profile Acceleration object (6083h) could be added to the above list – this would allow the EtherCAT controller to change the velocity profile.

Transmit PDO Example

When operating the SmartMotor in CiA 402 Profile Velocity Mode, a typical TxPDO can be dynamically mapped and configured to contain these objects from the CoE dictionary:

- Status Word object (6041h) – Unsigned 16 bits
- Position Actual Value object (6064h) – Signed 32 bits

To further support the application, other objects can be mapped to the TxPDO as needed. For example, the DC Link Circuit Voltage object (6079h) could be added to the above list – this would allow the EtherCAT controller to monitor the servo bus voltage at the drive section of the SmartMotor.

Synchronized PDO Communications

The SmartMotor supports isochronous (equal time interval) synchronization using the EtherCAT Distributed Clock feature. For supported synchronization modes, Object 1C32h DC-Sync Manager 2 Receive Object on page 96

Other Communications with the Motor

In addition to communicating with the SmartMotor as an EtherCAT 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 corresponding SmartMotor Installation and Startup Guide.

Supported Features

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

Supported CoE Features

This section describes the CANopen over Ethernet (CoE) features that are supported by the SmartMotor.

CiA 402 Motion Modes

These 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
- Torque (TQ, mode of operation: 4) – behaves like the SmartMotor MT mode
- Cyclic Sync Position (CSP mode, mode of operation: 8)
- Cyclic Sync Velocity (CSV mode, mode of operation: 9)
- Cyclic Sync Torque (CST mode, mode of operation: 10)
- Homing (HM mode, mode of operation: 6)

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

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 EtherCAT mapping procedure.

Dynamic mapping of objects to PDO is configured using objects 1600h, 1601h, 1602h, 1603h, 1604h, 1A00h, 1A01h, 1A02h, 1A03h and 1A04h. For details, see Dynamic PDO Mapping Using CoE on page 55.

Configurable Sync Manager 2 and 3 Assignment

For cyclic transfers, the SmartMotor allow the application program to define the number of PDO mappings to a Sync Manager.

DC-Sync Subordinate Mode with SYNC0 and SYNC1

The SmartMotor EtherCAT Distributed Clock Synchronization (DC-Sync) hardware supports real-time synchronization between the SmartMotors, the EtherCAT controller and other follower devices. The SmartMotor requires both SYNC0 and SYNC1 signals from the DC-Sync hardware to operate in Subordinate Mode. The SYNC0 signal is used to synchronize the internal time base of the SmartMotor with the DC-Clock of the EtherCAT network. The SYNC1 signal is used for PDO transfer update synchronization. The SYNC1 signal timing needs to be adjusted depending on the amount of PDO data using the feedback and measurements obtained from objects 1C32h and 1C33h. For details, see Object 1C32h DC-Sync Manager 2 Receive Object on page 96.

DC-Sync Follower

The SmartMotor EtherCAT hardware supports features enabling it to maintain Distributed Clock Synchronization (DC-Sync) across a network segment.

Selectable Homing Modes

The SmartMotorEtherCAT hardware supports selectable homing modes (with or without index pulse). Homing methods 1–14 are available, which use an index pulse; homing methods 17–30 are also available, which are similar to methods 1–14 but do not use an index pulse.

Selectable Interpolation Modes

The SmartMotorEtherCAT hardware supports selectable (linear or spline) interpolation modes. Linear interpolation (default) generates a linear set of positions in the times between the data points. Spline interpolation uses the current point, the next point, and the previous point to generate curvature of the path over time.

Touch Probe Function

The SmartMotor EtherCAT hardware supports a touch probe function, which allows the motor's position to be captured on a specific event. This feature is commonly used for homing, registration applications or other cases where the motor position must be recorded at a specific point in time. This value can be read back later, in a less time-critical manner, from the capture register.

Status LEDs

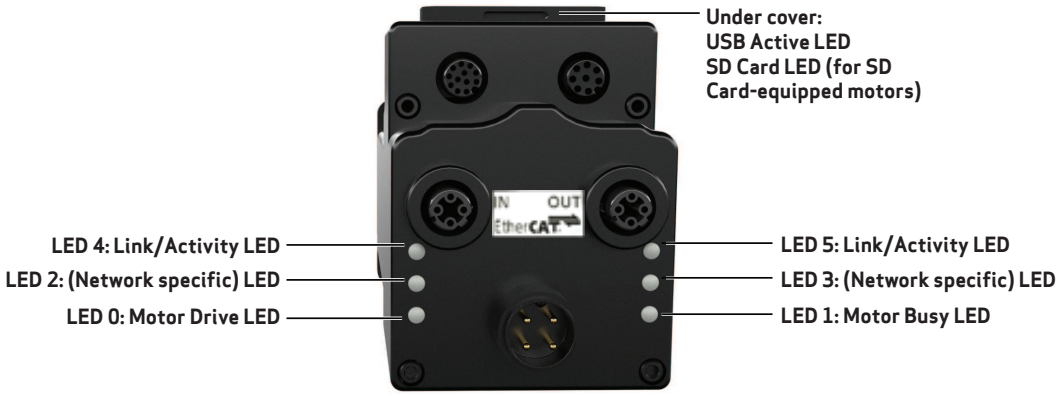
This chapter provides a description of the SmartMotor status LEDs.

NOTE: For information on the SmartMotor's connector pinouts and cable diagrams, refer to the corresponding SmartMotor Installation and Startup Guide. Pay particular attention to the IN/OUT labels on the Ethernet ports – these must be observed for EtherCAT networks.

NOTE: If you have set your PC's network adapter to a fixed IP address for temporary connections to SmartMotors with SMI, remember to return it to DHCP when done to avoid local area network connectivity issues.

Status LEDs - Class 6 M-Style

This section describes the functionality of the Status LEDs on the Class 6 M-style SmartMotor.



SD Card LED (for SD Card-equipped motors)

Off	No card, bad or damaged card
Blinking green	Busy, do not remove card
Solid green	Card detected
Solid red	Card with no SmartMotor data

See the topic "Understanding the SD Card" for details.

USB Active LED

Flashing green	Active
Flashing red	Suspended
Solid red	USB power detected, no configuration

If the USB port is plugged in at power up, it flashes for ~4 seconds, turns solid red until it is detected through SMI, then it returns to flashing

LED 0: Motor Drive LED

Off	No power
Solid green	Drive on
Blinking green	Drive off, no faults
Triple red flash	Watchdog fault
Solid red	Faulted or no drive enable input
Alt. red/green	In boot load; needs firmware

LED 0 and 1 Status on Power-up:

- With no program and the travel limit inputs are low:
  LED 0 solid red; motor in fault state due to travel limit fault
  LED 1 off
- With no program and the travel limits are high:
  LED 0 solid red for 500 milliseconds then flashing green
  LED 1 off
- With a program that only disables travel limits:
  LED 0 red for 500 milliseconds then flashing green
  LED 1 off

LED 1: Motor Busy LED

Off	Not busy
Solid green	Drive on, trajectory in progress
Flashing # red	Flashes fault code (see below) when Drive LED is solid red

Fault Codes: pauses for 2 sec before flashing the code

Flash	Description
1	NOT Used
2	Bus Voltage
3	Over Current
4	Excessive Temperature
5	Excessive Position
6	Velocity Limit
7	dE/Dt - First derivative of position error is excessive
8	Hardware Positive Limit Reached
9	Hardware Negative Limit Reached
10	Software Positive Travel Limit Reached
11	Software Negative Travel Limit Reached

LED 2: EtherCAT Error LED

Off	No error
Flickering red	Booting error
Blinking red	Invalid configuration; check DC sync settings
Single red flash	Follower device application has changed the EtherCAT state
Double red flash	Application controller failure
Solid red	Application controller failure

LED 4: Link/Activity LED

Off	No/bad cable; no/bad Link port
Solid green	Link established
Blinking green	Activity

LED 3: EtherCAT Run LED

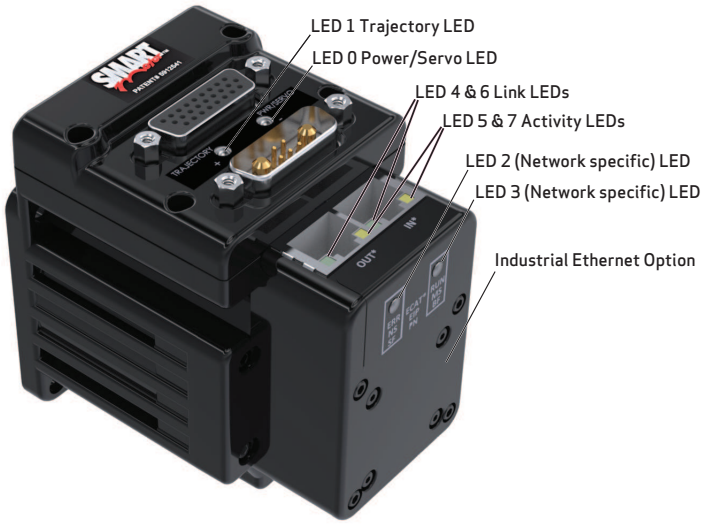
Off	No Error
Flickering	Initializing
Blinking	Pre-operational state
Single flash	Safe-operational state
Solid	Normal operation

LED 5: Link/Activity LED

Off	No/bad cable; no/bad Link port
Solid green	Link established
Blinking green	Activity

Status LEDs - Class 6 D-Style

This section describes the functionality of the Status LEDs on the Class 6 D-style SmartMotor.



LED 0: Power/Servo LED

Off	No power
Solid green	Drive on
Blinking green	Drive off, no faults
Triple red flash	Watchdog fault
Solid red	Faulted or no drive enable input
Alt. red/green	In boot load; needs firmware

LED 0 and 1 Status on Power-up:

- With no program and the travel limit inputs are low:
  LED 0 solid red; motor in fault state due to travel limit fault
  LED 1 off
- With no program and the travel limits are high:
  LED 0 solid red for 500 milliseconds then flashing green
  LED 1 off
- With a program that only disables travel limits:
  LED 0 red for 500 milliseconds then flashing green
  LED 1 off

LED 1: Trajectory LED

Off	Not busy
Solid green	Drive on, trajectory in progress
Flashing # red	Flashes fault code (see below) when Power/Servo LED is solid red

Fault Codes: pauses for 2 sec before flashing the code

Flash	Description
1	NOT Used
2	Bus Voltage
3	Over Current
4	Excessive Temperature
5	Excessive Position
6	Velocity Limit
7	dE/Dt - First derivative of position error is excessive
8	Hardware Positive Limit Reached
9	Hardware Negative Limit Reached
10	Software Positive Travel Limit Reached
11	Software Negative Travel Limit Reached

Industrial Ethernet Option

LED 2: EtherCAT Error LED

Off	No error
Flickering red	Booting error
Blinking red	Invalid configuration; check DC sync settings
Single red flash	Follower device application has changed the EtherCAT state
Double red flash	Application controller failure
Solid red	Application controller failure

LED 4 & 6 Link LEDs

Off	No/bad cable; no/bad Link port
Solid green	Link established

Industrial Ethernet Option

LED 3: EtherCAT Run LED

Off	No Error
Flickering	Initializing
Blinking	Pre-operational state
Single flash	Safe-operational state
Solid	Normal operation

LED 5 & 7 Activity LEDs

Off	No activity
Blinking amber	Activity

Manufacturer-Specific Objects

This chapter provides details on manufacturer-specific objects.

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 192 and Object 60FEh: Digital Outputs on page 194.

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 EtherCAT motors. It can be used to disable the limit inputs if desired. For more details, see Object 2101h: Bit IO on page 101.

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, EtherCAT 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 EtherCAT objects. This provides a common area where information can be shared between a user program and the EtherCAT 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*.

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 controller 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 102.

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 next 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, these 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 next 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 130.

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 EtherCAT access to the GOTO function.

Command Interface (Object 2500h)

The SmartMotor has many commands that are not mapped to CoE 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 that:

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

The next 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.
2500h	4	Program print & report string "OCTET STRING" type. It may contain nulls. Note that null is not considered a terminator in this string. The string length is indicated by the size of SDO transfer, up to 64 bytes.

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

Bit	Description
0	Command in progress.
1	Command complete/response ready (of object 2500h subindex 2).
2	Overflow of command response (of object 2500h subindex 2).
3	Reserved.
4*	Program output (of object 2500h subindex 4)–PRINT or report in a running user program is waiting to be read.
5*	Program output overflow (of object 2500h subindex 4)–the 64 bytes was exceeded by a PRINT in the user program.
6-7	Reserved.
*These two bits are cleared by a read of object 2500h subindex 4. Note that a user program will pause on the next PRINT statement for up to a certain timeout period specified by the CANCTL command. For details, see CANCTL(action, value) on page 1.	

This 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 next sections describe the upload and download procedures.

Upload from Motor

These 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 (SMX file)

First, an SMX file must be generated from the SMS source program in the SMI software, Be sure that the correct motor target was chosen, You may need to select Compile > Compiler Default Firmware Version from the SMI main menu.

These 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 (bit 0) 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 comes after it.
4. The host waits for the "Command in progress" flag (bit 0) 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. There may be a delay in responses from CANopen as the motor is busy finalizing the program load. If this causes timeouts, increase the amount of time before requesting handshake on this last section.

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

Download to Motor (SMXE encrypted file)

First, convert an existing SMX file to SMXE format. From the SMI software main menu, selectTools > Create smxe File.

NOTE: At the time the SMX is compiled from the SMS program, be sure that the SMX file is compiled for the specific motor type that you will be loading into. From the SMI software main menu, select Compile > Compiler Default Firmware Version.

These 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(7) command. Strings need to be null-terminated like most commands.
2. The host waits for the "Command in progress" flag (bit 0) 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. Because the data is encrypted, you will simply copy byte-for-byte from the source SMXE file. This can include a header and anything after the header. The CAN command manager will consume the header and whatever comes after it.
4. The host waits for the "Command in progress" flag (bit 0) 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. There may be a delay in responses from CANopen as the motor is busy finalizing the program load. If this causes timeouts, increase the amount of time before requesting handshake on this last section.

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 137). This profile is supported by many vendors of industrial controls.

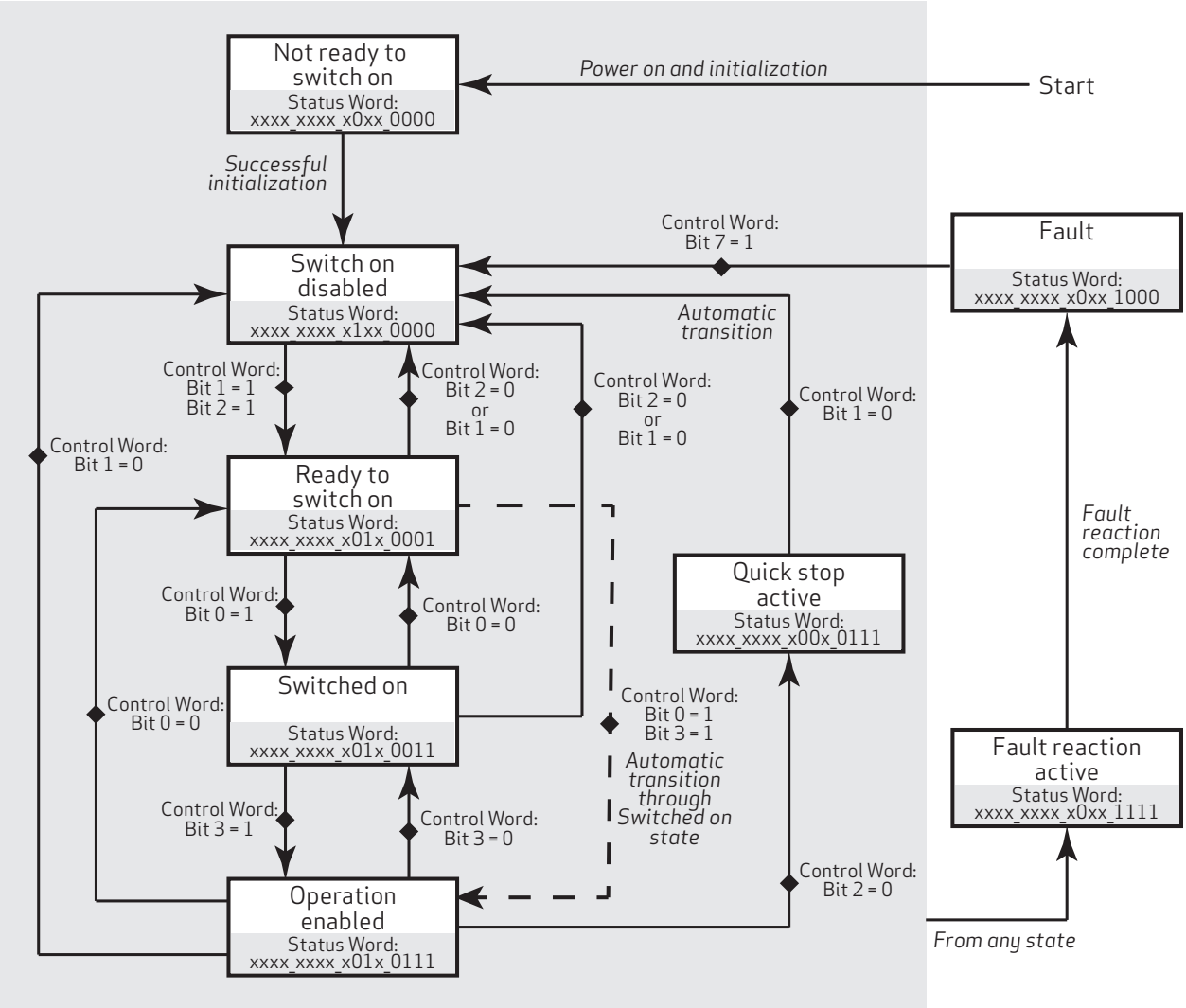
CiA 402 Profile Motion State Machine

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 43.

Control Words, Status Words and the Drive State Machine

Refer to the next 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. These 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 141.

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 next table describes the bits in the Control Word object (6040h). For more details, see Object 6040h: Control Word on page 139.

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 controller 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 2101h: Bit IO on page 101.

Absolute Position Mode Summary

The nexttable 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	128 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	6 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	7 0000 0000 0000 0111
Enable command, single set-point (motion not actually started yet)		6040h	00	02	002F	47 0000 0000 0010 1111
Begin motion to target position	G	6040h	00	02	003F	63 0000 0000 0011 1111
Prepare for next command		6040h	00	02	002F	47 0000 0000 0010 1111
Set target position	PT=1000	607Ah	00	04	000003E8	1000
Begin motion to target position	G	6040h	00	02	003F	63 0000 0000 0011 1111

Absolute Position Mode Example

This 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 (sample^2)) * 65536
Velocity Units (VU): encoder counts per sample * 65536

1. Clear the faults by setting the Control Word object (6040h) to these 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 40.
    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 these 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 these 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

This 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 these 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 these 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 these 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 these 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 2101h: Bit IO on page 101.

Velocity Mode Summary

The next 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	128 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	6 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	7 0000 0000 0000 0111
Start command	G	6040h	00	02	000F	15 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	271 xxxx xxx1 0000 1111
Start command	G	6040h	00	02	000F	15 0000 0000 0000 1111
Quick stop command (bit 2 = 0)	Quick stop then OFF See objects 6085h, 605Ah	6040h	00	02	000B	11 xxxx xxxx 0000 1011

Velocity Mode Example

This 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 (sample^2)) * 65536
Velocity Units (VU): encoder counts per sample * 65536

1. Clear the faults by setting the Control Word object (6040h) to these 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 40.
7. Start, stop and resume the motion by setting the Control Word object (6040h) to these 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 2101h: Bit IO on page 101.

Torque Mode Summary

The next 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	128 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	6 0000 0000 0000 0110
Change state: Switched on		6040h	00	02	0007	7 0000 0000 0000 0111
Start command	G	6040h	00	02	000F	15 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)	X (default) See object 605Dh	6040h	00	02	010F	271 xxxx xxx1 0000 1111
Start command	G	6040h	00	02	000F	15 0000 0000 0000 1111
Quick stop command (bit 2 = 0)	Quick stop then OFF See object 605Ah	6040h	00	02	000B	11 xxxx xxxx 0000 1011

Torque Mode Example

This 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 154. Also, see
Object 6087h: Torque Slope on page 166.

1. Clear the faults by setting the Control Word object (6040h) to these 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 40.
6. Start, stop and resume the motion by setting the Control Word object (6040h) to these 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.

Cyclic Synchronous Position (CSP) Mode

This section describes the process for creating a motion using Cyclic Synchronous Position (CSP) mode.

In CSP mode, the EtherCAT controller is responsible for defining the velocity profile to the motor and is in control of the trajectory generation. The SmartMotor will interpolate the target position between these cyclic updates. The EtherCAT controller defines the acceleration and velocity provided by the cyclic updates of the absolute target-position values.

NOTE: DC Sync (both SYNC0 and SYNC1) must be configured and enabled. SYNC1 is used as the time base for interpolation between position data points.

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 2101h: Bit IO on page 101.

CSP Control and Status Word

For this operation mode, there are no mode-specific bits in the Control Word object (6040h) – operation mode bits 4, 5, 6 and 9 are ignored by the SmartMotor. In this operation mode, the Halt bit (8) is also ignored by the SmartMotor because the Halt function is managed by the controlling device. For this mode, the definition of the Status Word object (6041h) is defined as shown in the next table.

Bit	Definition
10	Reserved (use either 0 or 1)
12	0 = Target Position is ignored; 1 = Target Position is used
13	0 = NO position error; 1 = position error

CSP Mode Example

This procedure shows the steps for creating a motion using cyclic updates of the target position.

1. Clear the faults by setting the Control Word object (6040h) to these values:
    a. 0
    b. 0080h (128 decimal)
    c. 0
2. Using the same object, enable the ready switch by setting the value to 0006h.
3. Set the Modes of Operation object (6060h) to the value 8 (decimal).
4. Set the Max Motor Speed object 6080h to the desired RPM limit (for example, the decimal value 1000).
5. Enable motion by setting the Control Word object (6040h) to the value 000Fh. The SmartMotor:
    - Monitors the position error.
    - Uses the target position updates as absolute values. See the next NOTE.
6. Read the Position Actual Value object (6064h), and initialize the trajectory generator.
7. Update the Target Position object (607Ah). This step must be cyclically repeated to complete the move.

    NOTE: The motor interprets the value written to the Target Position object (607Ah) as an absolute position. For example, if the previous value written to object 607Ah was 100, and the next value written is 500, then the move will be 400 counts forward. Therefore, to avoid a large jump, the first target written to object 607Ah should be chosen carefully in relation to the motor's current position.
8. Disable the operation by setting the Control Word object (6040h) to the value 0007h.
9. Turn off the motor by setting the Control Word object (6040h) to the value 0.

Cyclic Synchronous Velocity (CSV) Mode

This section describes how to use Cyclic Sync Velocity (CSV) mode to create a motion.

In CSV mode, the trajectory generator is located in the control device, not in the SmartMotor device. In a cyclic-synchronous manner, it provides a target velocity to the SmartMotor, which performs velocity control and torque control. If desired, the position-control loop may be closed over the communication system. Also, the SmartMotor can provide the controlling device with actual values for position, velocity and torque.

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 2101h: Bit IO on page 101.

CSV Control and Status Word

For this operation mode, there are no mode-specific bits in the Control Word object (6040h) – operation mode bits 4, 5, 6 and 9 are ignored by the SmartMotor. In this operation mode, the Halt bit (8) is also ignored by the SmartMotor because the Halt function is managed by the controlling device. For this mode, the definition of the Status Word object (6041h) is defined as shown in the next table.

Bit	Definition
10	Reserved (use either 0 or 1)
12	0 = Target Velocity is ignored; 1 = Target Velocity is used
13	Reserved (use either 0 or 1)

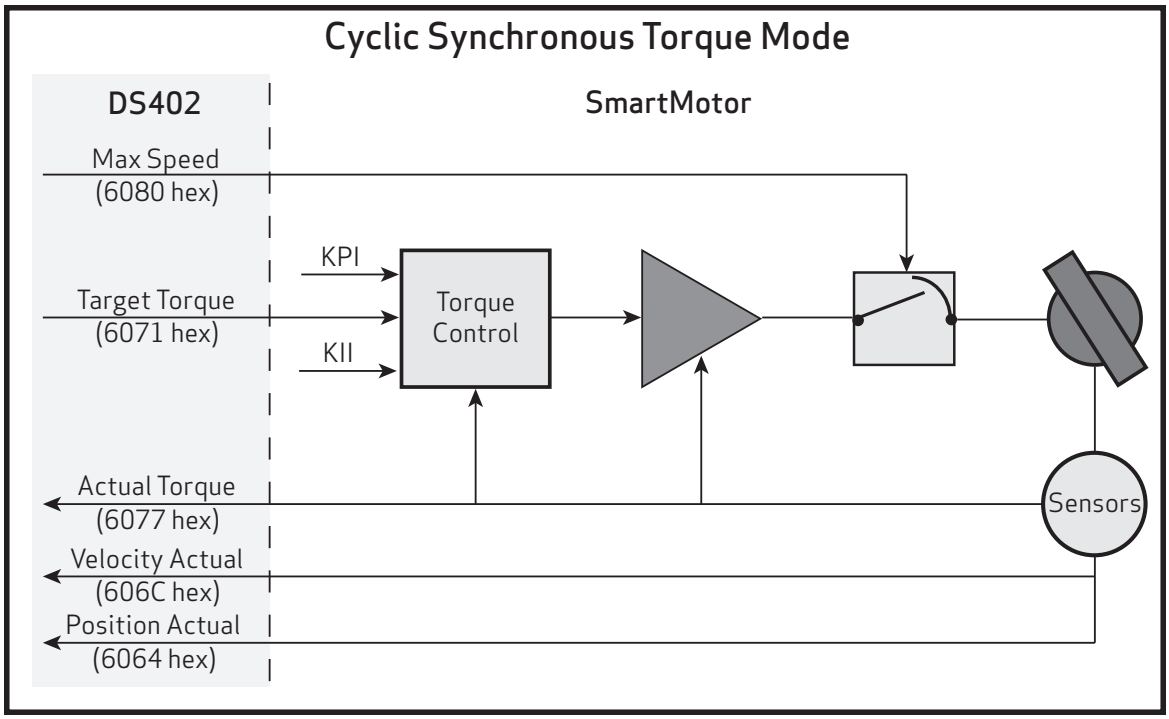
CSV Mode Example

This procedure shows the steps for creating a motion using cyclic updates of the target velocity. Incrementing the target velocity from one update to the next will cause acceleration.

1. Clear the faults by setting the Control Word object (6040h) to these values:
    a. 0
    b. 0080h (128 decimal)
    c. 0
2. Using the same object, enable the ready switch by setting the value to 0006h.
3. Set the Modes of Operation object (6060h) to the value 0009h.
4. Set the Max Motor Speed object 6080h to the desired RPM limit (for example, the decimal value 1000).
5. Enable motion by setting the Control Word object (6040h) to the value 000Fh.
6. Read the Velocity Actual Value object (606Ch), and initialize the trajectory generator.
7. Update the Target Velocity object (60FFh). This step must be cyclically repeated to complete the move.
8. Disable the operation by setting the Control Word object (6040h) to the value 0007h.
9. Turn off the motor by setting the Control Word object (6040h) to the value 0.

Cyclic Synchronous Torque (CST) Mode

This section describes the process for creating a force using Cyclic Synchronous Torque (CST) mode.



Cyclic Synchronous Torque Mode Diagram

In CST mode, the trajectory generator is located in the EtherCAT control device instead of the SmartMotor. In a cyclic synchronous manner, the control device provides a target torque to the SmartMotor, which performs torque control. The SmartMotor can provide actual values for position, velocity and torque to the control device. However, the control device must close the velocity loop and/or position loop.

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 2101h: Bit IO on page 101.

CST Control and Status Word

For this operation mode, there are no mode-specific bits in the Control Word object (6040h) – operation mode bits 4, 5, 6 and 9 are ignored by the SmartMotor. In this operation mode, the Halt bit (8) is also ignored by the SmartMotor because the Halt function is managed by the controlling device. For this mode, the definition of the Status Word object (6041h) is defined as shown in the next table.

Bit	Definition
10	Reserved (use either 0 or 1)
12	0 = Target Torque is ignored; 1 = Target Torque is used
13	Reserved (use either 0 or 1)

CST Mode Example

This procedure shows the steps for creating a moment of force (torque) at the SmartMotor shaft using cyclic updates of the target torque.

1. Clear the faults by setting the Control Word object (6040h) to these values:
    a. 0
    b. 0080h (128 decimal)
    c. 0
2. Using the same object, enable the ready switch by setting the value to 0006h.
3. Set the Modes of Operation object (6060h) to the value 8 (decimal).
4. Set the Max Motor Speed object 6080h to the desired RPM limit (for example, the decimal value 1000).
5. Enable motion by setting the Control Word object (6040h) to the value 000Fh
6. Read the Torque Actual object (6077h), and initialize the trajectory generator.
7. Update the Target Torque object (6071h). This step must be cyclically repeated to complete the move and close the control loops.
8. Disable the operation by setting the Control Word object (6040h) to the value 0007h.
9. Turn off the motor by setting the Control Word object (6040h) to the value 0.

Dynamic PDO Mapping Using CoE

This section describes the low-level steps for enabling PDO communications that must occur at startup between the EtherCAT controller and the SmartMotor.

Overview

Process Data Objects (PDOs) are containers that hold one or more objects of data. 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 sent from the motor. The same can be done for receive PDOs; the motor will unpack the PDO it receives according to the mapping configuration and consume the data objects.

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 EtherCAT devices that support dynamic mapping, which includes CoE devices. For details on the Communication Profile objects, see Communication Profile on page 76.

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

Mapping and Communication Parameters Objects

The next 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**
5632	1600	Receive PDO1 Mapping Parameters
5633	1601	Receive PDO2 Mapping Parameters
5634	1602	Receive PDO3 Mapping Parameters
5635	1603	Receive PDO3 Mapping Parameters
5636	1604	Receive PDO5 Mapping Parameters
6656	1A00	Transmit PDO1 Mapping Parameters
6657	1A01	Transmit PDO2 Mapping Parameters
6658	1A02	Transmit PDO3 Mapping Parameters
6659	1A03	Transmit PDO3 Mapping Parameters
6660	1A04	Transmit PDO5 Mapping Parameters
7186	1C12	Receive PDO SM Assignment
7187	1C13	Transmit PDO SM Assignment

Mapping Parameters Objects

The Mapping Parameters objects (receive and transmit) all 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 number 2 here.
1	Mapping Entry 1: Points to mapped object.*
2	Mapping Entry 2: Points to mapped object.*
3	Mapping Entry 3: Points to mapped object.*
4	Mapping Entry 4: Points to mapped object.*
*For details, see Mapping Entries on page 57, and see Dynamic PDO Assignment and Mapping Procedure on page 58.	

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 to be mapped:

- The object number
- The object subindex (0 if no sub-object)
- The object size (in bits)

Therefore, in the form: (hex) nnnniiss

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

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

(hex) 606c0020

CAUTION: There is a specific procedure defined by the EtherCAT specification for mapping a variable. This procedure must be used or an error will occur and prevent the change to the mapping.

Sync Manager Assignment Parameters

The Sync Manager Assignment objects (receive and transmit) 1C12 and 1C13 all have sub-objects of the same structure:

subindex (decimal)	Description
0	Number of Entries. Defines the number of PDO objects that are assigned to the SM. For instance, if "Assignment Entry 1" and "Assignment Entry 2" have been set up, then write the value 2.
1	For 1C12, valid entries are 1600h-1604h. For 1C13, valid entries are 1A00h-1A04h.
2	For 1C12, valid entries are 1600h-1604h. For 1C13, valid entries are 1A00h-1A04h.
3	For 1C12, valid entries are 1600h-1604h. For 1C13, valid entries are 1A00h-1A04h.
4	For 1C12, valid entries are 1600h-1604h. For 1C13, valid entries are 1A00h-1A04h.
5	For 1C12, valid entries are 1600h-1604h. For 1C13, valid entries are 1A00h-1A04h.

Dynamic PDO Assignment and Mapping Procedure

This 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 ESM Pre-Operational state.
2. Set the number of entries to 0 in subindex 0 of the Sync Manager 2 PDO Assignment object (1C12h) and the Sync Manager 3 PDO Assignment object (1C13h).
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 order: highest 2 bytes: object; next byte: subindex; the last byte: length in bits.
    a. For the actual velocity, set subindex 2 = 606c0020h.
    b. For the status word, set subindex 1 = 60410010h.
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. Set the number of entries to 0 in subindex 0 of the Receive PDO Mapping Parameter 1 object (1600h).
7. Set the number of entries to 0 in subindex 0 of the Receive PDO Mapping Parameter 2 object (1601h).
8. Set the mapping object. The value is a 32 -bit value in this order: highest 2 bytes: object; next byte: subindex; the last byte: length in bits.
    a. Set subindex 1 of object 1600h = 60400010h for Control Word.
    b. Set subindex 1 of object 1601h = 60FF0020h for Actual Velocity.

9. Set the number of entries in the mapping parameter objects back to the number of items created in the previous step:
    a. Set subindex 0 of object 1600h to the value 1.
    b. Set subindex 0 of object 1601h to the value 1.
10. Set the assignments for Sync Manager 2:
    a. Set subindex 1 of object 1C12h = 1600h.
    b. Set subindex 2 of object 1C12h = 1601h.
11. Set the assignments for Sync Manager 3:
    a. Set subindex 1 of object 1C13h = 1A00h.
12. Set the number of entries in the assignment parameter objects back to the number of items created in the previous step:
    a. Set subindex 0 of object 1C12h to the value 2.
    b. Set subindex 0 of object 1C13h to the value 1.
13. Enter the ESM Safe Operational state.

EtherCAT Synchronization Overview

Using the hardware signals SYNC0 and SYNC1, which are based on the EtherCAT distributed clock (DC) unit, the SmartMotor can operate in either Free Run or DC Synchronization (DC-Sync) mode.

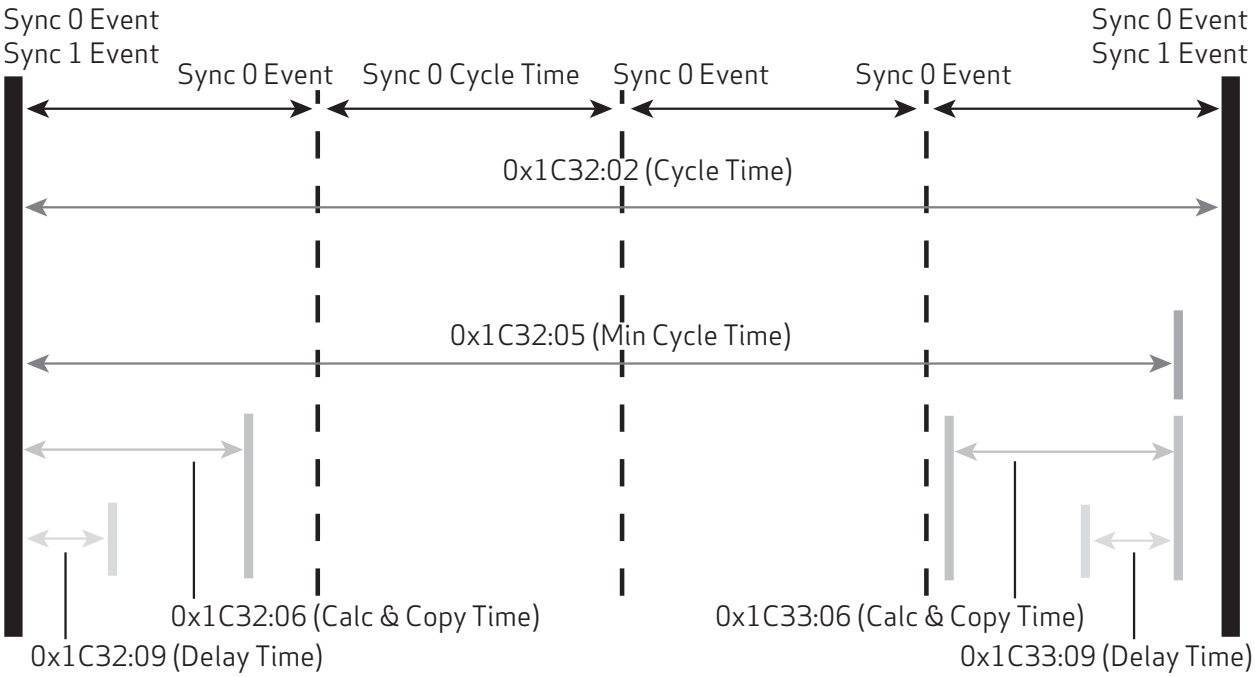
Free Run Mode

As the name implies, Free Run mode offers no synchronization between the controller and followers (SmartMotors). The SmartMotor will accept data routinely but without synchronization.

DC Synchronization – Subordinate Mode

The SmartMotor offers DC-Sync using the EtherCAT Subordinate mode. In this mode, the SmartMotor will phase lock loop its internal real-time motor control to the SYNC0 signal derived from the DC unit. When Subordinate mode is used, the SYNC1 signal is a multiple of the SYNC0 signal. In the SmartMotor, the SYNC1 command is used to synchronize processing of the Receive PDO and Transmit PDO process-data transfers.

DC-Sync Manager 2 Receive object (1C32h) and DC-Sync Manager 3 Transmit object (1C33h) are used during the initial phases of the EtherCAT network to configure and maintain DC-Sync. The size of the process data Receive PDO and Transmit PDO can change the "Calc and Copy Time" needed by the SmartMotor. This time can be read from subindex 6 of objects 1C32h and 1C33h. For more details on these objects, see Object 1C32h DC-Sync Manager 2 Receive Object on page 96 and Object 1C33h DC-Sync Manager 3 Transmit Object on page 98.



DC Synchronization Diagram

EtherCAT User Program Commands

This chapter provides details on the EtherCAT 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.

EtherCAT Error Reporting Commands

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

=ETH, RETH

Get EtherCAT error

The =ETH and RETH commands are used to assign/report errors and certain status information for the EtherCAT bus.

- Assigned to a program variable: x=ETH(y)
- As a report: RETH(y)

Where y is:

Assignment	Report	Description
=ETH(0)	RETH(0)	Gets the Ethernet status bits: (*Indicates an error bit) 0* Network processor initialization failure Read specific error code from ETH(54); Contact Moog Animatics 1* Network processor configuration failure Typically from an invalid EEPROM setting, or possibly an ETHCTL command parameter error; read specific error code from ETH(54) 2 Reserved 3* Network processor failure Likely due to excessive control power supply noise or ESD event 4 Reserved 5 Reserved 6 Reserved 7 Reserved 8 Reserved 9 Reserved 10 Sync 1 Interrupt active The configuration from the controller uses sync signals, and the system is in Safe-Operational or Operational mode 11 Waiting for sync signals The controller requested the motor to go to Operational mode; it is waiting to receive a Sync 0 and Sync 1 interrupt 12 Received the sync signals The motor sent an acknowledgment to the controller confirming that it is ready to go Operational 13 Reserved 14 Reserved 15 Reserved 16* SDO Error

Assignment	Report	Description
		An error occurred when writing or reading a valid SDO; read specific error code from ETH(20), ETH(21), ETH(22), ETH(23) 17* PDO Error An error occurred when writing or reading a valid PDO; read specific error code from ETH(25), ETH(26), ETH(27), ETH(28) NOTE: Object 2304h, subindex 3, bit 6 (Ethernet error) reports true if any error indications above are set. In a user program, this is a simpler test than attempting to filter the result of RETH for the error conditions.
=ETH(1)	RETH(1)	Gets the value of the current EtherCAT State Machine (ESM) state: 1 Init 2 Pre-Operational 4 Safe-Operational 8 Operational
=ETH(2)	RETH(2)	Gets the value of the Control Word object (6040h)
=ETH(3)	RETH(3)	Gets the value of the Status Word object (6041h)
=ETH(5)	RETH(5)	LFW firmware version as 32-bit integer. E.g., 3.1.0.1 would be a value 50397185 (0x03010001).
=ETH(6)	RETH(6)	The current Network Lost program label number. For details, see ETHCTL (action, value) on page 67.
=ETH(7)	RETH(7)	Processor type: -1 Failed 0 Unknown 1 netX 10 2 netX 50 3 netX 51/52 4 netX 100
=ETH(8)	RETH(8)	Protocol type: 0 Not defined 1 PROFINET 2 EtherCAT 3 EtherNet/IP
=ETH(9)	RETH(9)	The current value assigned to the Network Lost action. For details, see ETHCTL(action, value) on page 67.
=ETH(12)	RETH (12)	Gets status of ETHCTL(12). For details, see ETHCTL(action, value) on page 67.
=ETH(13)	RETH (13)	Gets status of ETHCTL(13). For details, see ETHCTL(action, value) on page 67.
	RETH (18)	MAC ID string formatted; report only. E.g., 00:01:02:a9:ff:00
=ETH(19)	RETH (19)	Report the detected LFW Protocol Class. This gives a wider range of values than the known and supported protocols listed in ETH(8). Values designated according to NXF/LFW file loaded into network processor and are too numerous to list here. These are the values for the supported protocols: (introduced in firmware 6.0.2.41 or later).

Assignment	Report	Description
		0 Not defined 21 PROFINET 9 EtherCAT 10 EtherNet/IP ... Other values may be possible, but only those listed above are supported.
=ETH(20)	RETH (20)	Gets last error code from a SDO read or write; refer to this when ETH(0), bit 16 is indicated
=ETH(21)	RETH (21)	Gets object index from a SDO read or write error; refer to this when ETH(0), bit 16 is indicated
=ETH(22)	RETH (22)	Gets object subindex from a SDO read or write error refer to this when ETH(0), bit 16 is indicated
=ETH(23)	RETH (23)	Gets the direction of the SDO error; 0=read, 1=write; refer to this when ETH(0), bit 16 is indicated
=ETH(25)	RETH (25)	Gets error code from a PDO read or write; refer to this when ETH(0), bit 17 is indicated
=ETH(26)	RETH (26)	Gets object index from a PDO read or write error; refer to this when ETH(0), bit 17 is indicated
=ETH(27)	RETH (27)	Gets object subindex from a PDO read or write error; refer to this when ETH(0), bit 17 is indicated
=ETH(28)	RETH (28)	Gets the direction of the PDO error; 0=read, 1=write; refer to this when ETH(0), bit 17 is indicated
=ETH(30)	RETH (30)	Gets the present receive PDO size in bytes
=ETH(31)	RETH (31)	Gets the present transmit PDO size in bytes
=ETH(32)	RETH (32)	Gets the current state of the error LED: 0 LED off 1 LED permanently on 2 LED flickering 3 LED flickers only once 4 LED blinking 5 LED single flash 6 LED double flash 7 LED triple flash 8 LED quadruple flash 9 LED quintuple flash
=ETH(33)	RETH (33)	Get the Sync control bits: 8 Sync Out unit is activated 9 SYNC0 generation is activated 10 SYNC1 generation is activated 11 Sync Out unit is activated automatically when system time is written NOTE: More than one of these bits can be set during any read
=ETH(34)	RETH	Gets the value of the SmartMotor station alias; a value of zero means

Assignment	Report	Description
	(34)	unused
=ETH(35)	RETH (35)	Gets the SYNC0 cycle time in nanoseconds
=ETH(36)	RETH (36)	Gets the SYNC1 cycle time in nanoseconds
=ETH(37)	RETH (37)	Gets the SYNC1 cycle time in microseconds
=ETH(38)	RETH (38)	Gets the Sync PDI configuration bits: 0 SYNC0 Output type: 0 Push-pull 1 Open-drain/Open-source (depends on bit 1) NOTE: Bit is ignored; they always work as push-pull 1 SYNC0 Polarity: 0 Low active 1 High active 2 SYNC0 Output enable/disable: 0 Disabled 1 Enabled 3 SYNC0 mapped to PDI-IRQ: 0 Disabled 1 Enabled 4 SYNC1 Output type: 0 Push-pull 1 Open-drain/Open-source (depends on bit 5) NOTE: Bit is ignored; they always work as push-pull 5 SYNC1 Polarity: 0 Low active 1 High active 6 SYNC1 Output enable/disable: 0 Disabled 1 Enabled 7 SYNC1 mapped to PDI-IRQ: 0 Disabled 1 Enabled
=ETH(39)	RETH (39)	Gets the last-requested network state
=ETH(40)	RETH (40)	Gets the SYNC1 timeout value in milliseconds
=ETH(41)	RETH (41)	Gets the minimum cycle time in nanoseconds from Object 1C32h, subindex 005.
=ETH(42)	RETH (42)	Gets cycle time in nanoseconds. Measurements of the DC-Sync timing parameters in the SmartMotor is set in ETHCTL(42). For details, see ETHCTL(action, value) on page 67.
=ETH(48)	RETH (48)	Low 3 bytes of MAC ID (device ID) as integer. E.g., for a MACID of 00:01:02:a9:ff:00, this command reports 11140864 (00 a9 ff 00 hex)

Assignment	Report	Description
=ETH(49)	RETH (49)	High 3 bytes of MAC ID (device ID) as integer. E.g., for a MACID of 00:01:02:a9:ff:00, this command reports 258 (00 00 01 02 hex)
=ETH(50)	RETH (50)	Gets the last internal error code
=ETH(51)	RETH (51)	Gets the last internal error code source
=ETH(53)	RETH (53)	Gets the last error code from a request to change states: 17 Invalid requested state change 23 Invalid sync manager configuration 29 Invalid output Configuration 30 Invalid input configuration 43 No valid inputs or outputs configured 45 No Sync error 53 Invalid sync cycle time
=ETH(54)	RETH (54)	Gets the Initialization error code; for further information, read this error when ETH(0), bits 0 or 1 are indicated
=ETH(55)	RETH (55)	Gets the AL status code; see AL-Status Codes on page 24
=ETH(60)	RETH (60)	Gets the state of the homing invert option. See ETHCTL(action, value) on page 67

The ETH(0), ETH(33) ETH(38), RETH(0), RETH(33) and RETH(38) commands are used to report a bit map of information from the EtherCAT bus. For these commands, more than one bit can be set at a time. The commands report a decimal number that is a combination of the bits shown in the previous table.

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 Knowledgebase at:

https://www.animatics.com/support/downloads/knowledgebase/

EtherCAT Network Control Commands

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

ETHCTL(action, value)

Control network features

Commands execute based on the action argument as defined in the next table:

Action =	Description
6	ETHCTL(6,<value>) - Network Lost user program label number. This setting is nonvolatile. Program label to jump to if the Network Lost action, ETHCTL(9,<value>), is either set to 4 or 5.
9	ETHCTL(9,<value>) - Network Lost action. This setting is nonvolatile. 0 – Ignore, no action 1 – Send OFF command to motor (servo off) 2 – Send X command to motor (smooth stop) - default setting 3 – Send S command to motor (hard stop) 4 – Send GOSUB(x) command, where x is the value of the user program label as defined by ETHCTL(6,<value>). 5 – Send GOTO(x) command, where x is the value of the user program label as defined by ETHCTL(6,<value>). NOTE: Loss of network is an edge-triggered event if the state changed from Operational to Safe-Operational without being commanded.
12	ETHCTL(12,<value>) - This action uses these value arguments: • 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	ETHCTL(13,<value>) - This action uses these value arguments: • Value = 0: Disables access to several objects listed below. Clears "remote" bit 9 in the status word 6041 hex. • 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 6041 hex. The list of objects affected are: ◦ 6040h: Control Word ◦ 6060h: Modes of Operation ◦ 6071h: Target Torque ◦ 6072h: Max Torque ◦ 607Ah: Target Position ◦ 6081h: Profile Velocity (pp mode) ◦ 6083h: Profile Acceleration ◦ 6084h: Profile Deceleration ◦ 6087h: Torque Slope ◦ 60FBh: Subindex 1-8,10: PID parameters ◦ 60FFh: Target Velocity

Action =	Description
20-23	Resets the SDO error registers: RETH(20) to RETH(23); clears the SDO error bit in RETH(0). The value argument is ignored.
25-28	Resets the SDO error registers: RETH(25) to RETH(28); clears the SDO error bit in RETH(0). The value argument is ignored.
34	ETHCTL(34,<value>) - Sets the station alias; a value of 0 disables the station alias setting.
42	ETHCTL(42,<value>) - Sets the value of Object 1C32h, subindex 008. Starts measurements of the DC-Sync timing parameters in the SmartMotor. 0 – All measurements cleared 1 – initiates continuous measurements and updates subindexes 5 and 6 with the worst-case measurement.
50-51	Resets the internal error registers: RETH(50) and RETH(51); the value argument is ignored.
53	Resets the state change error register: RETH(53); the value argument is ignored.
55	Resets the AL status error register: RETH(55); the value argument is ignored.
60	Inverts the homing input if set to 1. Value is 0 by default to disable this option. Requires firmware 6.0.2.31 or later

Troubleshooting

This section provides troubleshooting information for solving SmartMotor problems that may be encountered when using EtherCAT. For additional support resources, see the Moog Animatics Support page at:

https://www.animatics.com/support.html

Issue	Cause	Solution
EtherCAT Communication Issues		
Controller does not recognize motor.	Motor not powered.	Check Pwr/servo LED. If LED is not lit, check wiring.
	Unconnected or miswired Ethernet connector, or wiring between motors is broken.	Check that EtherCAT connector is correctly wired and connected to motor. For details, see Connectors and Pinouts on page 1.
	Wrong firmware.	For Class 6 motors, the motor firmware should be: 6.x.2.42 or later and the industrial network communication firmware should be 4.3.18.0 and can be verified with the RSP5 command.
Red EtherCAT error LED.	Hard failure (not typically cleared).	Cycle motor power. If LED remains on (solid) after several power-cycle attempts, then contact product support.
Communication and Control Issues		
Motor control power light does not illuminate.	Control power supply is off or not properly connected.	Check that control power is properly connected and turned on. For details, see Connectors and Pinouts on page 1.
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 USB or 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.

Issue	Cause	Solution
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.
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 a default firmware version closest to the motor's firmware version. In the SMI software, view the motor's firmware version by right-clicking the motor and selecting Properties.

SDO Response Error Codes

The next table shows the list of possible errors (abort codes) from a remote device as defined by EtherCAT 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.
0504 0001h	84148225	Client/server command specifier not valid or unknown.
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.
0601 0003h	100728835	Subindex cannot be written, SI0 must be 0 for write access.

Code		Description
Hex	Dec	
0601 0004h	100728836	SDO complete access not supported for objects of variable length such as ENUM object types.
0601 0005h	100728837	Object length exceeds mailbox size.
0601 0006h	100728838	Object mapped to RxPDO, SDO download blocked.
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.
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 EtherCAT objects used with the Moog Animatics SmartMotor. The TOC below groups the objects by category.

Object Categories

The object descriptions are grouped by these categories:

- Communication Profile on page 76

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

- Manufacturer-Specific Profile on page 100

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

- Drive and Motion Control Profile on page 137

  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 EtherCAT communications. This configures EtherCAT services and PDO behavior.

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 EtherCAT 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 197.

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 that means:

Bit	Function
0	General error Includes any of these: • motion fault • drive not ready • CAN communication errors • program command error • program checksum error • serial communication error
1-7	Reserved

Object 1008h: Manufacturer Device Name

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1008h	000	Manufacturer Device Name			SMClass6	No	String	Read Only

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

Product	Value (string)
Class 6D-Style	SMClass6
Class 6M-Style	SMClass6

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:

Product	Value (string)	Length
Class 6D-Style	`6.4.y.z`	24
Class 6M-Style	`6.0.y.z`	24

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

Similar **SmartMotor** Commands: RFW, RSP (firmware) info

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 (EtherCAT)	00000000h	FFFFFFFFh	00C0FFEEh	No	Unsigned 32-bit	Read Only
1018h	002	Product Code	00000000h	FFFFFFFFh	00000001h	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 EtherCAT Vendor ID number assigned to Moog Animatics: 00C0FFEEh
- Subindex 2 contains the manufacturer-specific product code (varies by product):

Product	Code (EtherCAT)
Class 6 D-Style	1
Class 6 M-Style	1

- 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 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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1601h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60FF0020h	No	Unsigned 32-bit	Read Write
1601h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1602h	001	Mapping Entry 1	00000000h	FFFFFFFFh	607A0020h	No	Unsigned 32-bit	Read Write
1602h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1603h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60710010h	No	Unsigned 32-bit	Read Write
1603h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1604h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60600008h	No	Unsigned 32-bit	Read Write
1604h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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 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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1A01h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60640020h	No	Unsigned 32-bit	Read Write
1A01h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1A02h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60F40020h	No	Unsigned 32-bit	Read Write
1A02h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1A03h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60770010h	No	Unsigned 32-bit	Read Write
1A03h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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	01h	No	Unsigned 8-bit	Read Write
1A04h	001	Mapping Entry 1	00000000h	FFFFFFFFh	60610008h	No	Unsigned 32-bit	Read Write
1A04h	002	Mapping Entry 2	00000000h	FFFFFFFFh	00000000h	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 these items, refer to Mapping Parameters Objects on page 57:

- 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 1C00h: Sync Manager Com Type

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1C00h	0	Number of used Sync Manager channels	04	04	04	No	Unsigned 8-bit	Read Only
1C00h	1	Communication Type Sync Manager 0	00	04	01	No	Unsigned 8-bit	Read Only
1C00h	2	Communication Type Sync Manager 1	00	04	02	No	Unsigned 8-bit	Read Only
1C00h	3	Communication Type Sync Manager 2	00	04	03	No	Unsigned 8-bit	Read Only
1C00h	4	Communication Type Sync Manager 3	00	04	04	No	Unsigned 8-bit	Read Only

This object is a description of the Sync managers available. These fields are read-only and cannot be configured. The next table is a reference for Subindexes 1-4 Default column values in the previous table:

Value	'Type' Meaning
0	Unused
1	Mailbox receive
2	Mailbox send
3	Process data output
4	Process data input

Object 1C12h: Sync Manager 2 PDO Assignment

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1C12h	000	Number of Assignments	0h	05h	02h	No	Unsigned 8-bit	Read Write
1C12h	001	Receive PDO Mapping Object - Index of Assignment Object	1600h	1604h	1600h	No	Unsigned 16-bit	Read Write
1C12h	002	Receive PDO Mapping Object - Index of Assignment Object	1600h	1604h	1601h	No	Unsigned 16-bit	Read Write
1C12h	003	Receive PDO Mapping Object - Index of Assignment Object	1600h	1604h	0000h	No	Unsigned 16-bit	Read Write
1C12h	004	Receive PDO Mapping Object - Index of Assignment Object	1600h	1604h	0000h	No	Unsigned 16-bit	Read Write
1C12h	005	Receive PDO Mapping Object - Index of Assignment Object	1600h	1604h	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of Sync Manager 2 PDO assignments.

This object can be written to only during the Pre-Operational state.

To change subindexes 1-5, first write subindex 0 "Number of Assignments" to the value 0. After updating and writing the information in subindexes 1-5 to the correct information (per the application requirements), then write subindex 0 to the value corresponding to the desired Sync Manager Process Data exchanges.

Many EtherCAT controller applications manage reading and writing Sync Manager PDO assignment objects 1C12h and 1C13h from a graphical user interface (GUI) for easier configuration of the process data. This object's semantics are therefore handled during EtherCAT network startup by the EtherCAT Controller, which removes the burden from the application programmer in many cases.

Object 1C13h Sync Manager 3 PDO Assignment

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1C13h	000	Number of Assignments	0h	05h	02h	No	Unsigned 8-bit	Read Write
1C13h	001	Transmit PDO Mapping Object - Index of Assignment Object	1A00h	1A04h	1A00h	No	Unsigned 16-bit	Read Write
1C13h	002	Transmit PDO Mapping Object - Index of Assignment Object	1A00h	1A04h	1A01h	No	Unsigned 16-bit	Read Write
1C13h	003	Transmit PDO Mapping Object - Index of Assignment Object	1A00h	1A04h	0000h	No	Unsigned 16-bit	Read Write
1C13h	004	Transmit PDO Mapping Object - Index of Assignment Object	1A00h	1A04h	0000h	No	Unsigned 16-bit	Read Write
1C13h	005	Transmit PDO Mapping Object - Index of Assignment Object	1A00h	1A04h	0000h	No	Unsigned 16-bit	Read Write

This object controls the behavior of Sync Manager 3 PDO assignments.

This object can be written to only during the Pre-Operational state.

To change subindexes 1-5, first write subindex 0 "Number of Assignments" to the value 0. After updating and writing the information in subindexes 1-5 to the correct information (per the application requirements), then write subindex 0 to the value corresponding to the desired Sync Manager Process Data exchanges.

Many EtherCAT controller applications manage reading and writing Sync Manager PDO assignment objects 1C12h and 1C13h from a graphical user interface (GUI) for easier configuration of the process data. This object's semantics are therefore handled during EtherCAT network startup by the EtherCAT Controller, which removes the burden from the application programmer in many cases.

Object 1C32h DC-Sync Manager 2 Receive Object

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
1C32h	000	Number of Assignments	-	20h	20h	No	Unsigned 8-bit	Read Only
1C32h	001	Synchronization Type: 0h = free run 3h = DC-Sync with SYNC1	0h	3h	0h	No	Unsigned 16-bit	Read Write
1C32h	002	Cycle Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Write
1C32h	004	Synchronization Types Supported	-	-	0011h	No	Unsigned 16-bit	Read Only
1C32h	005	Minimum Cycle Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Only
1C32h	006	Calc and Copy Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Only
1C32h	007	Minimum Delay Time in nanoseconds	0h	FFFFFFFFh	0h	No	Unsigned 32-bit	Read Only
1C32h	008	Get Cycle Time (Command)	0h	1h	0h	No	Unsigned 16-bit	Read Write
1C332	009	Delay Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Only
1C32h	010	SYNC0 Cycle Time in nanoseconds	0h	FFFFFFFFh	0h	No	Unsigned 32-bit	Read Only
1C32h	011	SM-Event Missed Counter	0h	FFFFh	0h	No	Unsigned 16-bit	Read Only
1C32h	012	Cycle Time to Small Counter	0h	FFFFh	0h	No	Unsigned 16-bit	Read Only
1C32h	032	Sync Error	0h	FFh	0h	No	BOOL (Unsigned 8-bit)	Read Only

[a] Calculated by operating system once DC-Sync is established; see subindex 8.

This object controls the DC-Sync behavior and provides the status of Sync Manager 2 synchronization. The object is updated during the Pre-Operational state to reflect the settings produced by the EtherCAT controller.

- Subindex 2 is updated during the Pre-Operational state if DC-Sync is used. It reflects the DC-Sync timing and the expected SmartMotor update cycle for accepting PDO transfers.
- Subindex 5 needs to be queried from the SmartMotor during application development. If the minimum cycle time produced by the motor does not meet the performance demands of the application, contact Moog Animatics for possible improvement options.
- Subindex 6 is a measured time for PDO transfer within the SmartMotor. It varies with Receive PDO byte size. This subindex reflects the minimum time for the SYNC1 synchronization signal when setting up DC-Sync for the application.

  NOTE: If subindex 5 and 6 appear to be static, then current measurements are better than the previously-captured, worst-case value.

- Subindex 8 is used to start measurements of the DC-Sync timing parameters in the SmartMotor. Setting this subindex to the value 1 initiates continuous measurements and updates subindexes 5 and 6 with the worst-case measurement. The measurements are based on system latencies and PDO sizes. All measurements are cleared by setting this subindex to the value 0.
- Subindex 9 needs to be interrogated from the SmartMotor during application development. It reports the hardware delay within the SmartMotor for completion of a PDO transfer from the EtherCAT network. It varies with Receive PDO and Transmit PDO byte size.

Object 1C33h DC-Sync Manager 3 Transmit Object

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map.	Data type	Access
1C33h	000	Number of Assignments	-	20h	20h	No	Unsigned 8-bit	Read Only
1C33h	001	Synchronization Type: 0h = free run 3h = DC-Sync with SYNC1	0h	3h	0h	No	Unsigned 16-bit	Read Write
1C33h	002	Cycle Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Write
1C33h	004	Synchronization Types Supported	-	-	0011h	No	Unsigned 16-bit	Read Only
1C33h	005	Minimum Cycle Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Only
1C33h	006	Calc and Copy Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Only
1C33h	007	Minimum Delay Time in nanoseconds	0h	FFFFFFFFh	0h	No	Unsigned 32-bit	Read Only
1C33h	008	Get Cycle Time (Command)	0h	1h	0h	No	Unsigned 16-bit	Read Write
1C33h	009	Delay Time in nanoseconds	0h	FFFFFFFFh	Calculated[a]	No	Unsigned 32-bit	Read Only
1C33h	010	SYNC0 Cycle Time in nanoseconds	0h	FFFFFFFFh	0h	No	Unsigned 32-bit	Read Only
1C33h	011	SM-Event Missed Counter	0h	FFFFh	0h	No	Unsigned 16-bit	Read Only
1C33h	012	Cycle Time to Small Counter	0h	FFFFh	0h	No	Unsigned 16-bit	Read Only
1C33h	032	Sync Error	0h	FFh	0h	No	BOOL (Unsigned 8-bit)	Read Only

[a] Calculated by operating system once DC-Sync is established; see subindex 8.

This object controls the DC-Sync behavior and provides the status of Sync Manager 3 synchronization. The object is updated during the Pre-Operational state to reflect the settings produced by the EtherCAT controller.

- Subindex 2 is updated during the Pre-Operational state if DC-Sync is used. It reflects the DC-Sync timing and the expected SmartMotor update cycle for accepting PDO transfers.
- Subindex 5 needs to be queried from the SmartMotor during application development. If the minimum cycle time produced by the motor does not meet the performance demands of the application, contact Moog Animatics for possible improvement options.
- Subindex 6 is a measured time for PDO transfer within the SmartMotor. It varies with Receive PDO byte size. This subindex reflects the minimum time for the SYNC1 synchronization signal when setting up DC-Sync for the application.

  NOTE: If subindex 5 and 6 appear to be static, then current measurements are better than the previously-captured, worst-case value.

- Subindex 8 is used to start measurements of the DC-Sync timing parameters in the SmartMotor. Setting this subindex to the value 1 initiates continuous measurements and updates subindexes 5 and 6 with the worst-case measurement. The measurements are based on system latencies and PDO sizes. All measurements are cleared by setting this subindex to the value 0.
- Subindex 9 needs to be interrogated from the SmartMotor during application development. It reports the hardware delay within the SmartMotor for completion of a PDO transfer from the EtherCAT network. It varies with Receive PDO and Transmit PDO byte size.

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 adhere to a common standard. They provide access to SmartMotor commands and data.

Object 2101h: Bit IO

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2101h	000	Number of Entries	06h	06h	06h	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
2101h	004**	Make Output	0000h	7FFFh	0000h	Yes	Unsigned 16-bit	Read Write
2101h	005**	Enable Special	0000h	7FFFh	0000h	Yes	Unsigned 16-bit	Read Write
2101h	006**	Disable special	0000h	7FFFh	0000h	Yes	Unsigned 16-bit	Read Write

*Starting with firmware 6.0.2.41 or later, if given value is not a supported I/O number, then an error will be returned.

**Subindexes 004, 005, and 006 are only supported on Class 6 motors with 6.x.2.60 or later firmware.

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. See the Installation and Startup Guide for your motor I/O configuration.

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: Drive the specified I/O low.
- 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 a generic input, write the value 2 to subindex 3.
- subindex 4: Configure the specified I/O as an output and disable and special function associated with the I/O.
- subindex 5: Enable the special function associated with the I/O. User program command examples: EIGN(2), EIGN(3), EIGN(6), EIDE(-1), EOBK(-1), EOFT(-1).
- subindex 6: Disable the special function associated with the I/O. User program command examples: EILP, EILN, EISM(6), EIDE(7), EOBK(8), EOFT(9).

For more I/O details, see I/O on page 34.

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 next 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 34.

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.

For more details, see User Variables on page 34.

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

NOTE: This command is only available in Class 6 motors with 6.x.2.58 or later firmware.

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=3       ' The following motor's address.
. . .
' 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.
  IF CAN(4)!=0
    PRINT("Communication failed!",#13)
  ENDIF
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

NOTE: This command is only available in Class 6 motors with 6.x.2.58 or later firmware.

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

NOTE: This feature applies to firmware version 6.0.2.35 and later.

This object controls the behavior of Object 220Ch MFA and Object 220Dh MFD. Refer to the next table.

Bit	Meaning
Bit 0-1	Reserved. Write as 0.
Bit 2	Ramp-up command MFA controller or follower units. Object Object 220Ch MFA (not the serial command MFA) 0: controller 1: follower
Bit 3	Ramp-down command MFD controller or follower units. Object Object 220Dh MFD (not the serial command MFD) 0: controller 1: follower
Bit 4-15	Reserved. Write as 0.

Start/Stop Capability

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

NOTE: This feature applies to firmware version 6.0.2.35 and later.

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*.

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.								

NOTE: This feature applies to firmware version 6.0.2.35 and later.

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*.

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	7FFFFFFFh	0 (disabled)	No	Signed 32-bit	Read Write

NOTE: This feature applies to firmware version 6.0.2.35 and later.

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*.

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	7FFFFFFFh	0 (disabled)	No	Signed 32-bit	Read Write

NOTE: This feature applies to firmware version 6.0.2.35 and later.

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*.

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

NOTE: This feature applies to firmware version 6.0.2.35 and later.

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 116 and Object 2204h: Mappable 32-bit Variables on page 105.

For more details, see User Variables on page 34.

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

NOTE: This feature applies to firmware version 6.0.2.35 and later.

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 115 and Object 2204h: Mappable 32-bit Variables on page 105.

For more details, see User Variables on page 34.

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.

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 reports SmartMotor status word 9
- 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

Status Word 1	Index registration and soft limits
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	USB error
3	Reserved
4	CAN error
5	Reserved
6	Ethernet error
7	IIC communications active
8	Watchdog event
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	Reserved

Status Word 3	PID, brake, move generation
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	Peak current saturation
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	Time 8 event occurred
9	CDFH Drive enabled
10	CDFH Command request timeout
11	CDFH Enabled indication
12	CDFH Group fault
13	CDFH Group ready
14	CDFH Remote fault
15	CDFH Timeout event
CDFH = Combitronic Distributed Fault Handling	

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	MDT Trapezoidal commutation (Hall sensors)
1	MDE Enhanced trapezoidal commutation (encoder)
2	MDS Sinusoidal commutation
3	MDC Current vector FOC mode commutation
4	Reserved
5	Feedback fault
6	MDH mode active
7	Drive enable input fault
8	Electrical angle valid
9	TOB enabled (Torque overrun braking)
10	Invert direction enabled
11	MTB active
12	Encoder battery 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 Word 9	SD Card and DMX Information (Class 6 Only)
0	SD card present
1	SD card busy
2	SD card error
3	SD card valid SMX file
4	SD card valid parameters
5	SD card valid SMXE
6	DMX comm active
7	DMX data received
8	DMX sync event
9	Reserved
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Reserved

Status Word 10	RxPDO Arrival Notification
0	Controller 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

Status Word 10	RxPDO Arrival Notification
15	Reserved
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 controller status, see Network Control Commands on page 1. NOTE: The ZS command will have no effect on these bits.	

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
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

Status Word 13	User-settable status bits (Writable from this object)
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: Class 6D-style: 0-9 I/O: Class 6M-style: 0-9

Status Word 17	I/O: Class 6D-style: I/O: Reserved expansion I/O: Class 6M-style: I/O: Reserved expansion

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 2309h: GOSUB R2

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2309h	000	GOSUB R2	-17*	+999	-1	Yes	Signed 16-bit	Read Write
*The lower limit is -9 for Class 6 models with 6.x.2.59 or earlier firmware.								

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 next 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.
-11*	EIDE(-1)
-12*	EIDE(7)
-13*	EOFT(-1)
-14*	EOFT(9)
-15*	EOBK(-1)
-16*	EOBK(8)
-17*	EISM(6)
*These values are only supported by Class 6 motors with 6.x.2.60 or later firmware.	

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

Object 2310h: Modulo Position

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
2310h	000	Number of Entries	03h	03h	03h (3)	No	Unsigned 8-bit	Read Only
2310h	001	Modulo Position Limit	00000001h	7FFFFFFFh	000003E8h	No	Unsigned 32-bit	Read Write
2310h	002	Modulo Position Actual	00000000h	7FFFFFFEh	00000000h	Yes	Unsigned 32-bit	Read Only

NOTE: This feature is not available in Class 5 SmartMotors. This feature applies to Class 6 firmware version 6.x.2.72 and later.

This object provides read and write access to the Modulo Position Limit and read access to the Modulo Actual Position.

Similar **SmartMotor** Commands: PML=, RPML, RPMA

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	Reserved
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 134.

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 these 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 133.

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 next 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 these 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 these 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 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
2500h	004	Program Output (6.0.2.31 or later)				No	Octet String: 64 bytes	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 35.

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

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 ETHCTL(13,0) command.								

The control word is the primary method of commanding motion in the SmartMotor. The object provides access to these features:

- 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 40.

The SmartMotor =ETH and RETH 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 EtherCAT Error Reporting Commands on page 62.

The next 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 40.
1	Enable voltage	
2	Quick stop	
3	Enable operation	
4	Operation mode specific: "New set-point"	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 40. The SmartMotor =ETH and RETH 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 EtherCAT Error Reporting Commands on page 62.

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 40.
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 40.
6	Switch on disabled	
7	Warning	Not used (reports as 0).
8	Manufacturer-specific	Used by the GOSUB R2 object (2309h) to indicate the subroutine is busy.
9	Remote	Controlled through ETHCTL(13,x). This bit indicates if the motor is accepting commands from the EtherCAT 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 ETHCTL(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 139.

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 164); drive will automatically leave the quick stop state.
2	Decelerate on the quick stop ramp (see Object 6085h: Quick Stop Deceleration on page 165); 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 AniBasic MFD() command controls the deceleration in those cases. For details on that command, see the *SmartMotor™ Developer's Guide*.

Object 605Ch: Disable Operation Option Code

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

This object determines what action should be taken if the Enable Operation bit is cleared in the Control Word object (6040h) while in the operation (enabled drive) state. For details, see Object 6040h: Control Word on page 139.

In Profile Torque (TQ) mode, disable operation option code values 1 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 164)
2-32767	Reserved

If using Follow or Cam mode, be aware that these decelerations are not applied. The AniBasic MFD() command controls the deceleration in those cases. For details on that command, see the *SmartMotor™ Developer's Guide*.

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 139.

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 164)
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 AniBasic MFD() command controls the deceleration in those cases. For details on that command, see the *SmartMotor™ Developer's Guide*.

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	3*	-1	No	Signed 16-bit	Read Write
* Value 2 is not allowed and will return an error in firmware version 6.0.2.41 or later. Value 3 supported in firmware 6.0.2.15 or later.								

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 164)
2	Not supported
3	Decelerate on current limit
4	Not supported
5-32767	Reserved

If using Follow or Cam mode, be aware that these decelerations are not applied. The AniBasic MFD() command controls the deceleration in those cases. For details on that command, see the *SmartMotor™ Developer's Guide*.

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	-3**	10**	0	Yes	Signed 8-bit	Read Write*

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

**The value 0 is allowed and will not return error (but is does not enter a mode of motion). In firmware version 6.0.2.41 or later, error codes will be returned for unsupported modes: -1, 2, 5; also, previous versions inappropriately allowed a wider range than shown.

The type of motion control is selected by setting this object to one of the values shown in the next 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 147.

Value	Motion Control Mode
-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[1]
8	Cyclic Sync Position (CSP) mode
9	Cyclic Sync Velocity (CSV) mode
10	Cyclic Sync Torque (CST) mode
11 to 127	Reserved / not supported

1. This mode is not 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 146.

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 191.

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 149.

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 in firmware version 6.0.2.41 or later; also, previous versions inappropriately allowed a wider range than shown.								

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 ETHCTL(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's intent is 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 doesn't provide actual measurements of torque on the Class 5 D-Style SmartMotor. On the SmartMotors that do support it (Class 5 M-Style, Class 6 M-Style and D-Style), it is only valid while in MDC or MDS commutation mode. MDT or MDE mode operation will produce an undefined result for this value. Class 5 D-Style report the same data as object 6074h. For details, see Object 6074h: Torque Demand Value on page 155.

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

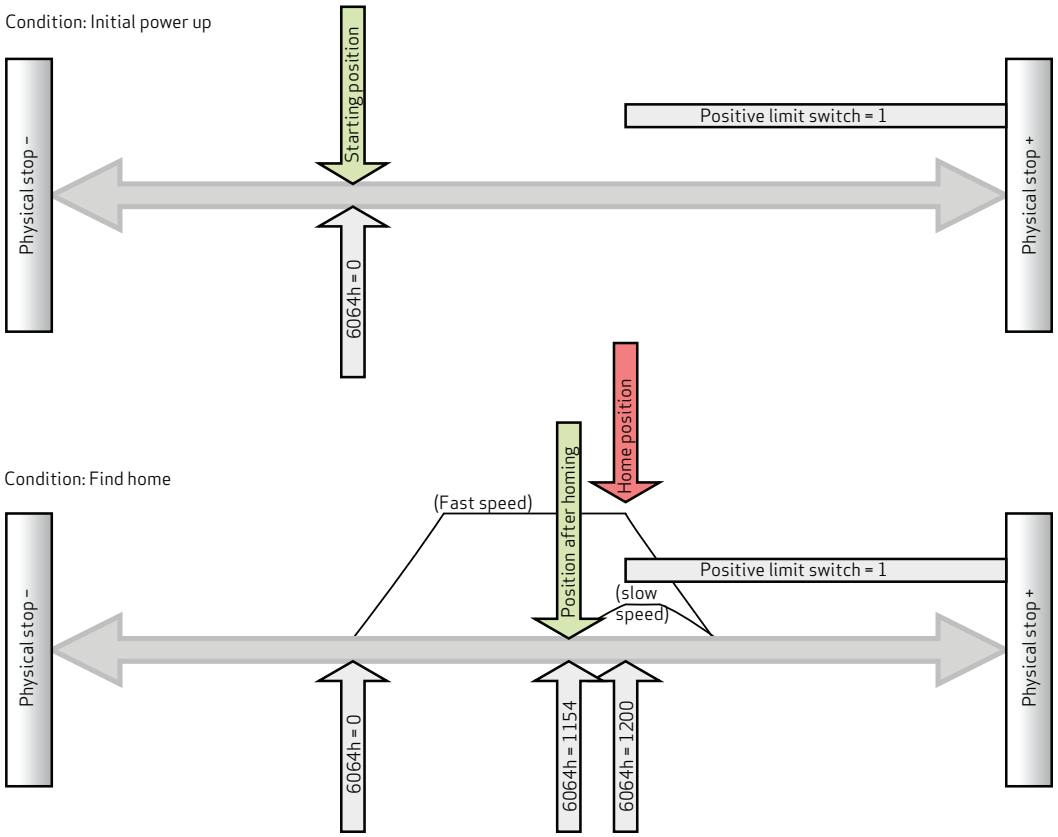
This object shifts the origin of the actual position when the Homing (HM) mode is executed. When HM mode is commanded to begin, the home position is first discovered. The home position is the physical location of the switch or index per the specific homing method. Once found, that physical location is assigned the negative of the home offset value:

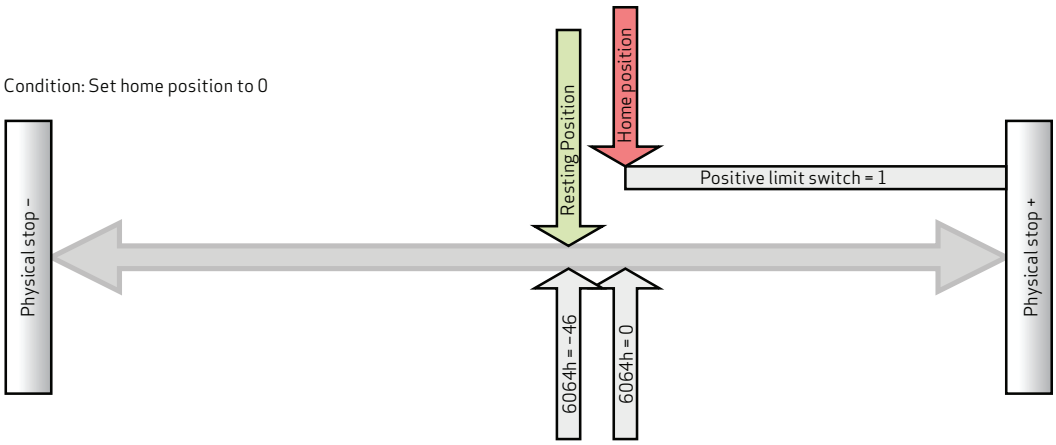
Home position = -Home offset

The home position is assigned with -home offset. See the next example.

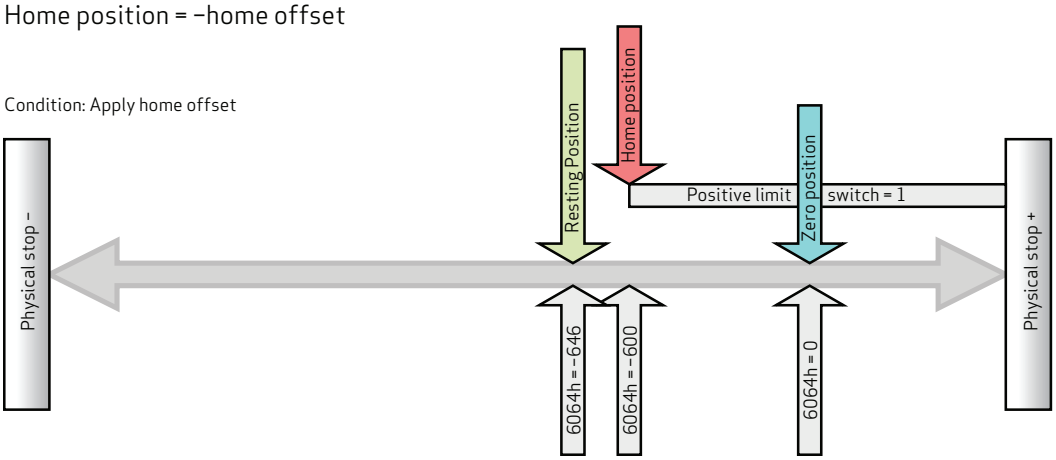
Homing offset object 607Ch = +600
Green: machine physical position
Red: Home Position — where the sensors say it is
Blue: Zero Position — after homing completes, where the machine reports 6064h = 0

Homing method = 18
Incremental encoder (powers up at value = 0)





CiA 402 and ETG guidelines state: “Zero position = home position + home offset”
0 = Home position + home offset
Home position = –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 for each SmartMotor model. For details, see the model 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 ETHCTL(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 195.

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 ETHCTL(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 (sample^2)) * 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 ETHCTL(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 (sample^2)) * 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 (sample^2)) * 65536.

For additional details, see Object 6040h: Control Word on page 139 and Object 605Ah: Quick Stop Option Code on page 142.

Object 6087h: Torque Slope

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
6087h	000	Torque Slope	00000000h	FFFFFFFFh	00F425E8h	Yes	Unsigned 32-bit	Read Write*
*Write access is restricted when the remote bit is cleared with the ETHCTL(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=134213, assuming the default PID rate of 16000 Hz.

For related information, see Object 6071h: Target Torque on page 154.

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 res-olution.	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 *SmartMotor™ Installation and Startup Guide* for your SmartMotor, and the *SmartMotor™ Developer's Guide*.

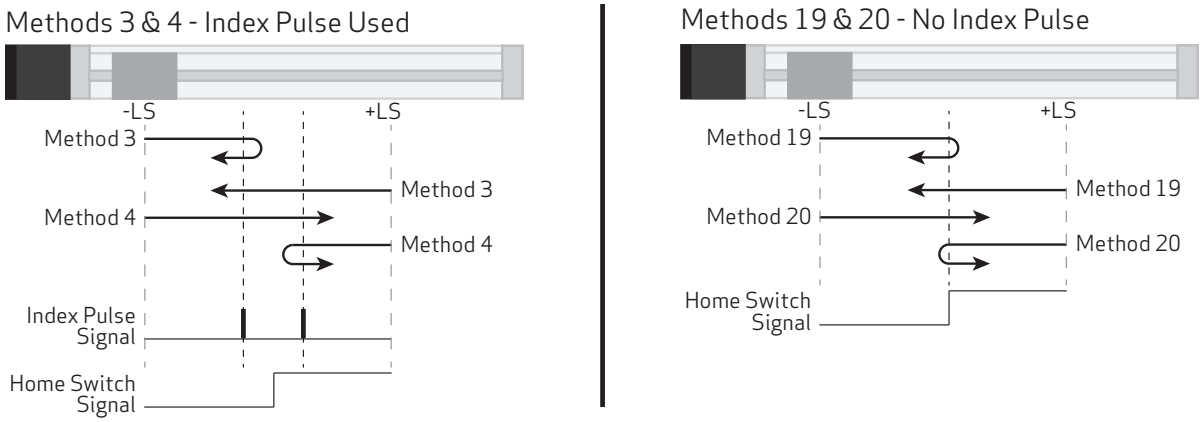
Homing Method Value	Description
1, 2	Home position is the first index in the positive direction from the negative limit switch (1), or in the negative direction from the positive limit switch (2) (requires that limit switches are enabled).
3, 4	Refer to the first figure after this table. Home position is the first index to the left (3) or right (4) from where the home switch changes state in the positive direction.
5, 6	Home position is the first index to the right (5) or left (6) from where the home switch changes state in the negative direction.
7-14	Refer to the second figure after this table. The home switch is active only during part of the travel, similar to a momentary switch. Initial direction for values 7-10 is to the right; initial direction for values 11-14 is to the left. However, if the home switch is active when motion begins, the initial direction depends on the desired signal edge (rising edge or falling edge). The home position will be at the index pulse located on either side of the rising or falling edge as described below: • 7 and 8 will be on the left and right, respectively, of the rising edge in the positive direction; • 9 and 10 will be on the left and right, respectively, of the falling edge in the positive direction; • 11 and 12 will be on the left and right, respectively, of the rising edge in the negative direction; • 13 and 14 will be on the left and right, respectively, of the falling edge in the negative direction; If the initial direction moves the drive away from the home switch, the direction will reverse when the drive reaches the limit switch (requires that limit switches are enabled).
17-30	The homing methods used for these values are similar to those used for values 1-14, except that there is no index pulse used in the process. In other words, each homing method makes use of the home switch and limit switch transitions (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 1-14, 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 1-30 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.

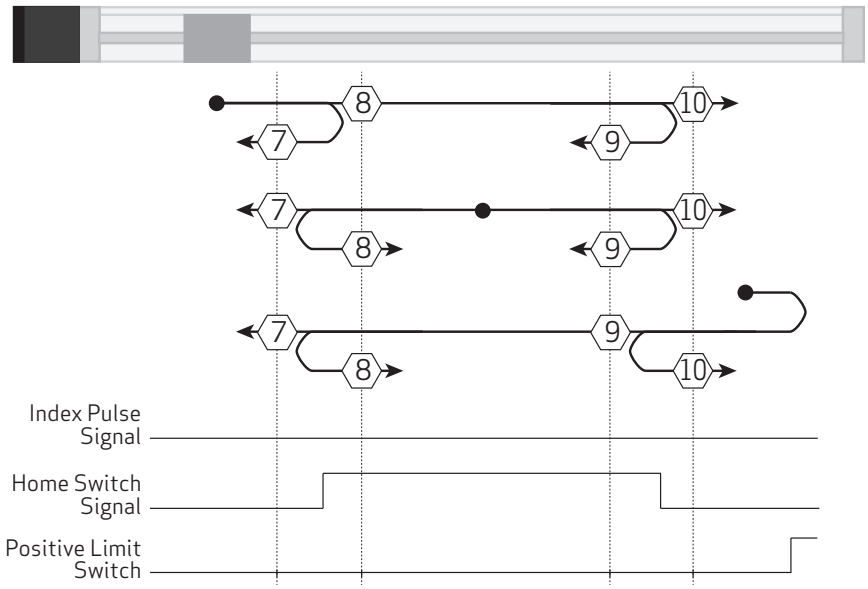
The next figures illustrate the differences between the methods that use an index pulse and those that do not. For example, methods 3 and 4 use an index pulse signal, while methods 19 and 20 do not.



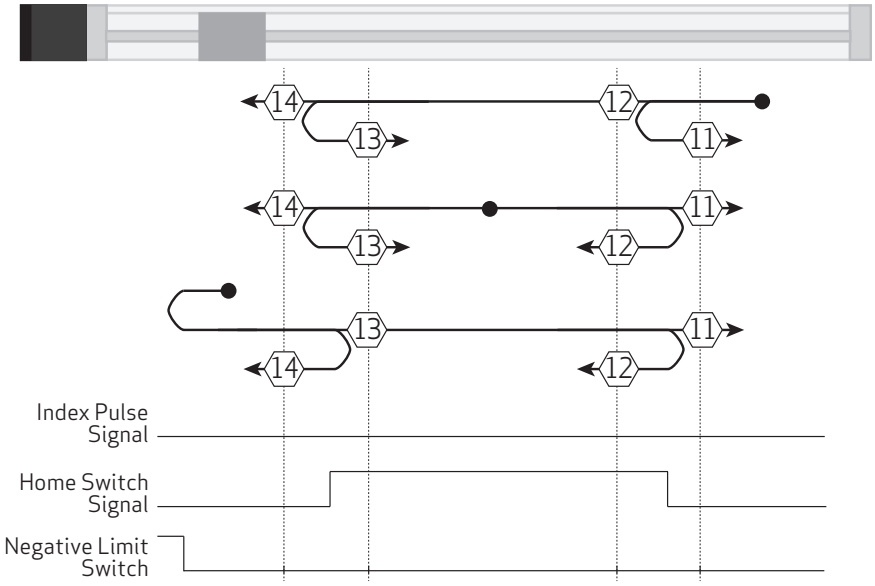
The next figures illustrate homing methods 7-14. Note that:

- 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 (sample^2)) * 65536.

Object 60B8h: Touch Probe Function

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60B8h	000	Touch Probe Function	0	65535	0	Yes	Unsigned 16-bit	Read Write

NOTE: This feature applies to firmware version 6.0.1.9 and later.

The touch probe function allows the motor's position to be captured on a specific event. This feature is commonly used for homing, registration applications or other cases where the motor position must be recorded at a specific point in time. This value can be read back later, in a less time-critical manner, from the capture register.

Object 60B8h is a bit field that can be written to for the purpose of configuring and setting the event trigger (s).

There are two independent touch probes–each has the ability to capture a rising and falling edge. Each of these four possible captures is recorded independently in its own register. For more details, see objects 60BAh, 60BBh, 60BCh, 60BDh.

NOTE: Touch probe 1 always records the value of the internal encoder RCTR(0); touch probe 2 always records the external encoder RCTR(1).

NOTE: When the touch probe is enabled, no changes should be made to the input source selection until the touch probe is disabled.

Also, see Object 60B9h: Touch Probe Status on page 176.

Bit	Touch Probe	Value	Definition
0	TP1	0	Switch off touch probe 1. Also, clears the corresponding captured status bit for any touch probe 1 event.
		1	Enable touch probe 1
1	TP1	0	Trigger on first event
		1	Continuously trigger
3,2	TP1	00	Trigger with touch probe 1's external input–for the Class 6 M-Style or Class 6 D-Style motor, this is general purpose input 5
		01	Trigger with internal encoder's index NOTE: Only the positive edge is supported.
		10	Touch probe source defined by object 60D0:1
		11	Reserved; do not use this state
4	TP1	0	Disable sampling of positive edge of touch probe 1; clears the corresponding captured status bit for this event
		1	Enable sampling of positive edge of touch probe 1
5	TP1	0	Disable sampling of negative edge of touch probe 1; clears the corresponding captured status bit for this event
		1	Enable sampling of negative edge of touch probe 1
6	N/A	0	Reserved; set bit to 0
7	N/A	0	Reserved; set bit to 0

Bit	Touch Probe	Value	Definition
8	TP2	0	Disable touch probe 2; clears the corresponding captured status bit for any touch probe 2 event
		1	Enable touch probe 2
9	TP2	0	Trigger on first event
		1	Continuously trigger
11,10	TP2	00	Trigger with touch probe 2's external input–for the Class 6 M-Style or Class 6 D-Style motor, this is general purpose input 4
		01	Trigger with differential input (using RS-485 port) Class 6 M-Style: (COM 0 differential pair with COM port 0 closed) Class 6 D-Style: (COM 1 differential pair with COM port 1 closed)
		10	Touch probe source defined by object 60D0:2
		11	Reserved; do not use this state
12	TP2	0	Disable sampling of positive edge of touch probe 2; clears the corresponding captured status bit for this event
		1	Enable sampling of positive edge of touch probe 2
13	TP2	0	Disable sampling of negative edge of touch probe 2; clears the corresponding captured status bit for this event
		1	Enable sampling of negative edge of touch probe 2
14	N/A	0	Reserved: set bit to 0
15	N/A	0	Reserved: set bit to 0

To arm a capture, the general enable (bit 0 for touch probe 1) and the rising and/or falling enable must be set (bits 4 and/or 5 for touch probe 1). For example, to capture a single, rising edge of the internal encoder on touch probe 1, follow this sequence:

1. Write value 0 to object 60B8h. This disables both touch probe 1 and touch probe 2 from any events. Any recorded events in the status register (60B9h) will also be cleared. The status register will report 0.
2. Write 21 decimal (15 hex) to object 60B8h. This will arm touch probe 1 to capture the rising edge of the internal encoder's index.
3. Read object 60B9h (touch probe status). If bit 1 is true (1), then the event has occurred. If bit 1 is false (0), then the event has not yet occurred. Therefore, repeat this step.
4. Read object 60BAh (which has become valid with the indication in the status word.) This is the value of position of RCTR(0) when the index event occurred.
5. Write the value 0 to object 60B8h to disable the touch probe feature.

There are two event-capture modes: a single-event mode and a continuous-trigger mode.

- The single-event mode captures the first event (the bit is set when the first capture occurs). It then disarms itself from capturing further events. The data remains valid as long as the corresponding status bit is true. To capture another event, the enable bit(s) must be cleared then reset.
- The continuous-trigger mode continuously captures the events (in other words, it captures each time the index or designated input has an event). The bit will not set until at least one event has occurred. However, there is no further indication as additional events occur. The value read will simply be the most recent position recorded. This mode is disabled by clearing the associated enable bit (e.g., positive edge enable of TP1).

CAUTION: The SmartMotor ZS command can clear the armed touch probe events. Therefore, use caution when operating the touch probes along with the ZS command or other fault-clearing events such as those in object 6040h or 2309h.

Object 60B9h: Touch Probe Status

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60B9h	000	Touch Probe Status	0	65535	0	Yes	Unsigned 16-bit	Read Only

NOTE: This feature applies to firmware version 6.0.1.9 and later.

This object is used to report when there is valid data in any of the four capture registers. If the bit is set, then the corresponding position register can be read as shown in the next table.

There are two event-capture modes: a single-event mode and a continuous-trigger mode.

- The single-event mode captures the first event (the bit is set when the first capture occurs). It then disarms itself from capturing further events. The data remains valid as long as the corresponding status bit is true. To capture another event, the enable bit(s) must be cleared then reset.
- The continuous-trigger mode continuously captures the events (in other words, it captures each time the index or designated input has an event). The bit will not set until at least one event has occurred. However, there is no further indication as additional events occur. The value read will simply be the most recent position recorded. This mode is disabled by clearing the associated enable bit (e.g., positive edge enable of TP1).

NOTE: For either mode, capture registers should not be read until the corresponding bit indicates that data is valid. Refer to the next table.

Bit	Touch Probe	Value	Definition
0	TP1	0	Touch probe 1 is switched off, or no rising or falling events are enabled
		1	Touch probe 1 is enabled (at least one rising or falling edge is enabled and the main enable for touch probe 1 is set)
1	TP1	0	No positive edge yet for touch probe 1
		1	Positive edge position stored for touch probe 1 in object 60BAh
2	TP1	0	No negative edge yet for touch probe 1
		1	Negative edge position stored for touch probe 1 in object 60BBh
3	N/A	0	Reserved
4	N/A	0	Reserved
5	N/A	0	Reserved
6	N/A	0	Reserved
7	N/A	0	Reserved
8	TP2	0	Touch probe 2 is switched off, or no rising or falling events are enabled
		1	Touch probe 2 is enabled (at least one rising or falling edge is enabled and the main enable for touch probe 2 is set)
9	TP2	0	No positive edge yet for touch probe 2
		1	Positive edge position stored for touch probe 2 in object 60BCh
10	TP2	0	No negative edge yet for touch probe 2
		1	Negative edge position stored for touch probe 2 in object 60BDh
11	N/A	0	Reserved

Bit	Touch Probe	Value	Definition
12	N/A	0	Reserved
13	N/A	0	Reserved
14	N/A	0	Reserved
15	N/A	0	Reserved

Object 60BAh: Touch Probe Position 1 Positive Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60BAh	000	Touch probe position 1 positive value	80000000h	7FFFFFFFh	0	Yes	Signed 32-bit	Read Only

NOTE: This feature applies to firmware version 6.0.1.9 and later.

This object is the captured value of RCTR(0) when the positive edge event of touch probe 1 occurs. The capture event is configured in object 60B8h (see Object 60B8h: Touch Probe Function on page 173). This data is only valid if object 60B9h, bit 1 is true (see Object 60B9h: Touch Probe Status on page 176).

Object 60BBh: Touch Probe Position 1 Negative Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60BBh	000	Touch probe position 1 negative value	80000000h	7FFFFFFFh	0	Yes	Signed 32-bit	Read Only

NOTE: This feature applies to firmware version 6.0.1.9 and later.

This object is the captured value of RCTR(0) when the negative edge event of touch probe 1 occurs. The capture event is configured in object 60B8h (see Object 60B8h: Touch Probe Function on page 173). This data is only valid if object 60B9h, bit 2 is true (see Object 60B8h: Touch Probe Function on page 173).

Object 60BCh: Touch Probe Position 2 Positive Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60BCh	000	Touch probe position 2 positive value	80000000h	7FFFFFFFh	0	Yes	Signed 32-bit	Read Only

NOTE: This feature applies to firmware version 6.0.1.9 and later.

This object is the captured value of RCTR(0) when the positive edge event of touch probe 2 occurs. The capture event is configured in object 60B8h (see Object 60B8h: Touch Probe Function on page 173). This data is only valid if object 60B9h, bit 9 is true (see Object 60B9h: Touch Probe Status on page 176).

Object 60BDh: Touch Probe Position 2 Negative Value

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60BDh	000	Touch probe position 2 negative value	80000000h	7FFFFFFFh	0	Yes	Signed 32-bit	Read Only

NOTE: This feature applies to firmware version 6.0.1.9 and later.

This object is the captured value of RCTR(0) when the negative edge event of touch probe 2 occurs. The capture event is configured in object 60B8h (see Object 60B8h: Touch Probe Function on page 173). This data is only valid if object 60B9h, bit 10 is true (see Object 60B9h: Touch Probe Status on page 176).

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 these ways:

- Linear interpolation (default): generates a path of 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 a position overshoot can occur. While this is generally avoided in the algorithm, extreme cases will overshoot.

The next 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 next 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 these 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 these 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	01h	No	Unsigned 8-bit	Read Only
60C1h	001	Data Record 1	80000000h	7FFFFFFFh	00000000h	Yes	Signed 32-bit	Read Write

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.

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 next 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 these 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 these 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 these 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 next 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 60D0h: Touch Probe Source

Object	Sub-Index	Description	Low Limit	High Limit	Default	PDO Map	Data type	Access
60D0h	000	Number of Entries	2	2	2	No	Unsigned 8-bit	Read Only
60D0h	001	Touch Probe 1 Source	1	6	5	Yes	Signed 16-bit	Read Write
60D0h	002	Touch Probe 2 Source	1	6	3	Yes	Signed 16-bit	Read Write

NOTE: This feature applies to firmware version 6.0.1.9 and later.

This object is used to select the external input that is applied to the specified touch probe. The selected input becomes the trigger source for initiating the capture of encoder data to the specific touch probe.

NOTE: The input source must be chosen before enabling the corresponding touch probe. After the touch probe is enabled, do not change the input-source selection until the touch probe is disabled.

Touch Probe 1: 60D0h, subindex 1

Value	Definition (Class 6 M-Style and Class 6 D-Style motor type)
1	Use single-ended input, general-purpose input 5
2	Not supported
3	Not supported
4	Not supported
5	Use internal encoder's index (support for rising edge only)
6	Not supported

Touch Probe 2: 60D0h, subindex 2

Value	Definition (Class 6 M-Style and Class 6 D-Style motor type)
1	Use single-ended input, general-purpose input 5
2	Use single-ended input, general-purpose input 4
3	Use differential input (using RS-485 port) Class 6 M-Style: (Uses COM0 pins, requires closing the COM 0 RS-485 port) Class 6 D-Style: (Uses COM1 pins, requires closing the COM 1 RS-485 port)
4	Not supported
5	Not supported
6	Not supported

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 (position) 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	From EEPROM	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 Feedforward 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	Write Only*
*Write access is restricted when the remote bit is cleared with the ETHCTL(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 next 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 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).

Class 6M-Style motor	
Bit	**Function**
0	Negative limit asserted (if enabled)
1	Positive limit asserted (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 (vendor-specific positive limit)
19	General purpose input 3 (vendor-specific negative limit)
20	General purpose input 4
21	General purpose input 5 (SYNC-encoder capture input)
22	General purpose input 6
23	Vendor Specific Drive Enable State (input 7)
24	Brake (reports intended state of output 8)
25	Not faulted (reports intended state of output 9)
26-31	Reserved

Class 6 D-Style motor	
Bit	**Function**
0	Negative limit asserted (if enabled)
1	Positive limit asserted (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 (vendor-specific positive limit)
19	General purpose input 3 (vendor-specific negative limit)
20	General purpose input* 4
21	General purpose input* 5 (SYNC-encoder capture input)
22	General purpose input 6
23	Vendor Specific Drive Enable State (input 7)
24	Brake output function by default / input* 8
25	Not faulted function by default / input* 9
26-29	Reserved
30	Input 14 (STO 1)
31	Input 15 (STO 2)

*These inputs: 4, 5, 8, 9, have output drivers, also. The input functionality is always available and reports based on voltage at the pin. Therefore, when outputs are active, the input effectively provides feedback information about the output driver state.

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.

Class 6 M-style motor	
Bit	**Function**
0	Brake Set - Not Supported
1-15	Reserved
16-19	Reserved
20	Output 4 (MT2 models only, not supported in MT)
21	Output 5 (MT2 models only, not supported in MT)
22-23	Reserved
24	Output 8. See EOBK command to enable this general purpose output. This output is controlled by brake function by default.
25	Output 9. See EOFT command to enable this general purpose output. This output is controlled by notFault status by default.
26-31	Reserved

Class 6D-Style motor	
Bit	**Function**
0	Brake Set - Not Supported
1-15	Reserved
16-19	Reserved
20	Output 4
21	Output 5
22-23	Reserved
24	Output 8. See EOBK command to enable this general purpose output. This output is controlled by brake function by default.
25	Output 9. See EOFT command to enable this general purpose output. This output is controlled by notFault status by default.
26-31	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 ETHCTL(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 162.

Similar **SmartMotor** Commands: VT=, RVT

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	Varies by motor class and version	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	Mode
0	Profile Position (PP)
1	Velocity (VL)
2	Profile Velocity (PV)
3	Torque (TQ)
4	Reserved
5	Homing (HM)
6	Interpolation (IP)
7	Cyclic Synchronous Profile (CSP)
8	Cyclic Synchronous Torque (CSV)
9	Cyclic Synchronous Torque (CST)
10-15	Reserved
16-31	Manufacturer specific
Bit value 0: Not supported Bit value 1: Supported	

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 next 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 77.

Reference Documents

These CiA documents were referenced for this guide:

- CiA 402 CANopen - Drives and motion control device profile:

  This specification is now comprised of these 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):

https://www.can-cia.org/

The IEC documents are maintained by the International Electrotechnical Commission (IEC):

https://www.iec.ch/homepage

These EtherCAT Technology Group (ETG) documents were referenced for this guide:

- ETG.1020 S (R) V1.0.0 - EtherCAT Protocol Enhancements
- ETG.1300 S (R) V1.1.0 - EtherCAT Indicator and Labeling
- ETG.6010 D (R) V1.0.0 - EtherCAT Implementation Directive for CiA402 Drive Profile
- EtherCAT Communication - Communication Principles

The EtherCAT documents are maintained by the EtherCAT Technology Group (ETG):

https://www.ethercat.org/default.htm

TAKE A CLOSER LOOK

Moog Animatics, a sub-brand of Moog Inc. since 2011, is a global leader in integrated automation solutions. With over 30 years of experience in the motion control industry, the company has U.S. operations and international offices as well as a network of Automation Solution Providers worldwide.

Moog Animatics
1995 NC Hwy 141
Murphy, NC 28906
United States

Email: animatics_sales@moog.com

For Animatics product information, visit **www.animatics.com**

For more information or to find the office nearest you, email **animatics_sales@moog.com**

Moog Animatics Class 6 SmartMotor™ EtherCAT Guide, Rev. L
SC80100002-001